

**Developing Cloud and Shared Data Capabilities to
Support Primary School Students in Creating
Mobile Applications that Affect Their Communities**

by

Natalie Lao

S.B., Massachusetts Institute of Technology (2016)

Submitted to the Department of Electrical Engineering and
Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

© Massachusetts Institute of Technology 2017. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
May 18, 2017

Certified by
Harold Abelson
Class of 1922 Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by
Leslie Kolodziejki
Chairman, Department Committee on Graduate Theses

Developing Cloud and Shared Data Capabilities to Support Primary School Students in Creating Mobile Applications that Affect Their Communities

by

Natalie Lao

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2017, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

As technology and society become further intertwined, it is imperative that we democratize the creation of technology and educate people to be capable of harnessing the full power of computational thinking. As such, developing meaningful tools and curricula for incremental learning of computational thinking concepts starting in primary education is an important endeavor [1]. My work focuses on making Cloud technology, one of the most powerful new computer science concepts, understandable and usable by anyone without the need for extensive computer science training.

I used MIT App Inventor, a blocks-based mobile application development tool for teaching computational thinking to young students, as the platform for my research. I developed CloudDB, a set of coding blocks for MIT App Inventor that allows users to store, retrieve, and share various types of data in tag-value pairs on a Redis server for their mobile applications. I created middle and high school level curricula based on CloudDB along with assessment tools to evaluate my materials and the extent to which young students can understand and utilize the concepts around shared data. Finally, I ran one of those workshops with middle school students in the MIT area.

My findings indicate that teaching shared data as a core computational thinking concept is entirely feasible to students as young as middle school level. Students are capable of inferring and extrapolating other use cases and potential problems with the Cloud, such as storage limits and security concerns. When given the context of solving a problem in their lives, they are very driven and able to design and create complex independent mobile application projects using MIT App Inventor and CloudDB.

Thesis Supervisor: Harold Abelson

Title: Class of 1922 Professor of Computer Science and Engineering

Acknowledgments

I thank members of the MIT App Inventor team for all of their knowledge and support throughout the past year. I have learned more than can be expressed in this thesis about the depths of computer science and education from all of you.

Specifically, I thank my advisor Hal Abelson for inspiring me and always pushing me to seek out the answers to the next question on the horizon; Evan Patton for his infinite patience throughout my coding struggles; Michael Tissenbaum, Josh Sheldon, and Karen Lang for their wisdom regarding educational research and working with children; Andrew McKinney and Jeff Schiller for their technical expertise, especially in helping me navigate the magical murky waters of network operations; my undergraduate research assistant Graeme Campbell for his crucial programming contribution with CloudDB security; Xinyue Deng for powering through the thesis writing process with me; and Marisol Diaz for feeding me and making sure that everything is always in order.

Finally, my work would not have been possible without the numerous computer scientists and educators who paved the way beforehand.

Contents

1	Introduction: Primary school students can use computational thinking to create technology that affects their world	15
1.1	Computational thinking has become increasingly prevalent in K-12 education	16
1.2	MIT App Inventor is a blocks-based programming tool for creating mobile applications	20
1.3	The Cloud is a powerful concept that allows people to have worldwide influence cheaply and quickly	23
1.4	Relevant Prior Work: Shariables and Cloud data-structures implemented Scratch enabled computational explorations of online data . .	25
1.5	The publish-subscribe messaging pattern allows an intermediary broker to send messages from publishers to invisible subscribers	27
2	Coding Methodology: CloudDB was implemented using the Redis API with simplicity as the design goal	29
2.1	Design Requirements: CloudDB must be easy to use and similar to FirebaseDB	30
2.2	CloudDB has two properties and fourteen blocks	31
2.3	Simple key-value formatting in the Redis database allows for flexibility in data sharing	35
2.4	Atomic operations were written in Lua script	37
2.5	Semantic change to list: Lists stored in CloudDB are passed by value instead of by reference	38

2.6	Data is encrypted through a SSL proxy server that communicates with a local Redis server through TCP	39
2.7	MusicShare: Example of a multi-functional app built using CloudDB	42
3	Teaching Methodology: Curriculum was created for a 6-lesson workshop for middle school students	45
3.1	Participants in the workshop had little to no experience with programming	46
3.2	A lesson plan was created summarizing the schedule, concepts taught, and data collected	47
3.3	Data Collected	53
4	Results: Data from the workshop shows that young students can quickly understand shared data	55
4.1	Students quickly understood high level ideas and strong role of shared data in modern society	56
4.2	Creating a class-wide shared drawing application successfully helped most students learn CloudDB	60
4.3	The final project design process indicated increased understanding of capabilities of shared data and strong interest in creating apps that influence the community	63
4.4	Case study of one group's progression of constructing knowledge about CloudDB and shared data	66
4.5	Students were generally enthusiastic and optimistic about creating apps that would be used by their peers	71
4.6	Post-questionnaires indicated high levels of satisfaction with making independent, creative apps	74
4.7	Workshop was generally effective but could have been longer to allow for more complete final projects	79
5	Discussion: The Cloud has powerful implications for computational	

thinking education for young students	81
6 Future Works: Improving security, providing access restrictions to project data, developing log viewing capabilities, and extensions to educational research	85
A Lua code for CloudDB atomic list functions	89
B Assent and Consent forms for data release to research	93
B.1 Minor Assent Form	93
B.2 Parental Consent Form	96
C Supplemental Workshop Handouts	101
C.1 Pre-questionnaire	101
C.2 CloudDB Handout	103
C.3 Draw Together App Design Worksheet	106
C.4 Draw Together Solution and Extension	109
C.5 CloudDB Individual Design Project Worksheet	112
C.6 CloudDB Group Design Project Worksheet	115
C.7 Design Project Peer Feedback Worksheet	119
C.8 Post-questionnaire	121

List of Figures

1-1	An MIT App Inventor 2 project is built using both a designer interface and a blocks interface. The designer interface (top) allows users to drag-and-drop and customize components such as buttons onto a mockup Android screen. The blocks interface (bottom) provides code blocks that can be arranged to form basic programs. In this example, a sound file is played with the button is clicked.	21
1-2	The FirebaseDB component currently available on MIT App Inventor has four properties set in the designer (left) and 12 blocks for programming.	22
1-3	Like many other Cloud computing services, AWS sales has shown exponential growth for the past decade [16].	25
1-4	Programmer interactions for using Cloud variables in Scratch 2.0 [7]. A variable can be set to be a Cloud variable by selecting the Cloud variable checkbox. A Cloud variable has the same methods/blocks as a normal variable.	26
2-1	A client running a CloudDB MIT App Inventor application first communicates its request to the proxy server through the TLS protocol which communicates with the local Redis server to handle any requests.	30
2-2	Although both of these blocks operate on the <i>value</i> returned by <i>when CloudDB.GotValue</i> , all edits are only local. Neither actually modify the list stored on the CloudDB server.	37

2-3	The proxy server at startup has one supervisor process and one outward-facing acceptor worker process.	40
2-4	A client connects to the proxy service via TLS. The SSL handshake is performed and the acceptor process instructs the supervisor process to spawn a client handler process as well as a Redis handler process. . .	41
2-5	The client now sends information to the client handler process, which forwards it to Redis. The Redis handler listens for responses from Redis and forwards them back to the client. The acceptor process waits for other clients to attempt to make new connections to the proxy.	41
2-6	Multiple clients connecting simultaneously to the proxy service, each with their own individual client handler process and Redis handler process.	42
2-7	Screenshots and CloudDB code from MusicShare, an example CloudDB app that lets users create, share, and listen to music.	44
4-1	A student's response to Pre-questionnaire, <i>In your own words or with pictures, describe what you think the Cloud is and what it does. Try not to use the examples from today.</i>	58
4-2	Code from the finished DrawTogether class-wide drawing app	62
4-3	Designer screen and CloudDB code from a two-player fighting game that was created for a group's final project	70
4-4	Student presenting final project on social media picture sharing and best city ranking app	73
4-5	Student presenting final project on oil spill cleanup game	74
4-6	A student's response to Post-questionnaire, <i>In your own words or with pictures, describe what you think the Cloud is and what it does. Try not to use the examples from today.</i>	77
6-1	A sample cloud data log in Scratch, which displays when, how, and by whom a cloud variable was modified [29].	86

List of Tables

2.1	An overview of CloudDB's two properties and fourteen blocks	35
3.1	Lesson 1 Plan, Introducing MIT App Inventor	48
3.2	Lesson 2 Plan, Introducing the Cloud	50
3.3	Lesson 3 Plan, CloudDB Draw Together Wrapup	51
3.4	Lesson 4 Plan, CloudDB Independent Design Project Workday 1	52
3.5	Lesson 5 Plan, CloudDB Independent Design Project Workday 2	53
3.6	Lesson 6 Plan, CloudDB Independent Design Project Final Presentations	53
4.1	Mean and range of responses for Pre-questionnaire Likert scale questions on a 1-5 scale.	57
4.2	App designs from individual design project worksheets.	65
4.3	Mean and range of responses for Post-questionnaire Likert scale questions on a 1-5 scale	75

Chapter 1

Introduction: Primary school students can use computational thinking to create technology that affects their world

Shared data is a new concept that has vastly changed the technological landscape over the past decade. It has led to innovations such as Dropbox, Google Docs, and essentially every modern scalable web service. While this generation of professional computer scientists often has trouble understanding and designing applications using Cloud data sharing due to its many operational complexities, I believe that young students who have grown up surrounded by technology will be able to understand and contribute to Cloud technology more quickly, easily, and adeptly. The objectives of this thesis are to learn if it is possible to teach concepts about shared data to students as young as middle school level and, if so, how to empower them to harness this technology to solve problems that they encounter in their lives. I believe that showing young students the significant impact they can have by being creators of technology instead of just consumers of it is the best way to guide them in sculpting the increasingly digital future.

In this work, I contribute (1) an approach for creating a technical system with simplicity and usability as its design goals that allows kids to interact with shared data within reasonable abstraction, (2) a detailed implementation of the working tool, which is a component called CloudDB within the MIT App Inventor web interface created using the Redis database, cache and message broker, (3) a set of teaching materials surrounding CloudDB and MIT App Inventor that aims to help middle school level students understand and utilize the power of shared data, (4) results of a workshop conducted around my curriculum, and (5) discussion around what is easy and difficult for young people to understand and how researchers and educators can think about letting kids have such a powerful ability.

In this chapter, I present the core technical concepts and previous work done in the computational thinking space upon which this thesis is built. I present frameworks for computational thinking, a description of MIT App Inventor as a suitable method for delivering computational thinking to primary school students, and an argument for why the Cloud is an essential concept to teach in early education. In chapter two, I describe the method used in designing and implementing the CloudDB component within MIT App Inventor using the Redis data structure store. In chapter three, I describe the design of the workshop curricula used to evaluate the CloudDB component and the knowledge that students are able to gain. In chapter four, I present the results of the workshop I ran for this thesis. In chapter five, I discuss my work in the context of computational thinking education and the implications it has for different parties. I conclude this thesis in chapter six by offering some future extensions of and questions that arose from my work.

1.1 Computational thinking has become increasingly prevalent in K-12 education

Computational thinking is the method of framing a problem and problem-solving that arises from the core principles of computing [2]. Aside from its influence on society

through computer science, computational thinking has led researchers to create innovative solutions for challenges from sequencing the human genome to microeconomics [3].

In 2012, Karen Brennan and Mitchel Resnick established a framework for studying and assessing the development of computational thinking. They proposed a three-pronged approach to considering computational thinking through: (1) concepts, which refer to technical programming concepts; (2) practices, or good programming methodology; and (3) perspectives, which describe how an individual relates to the technological world [4]. Brennan and Resnick analyzed their framework in the live Scratch environment, a blocks-based programming language that enables young students to create a wide array of stories, animations, and games. They outlined seven categories for computational thinking concepts: sequences, loops, parallelism, events, conditionals, operators, and data; four categories for practices: being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing; and three for perspectives: expressing, connecting, and questioning. They also detailed the difficulties around the assessment of computational thinking, especially in detecting how and if students' perspectives about themselves or their world have changed.

The curriculum and assessments that I created draw heavily on the ideas presented by Brennan and Resnick. With consideration to their framework: *shared* data is my computational thinking concept; debugging, reflection, meta-cognition, generating new ideas, and new avenues for exploration are the key computational thinking practices involved in learning the concept; and understanding shared databases as a basis for shared applications and feeling empowered to use shared data to create technology are the computational thinking perspectives. In order to assess the efficacy of my tool in empowering young people to interact with and use Cloud technology, I ran a Cloud app creation workshop, during which I collected surveys on attitudes towards the Cloud and analyzed students' responses to various design scenarios.

In recent years, computational thinking has gained funding and traction in primary education internationally due to the emergence of a generation more reliant on technology than ever before [12]. Researchers and teachers have also shown that it is

both possible and valuable to teach students computational thinking at a young age.

Two large surveys of overall computational thinking education that were conducted in the past decade have gained much traction. In Lee, Martin, and collaborators' 2011 paper detailing a set of NSF-supported STEM programs, it was found that existing definitions of computational thinking can be applied to K-12 settings for students from a wide range of technical and socioeconomic backgrounds [2]. To model and understand the specific values that students took away from computational thinking lessons, they developed a three-prong framework: (1) Abstraction, applying concepts from one use case to another; (2) Automation, using a computer to execute a set of repetitive tasks more efficiently than a human; and (3) Analysis, a reflective process of the students' assumptions and implementations. They also posited that creating cell phone apps has the potential to develop computational thinking in K-12 students due to the heavy application of all three aspects of their framework throughout that process.

More recently, Duncan and Bell analyzed a variety of English, Australian, and CSTA curricula that had been published for primary schools. They established the main topics covered in all of these curricula and the suitability of the material for first year to eighth year primary school students. They also emphasized the importance of teacher development—once teachers understood why computational skills were important to teach, they were more comfortable and confident with teaching the material.

One key takeaway from this study was that the content of most classes from the curricula analyzed focused on programming skills and data representation knowledge instead of developing more intrinsic perspectives, such as empowering students to be technological creators. The researchers had hoped that computational thinking skills would be taught indirectly via the learning of computer skills, but found that this was generally not the case. Duncan and Bell posited that computational thinking needs to be explicitly taught, which is difficult without first giving students the necessary technical background knowledge.

Grover and Pea then used MIT App Inventor to conduct a pedagogical study of

how to introduce computational thinking concepts in middle schools [14]. Based on the case studies they analyzed, a pilot curriculum was designed and implemented with the following goals [13]:

- Students are engaged with the content presented and enjoy the classes.
- Students become familiar with the basics of programming, rather than just learning to use a specific programming tool/language.
- Students become familiar with the concept of binary number representation.
- Students develop some basic Computational Thinking skills.
- Students have a personal sense of achievement at the end of the course, have opportunities to be creative, and have some form of control over their work.
- Students are able to share their success outside of the classroom with their friends and family.
- Extension activities are available for those that need it, but the main course content is suitable for all students.

Through the workshops that they ran, they concluded that there should be a much heavier emphasis on social interaction in the development of individual mental processes in computational thinking education. Specifically, there is a need to develop communicative activities during classes so that students can foster better individual and group learning. Grover and Pea felt that ideas of classroom discourse should be brought in from the learning sciences to teach introductory computational concepts. In concert with computationally rich activities, discussions and reflection sessions with peers significantly influenced the organic introduction and use of new vocabulary as well as important foundational computing concepts that learners were previously unaware of. In the workshop that I ran for my thesis, I included reflection and peer-to-peer communication activities whenever a new concept was introduced. I also created a final group design project activity.

1.2 MIT App Inventor is a blocks-based programming tool for creating mobile applications

MIT App Inventor is a free and open-source webservice that allows users to create Android mobile applications through a blocks-based programming language and high level abstractions (see Figure 1-1). Since its development began in 2009, the primary goal of MIT App Inventor has been to democratize the creation of mobile applications by allowing people with little to no programming experience to create highly functional apps. Through MIT App Inventor, users can build apps that interface with sensors in their smartphone and the web as a whole, including the smartphone camera, accelerometer, text messaging, voice calls, external Bluetooth/IoT devices, and internet connectivity. Due to its low barrier of entry, MIT App Inventor is being used as the primary teaching tool for many introductory programming classes and workshops targeting students anywhere from late elementary school to university and professional level. This web interface is available worldwide, with 5 million users from 195 countries that have created a total of 16.2 million apps [5]. MIT App Inventor's accessibility to young students and popularity as an international computer science teaching tool makes it a great platform for my research on computational thinking education.

MIT App Inventor currently has one experimental component called FirebaseDB that allows users to interact with the Cloud [6]. Users can create apps using the component that store and access a large pool of shared data across multiple devices. This component relies on the Firebase platform, a cloud data store service provided by Google to Android application developers. Google has recently indicated that they are pursuing a change in direction for Firebase and are planning to tightly integrate the Firebase database service with the Google Play Store, in effect requiring developers who wish to use the Firebase database to register their applications on the Play Store. This is incompatible with much of MIT App Inventor's user base. A large portion of the user population is primarily focused on education and are generally not professional developers. These users distribute the applications they write on MIT

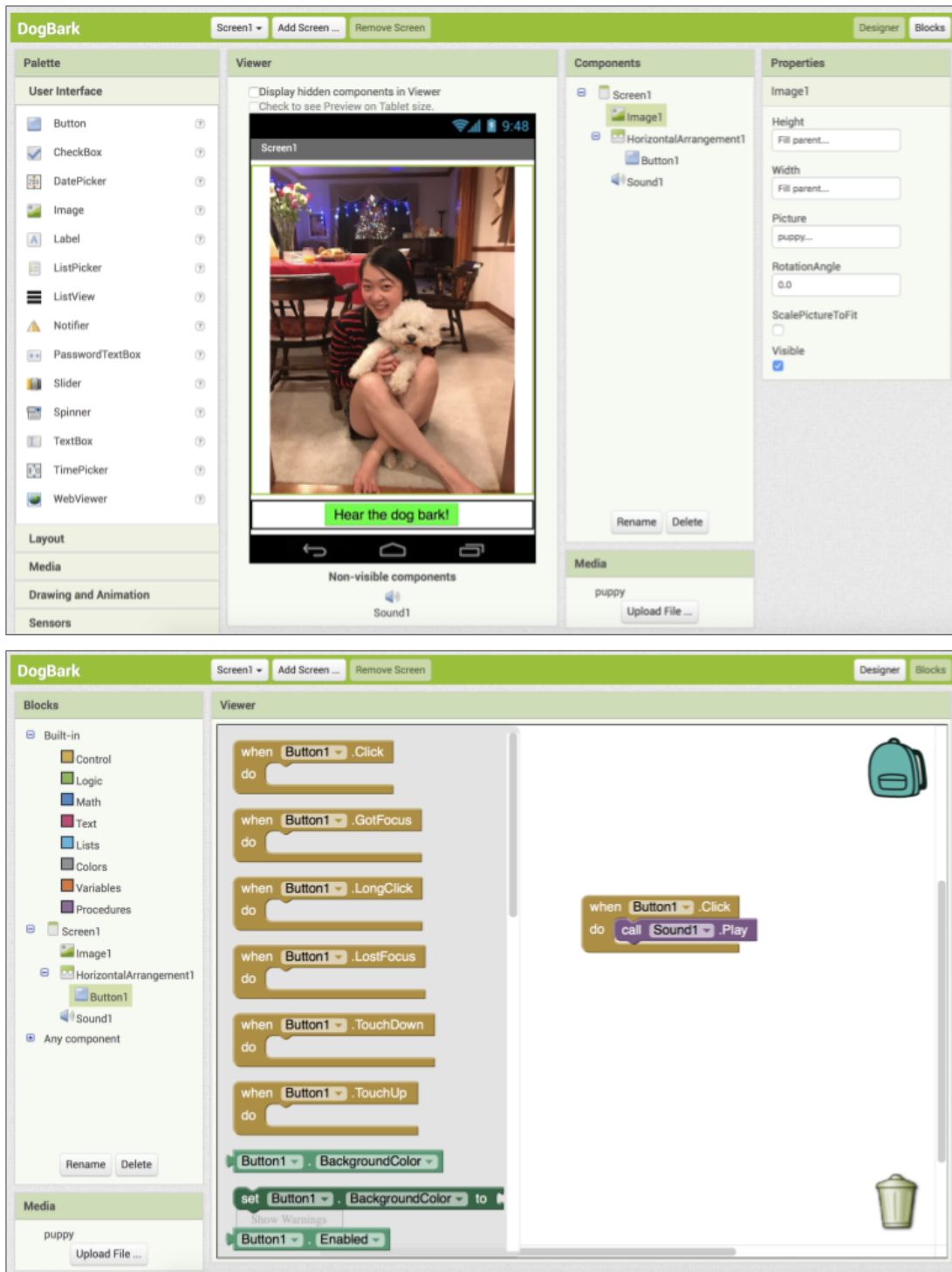


Figure 1-1: An MIT App Inventor 2 project is built using both a designer interface and a blocks interface. The designer interface (top) allows users to drag-and-drop and customize components such as buttons onto a mockup Android screen. The blocks interface (bottom) provides code blocks that can be arranged to form basic programs. In this example, a sound file is played with the button is clicked.

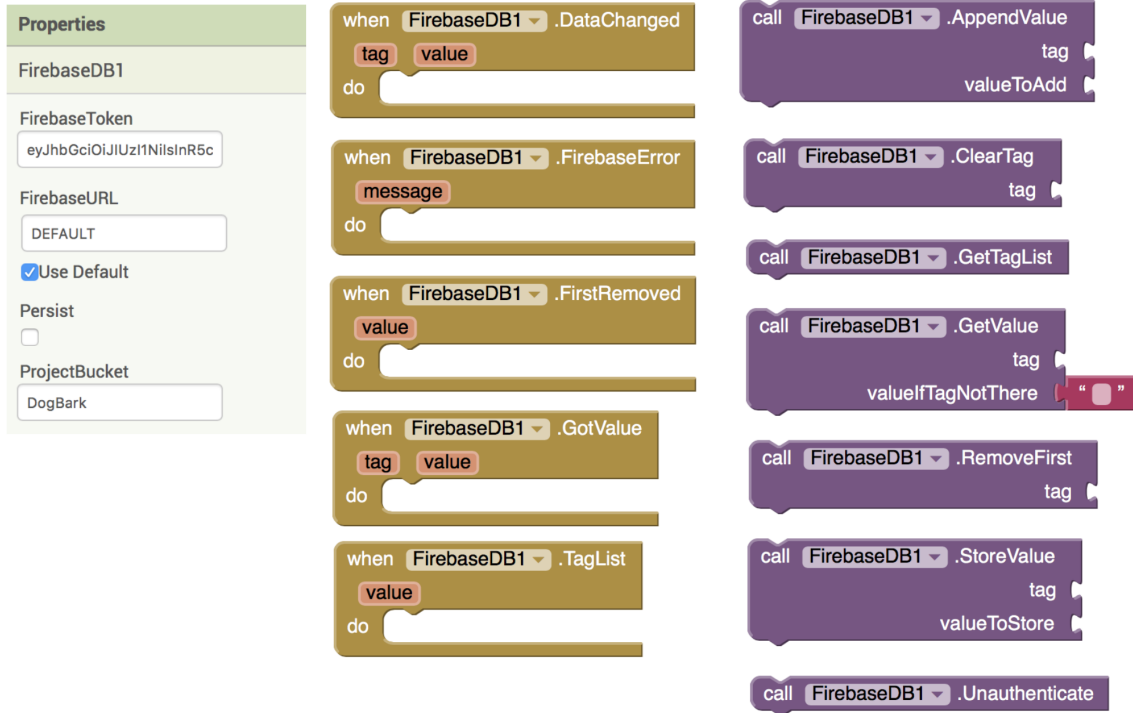


Figure 1-2: The FirebaseDB component currently available on MIT App Inventor has four properties set in the designer (left) and 12 blocks for programming.

App Inventor via QR code either through communication with the App Inventor Companion App or through direct installation of the packaged app onto an Android device. Neither option is compatible with the future of the Firebase database, so this method of providing cloud storage capabilities for MIT App Inventor is quickly reaching its end-of-life.

The FirebaseDB component has four properties, five event blocks, and seven method blocks (see Figure 1-2). Of its properties, the *FirestoreToken* and the *FirestoreURL* are usually preset by MIT App Inventor to direct to MIT App Inventor’s free Firebase account. The user may also change the *FirestoreURL* to point to their personal Firebase account. The *Persist* property deals with whether or not the developer wants variables to retain their values when the device is off-line or when the application is closed. If it is checked, values will be uploaded to Firebase the next time the app is run while connected to the network. The *ProjectBucket* property refers to the bucket that this project’s data resides in on the Firebase account.

Of the method blocks (which are colored purple), there are four that allow data modification in Firebase: *call FirebaseDB.StoreValue* stores the given value under the given tag; *call FirebaseDB.ClearTag* removes the tag and its stored value from Firebase; *call FirebaseDB.AppendValue* appends a value to the end of a list atomically; and *call FirebaseDB.RemoveFirst* returns the first element of a list to the *when FirebaseDB.FirstRemoved* event block (yellow) and atomically removes it.

The two *getter* method blocks ask for data from Firebase: *call FirebaseDB.GetTagList* asks for all of the tags corresponding to this project bucket, which triggers the *when FirebaseDB.TagList* event block; and *call FirebaseDB.GetValue* asks for the value stored under the given tag and returns it to the *when FirebaseDB.GotValue* event block. This method will return the input of the *valueIfTagNotThere* field if the tag does not exist.

The *when FirebaseDB.DataChanged* event block will return the tag and value of any data that has been changed or added to the project bucket. The *when FirebaseDB.FirebaseError* event block returns any error message that Firebase may send the application.

The CloudDB component I created to replace FirebaseDB is heavily modeled after its predecessor. It is important that current FirebaseDB users on MIT App Inventor are able to quickly and easily transition their FirebaseDB projects over to CloudDB and to continue creating projects that use shared data without much of a learning curve. A similar set of blocks will best help achieve this goal.

1.3 The Cloud is a powerful concept that allows people to have worldwide influence cheaply and quickly

Cloud computing refers to "the delivery of computing services—servers, storage, databases, networking, software, analytics, and more—over the Internet" [15]. Through this service, networks of servers in various remote locations can be managed to seamlessly conduct large scale computations or host vast amounts of data. Companies offering

Cloud services allow the user to quickly receive the results of their computational needs without having to worry about how these needs are fulfilled. It is referred to as "the Cloud" to highlight the idea that in this system, it doesn't matter where data is stored or on which machines computations are being carried out—only the abstraction matters.

Cloud computing is one of the most powerful computer science concepts from the last few decades regarding mass sharing of data and computational load. There has been a large boom in popularity for commercial Cloud services in the past decade, as seen by the exponential growth of the Amazon Web Services (AWS) and Google Cloud Services (GCS) businesses (see Figure 1-3). As such, an increase in educating students about the power of the Cloud should follow in schools. However, there is no move for such a suggestion in the current United States education system or even in the Computer Science Teachers Association (CSTA) standards. This may be in part due to the public's general impression of Cloud computing as a tool that is only used by the technological elite as opposed to something that can be made accessible to anyone.

MIT App Inventor is in a good position to show that anyone, even primary school students, can understand and use the Cloud to build something of relevance to themselves and their communities. Through App Inventor's implementation of a Cloud computing service with CloudDB, students will not need to do any highly technical tasks like learn to use AWS/GCS or set up and manage a server. They can truly view CloudDB as an abstraction. When FirebaseDB was created, there was no large push to create materials that would show young students how to use the component in a way appropriate for their age and interests. Additionally, the implementation of FirebaseDB included features that were useful for further customization of apps but could've been confusing for young children to use; for example, customizing the FirebaseToken and FirebaseURL to point to a personal Firebase account, and interfacing with Google's Firebase console. The design of the CloudDB component aims to make its usage as simple as possible with minimal extraneous information. Of course, even though we are using MIT App Inventor to teach the concepts around shared data

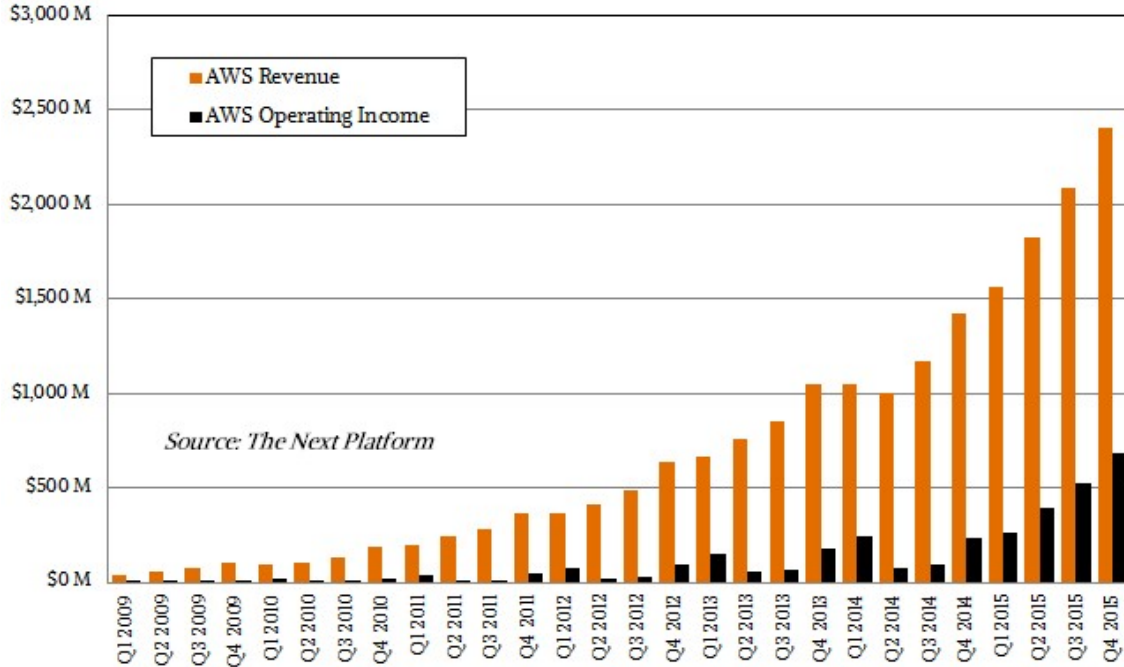


Figure 1-3: Like many other Cloud computing services, AWS sales has shown exponential growth for the past decade [16].

and the Cloud, knowledge about the Cloud should be transferrable beyond just the App Inventor tool. My study will also analyze how well students can understand the effect and usage of the Cloud in non-mobile scenarios.

1.4 Relevant Prior Work: Shariables and Cloud data-structures implemented Scratch enabled computational explorations of online data

There has been little prior work in the area of shared data education for primary school students with the exception of two data sharing projects for the Scratch programming language: Shariables and Cloud data-structures. Scratch is a visual, block-based programming language and environment for children that consists of graphical sprites on a stage that can be programmed using a set of instructional blocks [8]. As a blocks-based programming environment, Scratch’s coding interface is similar to that of MIT



Figure 1-4: Programmer interactions for using Cloud variables in Scratch 2.0 [7]. A variable can be set to be a Cloud variable by selecting the Cloud variable checkbox. A Cloud variable has the same methods/blocks as a normal variable.

App Inventor. However, Scratch projects are meant to be played and viewed on the Scratch website whereas MIT App Inventor projects are meant for mobile devices.

A normal Scratch variable can be a list, piece of text, or a number. It is only modifiable by the user of the Scratch project that it exists in and is only shared within that project. In 2007, Stern implemented Shariables for Scratch, which were variables stored on a server that could be modified across multiple projects and multiple users [9]. These Shariables had server-side persistence and had basic access control features so that the creator of the Shariable could choose whether the Shariable was open to anyone on Scratch, specific Scratch programmers, or multiple projects by the same programmer. Stern ran a 6-person study with students to teach Shariables and found that children were inspired to create personally meaningful projects, including networked games and personalized chat systems.

In 2013, Dasgupta implemented Cloud data-structures for Scratch 2.0, which extended scalar variables and lists in Scratch through a boolean property, thereby enabling Scratch programmers to store and retrieve data through their projects (see Figure 1-4) [7]. These Cloud data-structures were both persistent across multiple execution instances and shared between simultaneous instances. The key difference between Dasgupta’s work on Cloud data-structures and Stern’s work with Shariables was that Cloud data-structures are only accessible and writeable from the Scratch program that originally contained it.

A number of intermediate to advanced members of the Scratch community were

invited to participate in a user study for Cloud data-structures. Dasgupta presented four case studies of projects made by these Scratch programmers: (1) Two 13-year-old students used Cloud lists to keep a sorted collection of numerical scores, which was meant to persistently store high-scores in games; (2) A 13-year-old student remixed sample code from a Scratch 2.0 collaborative drawing project to improve the coordinate storage scheme and add new drawing features; (3) A 17-year-old member of the community created a survey that would plot users on a political map and show them everyone else who had taken the survey; (4) A 14-year-old student remixed sample code from a Scratch 2.0 chat room project to detect when a member of the chat was idle. From these case studies, Dasgupta found that children gained perspectives on larger issues such as privacy, safety, and server scale through working with Cloud data-structures in their personal Scratch projects.

1.5 The publish-subscribe messaging pattern allows an intermediary broker to send messages from publishers to invisible subscribers

Before I discuss the technical implementation of the CloudDB component, I present the communications model that makes it possible. The publish-subscribe (Pub/Sub) model is a common messaging pattern used in peer-to-peer applications. In Pub/Sub, the entity sending messages is called the publisher and the entities receiving the publisher's messages are referred to as the subscribers. The publisher sends its messages to a channel or some separate entity, which relays the messages along to relevant subscribers for the publisher. Through this indirect messaging pattern, the publishers and subscribers do not need to have any knowledge of each other. The intermediary node acts as a black box for communications [18].

The Pub/Sub model is implemented by Redis, the API used as a database, cache, and message broker in the CloudDB project. This allows messages from CloudDB to be sent on various channels. Redis's implementation is also extremely robust and

scalable so that massive numbers of subscribers may listen to the messages broadcasted by the publisher. This is an essential requirement for CloudDB, as one mobile application created with MIT App Inventor may be downloaded by large numbers of mobile devices, each serving as a separate subscriber. Subscribers may choose which channels to listen to and can also subscribe to messages based on glob-style pattern matching [19].

Chapter 2

Coding Methodology: CloudDB was implemented using the Redis API with simplicity as the design goal

The CloudDB component was designed and developed to allow MIT App Inventor developers to easily set up data sharing for their apps. Through CloudDB, data is created, modified, and shared between devices using two properties and fourteen blocks. The actual database was written using the Redis application programming interface (API). Redis was chosen because it is a large and well-maintained open-source project like MIT App Inventor with an expansive selection of features, including easy-to-use Pub/Sub capabilities [20]. It is also an in-memory data storage, which makes it exceptionally fast. Specifically, Jedis, a Redis Java driver, was used for compatibility with the MIT App Inventor codebase.

When a client, in this case a mobile device running a MIT App Inventor application that uses CloudDB, wishes to make a request to view or change data on CloudDB, it first communicates with the proxy server through the Transport Layer Security (TLS) protocol. Once the proxy server validates the request, it sends the request through the Transmission Control Protocol (TCP) to the Redis server, which is hosted locally on port 6379. For security purposes, the Redis database only accepts communications from localhost, which in this case is just the proxy server. The Redis server responds to

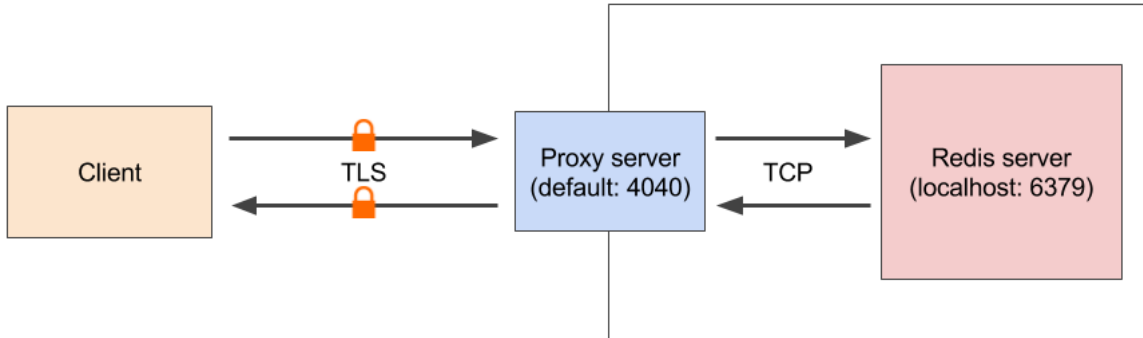


Figure 2-1: A client running a CloudDB MIT App Inventor application first communicates its request to the proxy server through the TLS protocol which communicates with the local Redis server to handle any requests.

the proxy server’s request through TCP. Finally, the proxy server sends the response back to the client through TLS (see Figure 2-1).

Client-side programming for the CloudDB component was done in Java and Lua script, Redis configuration directives were used to set up the server, and the Secure Sockets Layer (SSL) proxy server was written in Elixir/Erlang to encrypt CloudDB data.

2.1 Design Requirements: CloudDB must be easy to use and similar to FirebaseDB

I imposed one key requirement in designing the CloudDB component: Since one of the goals of the MIT App Inventor project is to democratize mobile application creation technology, CloudDB must be understandable and usable by young students and generally by anyone without extensive computer science training.

Additionally, since CloudDB was envisioned as a replacement for FirebaseDB, I also wanted users of the FirebaseDB component in MIT App Inventor to be able to easily transition to using CloudDB. Thus, I reused many of the same block names from FirebaseDB and followed the prior naming convention for any new blocks that I created.

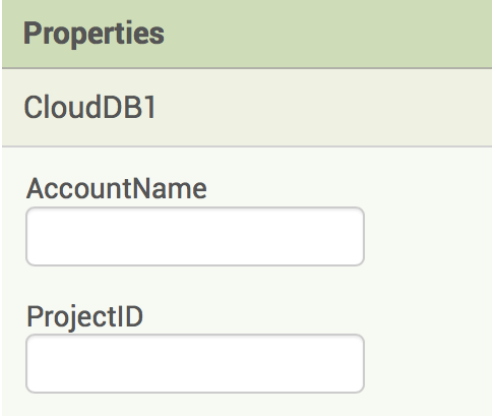
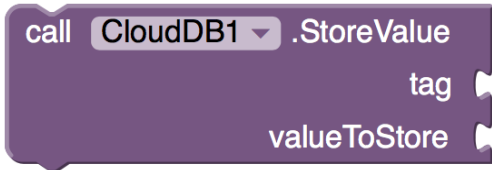
With these considerations in mind, I designed the CloudDB component to include

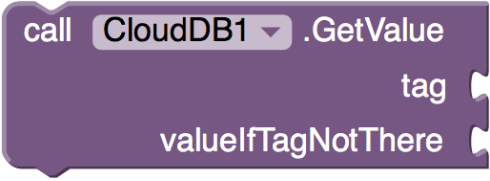
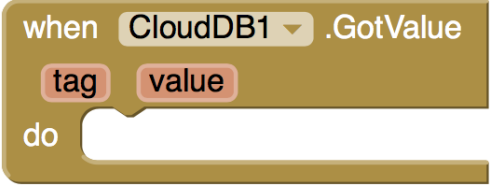
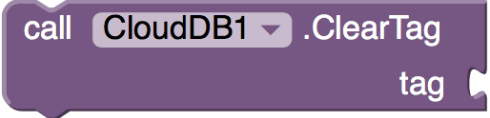
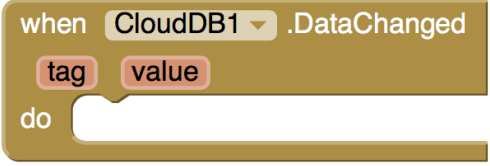
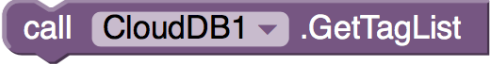
the following features: (1) Data buckets identified by just two text properties, *AccountName* and *ProjectID*, (2) A simple key-value pair data storage, modification, and retrieval scheme that works with a text-based key and a value of any type, (3) An event listener that returns the tag and new value any time data in the specific CloudDB project bucket is changed, (4) Atomic pop and append list operations, and (5) Under-the-hood SSL encryption.

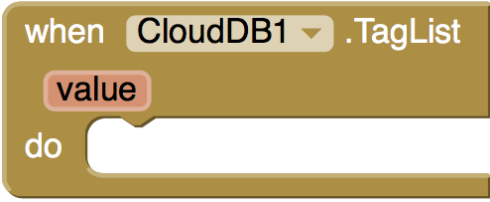
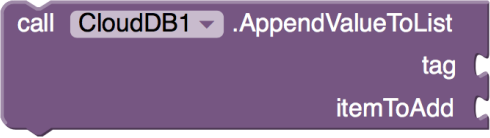
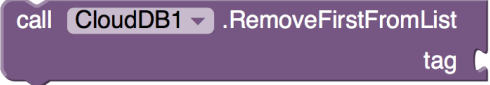
2.2 CloudDB has two properties and fourteen blocks

The CloudDB component has two properties in the MIT App Inventor Designer and fourteen blocks in the Blocks editor. An overview of the functions of each is given below (see Table 2.1).

	Block Image	Function
--	-------------	----------

a.	 <p>The screenshot shows the 'Properties' panel for a 'CloudDB1' component. Under the 'CloudDB1' section, there are two input fields: 'AccountName' and 'ProjectID', each with a text box for user input.</p>	<p>The <i>AccountName</i> and <i>ProjectID</i> are the only two properties of the CloudDB component. They are both Strings and allow the project to access the correct set of data in the CloudDB server. The <i>AccountName</i> is autofilled to be the email account associated with the MIT App Inventor developer account. The <i>ProjectID</i> is autofilled to be the name of the current project. Both fields can be changed by the developer to any String. For example, if the developer wishes to access the CloudDB data set from another user's project, the <i>AccountName</i> would be changed to the other developer's email address and the <i>ProjectID</i> would be the name of their project.</p>
b.	 <p>The screenshot shows a purple code block with the text: 'call CloudDB1 .StoreValue tag valueToStore'. The 'CloudDB1' is in a dropdown menu, and 'tag' and 'valueToStore' are input fields.</p>	<p>Data is initialized and stored in CloudDB through the <i>StoreValue</i> block as a key-value pair. The key is labeled as <i>tag</i> and should be a String. The value is labeled as <i>valueToStore</i> and should be the data being shared. The <i>valueToStore</i> can be any datatype currently supported by MIT App Inventor, including text, lists, images, or sound files.</p>

c.	 <pre>call CloudDB1 .GetValue tag valueIfTagNotThere</pre>	<p>The <i>GetValue</i> block sends the request to retrieve a specific value from CloudDB. There are two inputs: (1) the <i>tag</i> of the desired data and (2) <i>valueIfTagNotThere</i>, the value that should be returned if the desired tag does not exist in CloudDB.</p>
d.	 <pre>when CloudDB1 .GotValue tag value do</pre>	<p>The <i>GotValue</i> block is the event block that pairs with the <i>GetValue</i> block. It is triggered when the server returns the desired tag and the corresponding value when CloudDB completes the <i>GetValue</i> request.</p>
e.	 <pre>call CloudDB1 .ClearTag tag</pre>	<p>The <i>ClearTag</i> block removes the key associated with the input <i>tag</i> from the database, thereby deleting both the tag and its value.</p>
f.	 <pre>when CloudDB1 .DataChanged tag value do</pre>	<p>The <i>DataChanged</i> block is an event block that is triggered whenever anything is changed in the CloudDB project. It returns the tag and updated value of any data that was stored or cleared.</p>
g.	 <pre>call CloudDB1 .GetTagList</pre>	<p>The <i>GetTagList</i> block sends the request to retrieve a list of all of the tags in the CloudDB project from the server.</p>

h.	 <p>A Scratch 'when' block with a dropdown menu set to 'CloudDB1' and the text '.TagList'. Below it is a 'value' block, and then a 'do' block with a large empty space for code.</p>	<p>The <i>TagList</i> block is the event block that pairs with the <i>GetTagList</i> block. It is triggered when the server returns the list of all tags in the project bucket once CloudDB completes the <i>GetTagList</i> request.</p>
i.	 <p>A Scratch 'call' block with a dropdown menu set to 'CloudDB1' and the text '.AppendValueToList'. It has two input fields: 'tag' and 'itemToAdd'.</p>	<p>The <i>AppendValueToList</i> block allows items to be atomically appended to a list. Thus, when multiple devices attempt to add items to a CloudDB list at the same time, no items will be lost or overwritten. The inputs are the <i>tag</i> associated with the list and the <i>itemToAdd</i>. If the tag does not exist or if the tag does not have a list as its value, an error will be returned.</p>
j.	 <p>A Scratch 'call' block with a dropdown menu set to 'CloudDB1' and the text '.RemoveFirstFromList'. It has one input field: 'tag'.</p>	<p>The <i>RemoveFirstFromList</i> block sends the request to atomically remove the first element from a list. Thus, when multiple devices attempt to add to or remove items from a CloudDB list at the same time, no items will be accidentally lost or overwritten. The input is the <i>tag</i> of the list. If the tag does not exist or if the tag does not have a list as its value, an error will be returned.</p>

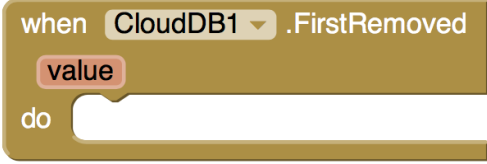
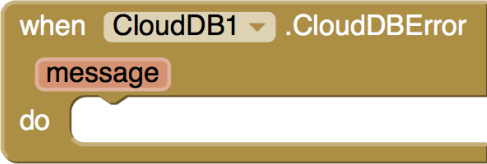
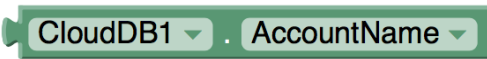

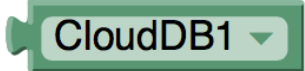
k.		<p>The <i>FirstRemoved</i> block is the event block that pairs with the <i>Remove-FirstFromList</i> block. It is triggered when the server successfully removes the first element from a valid list. This block returns the value that was removed when CloudDB completes the request.</p>
l.		<p>The <i>CloudDBError</i> block is an event block that is triggered whenever CloudDB throws an error. It returns the error message as a String.</p>
m.		<p>The <i>AccountName</i> block is a getter block for the (String) <i>AccountName</i> property of the CloudDB project.</p>
n.		<p>The <i>ProjectID</i> block is a getter block for the (String) <i>ProjectID</i> property of the CloudDB project.</p>
o.		<p>The <i>CloudDB</i> block is a getter block for the specific CloudDB instance.</p>

Table 2.1: An overview of CloudDB’s two properties and fourteen blocks

2.3 Simple key-value formatting in the Redis database allows for flexibility in data sharing

Unlike in the FirebaseDB component, where the default data bucket is tied to the specific application, the *AccountName* and *ProjectID* fields are the only identifiers of the data bucket being accessed in the Redis server by a CloudDB instance. Every piece

of data coming from the CloudDB component in any MIT App Inventor project is stored in the same Redis database. For a given tag, its corresponding key in Redis is a String joining together the given *AccountName*, *ProjectID*, and *tag*. The *valueToStore* is converted into its JSON representation and stored as a JSON-serialized String in Redis. Redis's lightweight Pub/Sub and pattern matching subscription capabilities allow a CloudDB instance to easily subscribe to relevant messages from the Redis server.

It is also possible for multiple MIT App Inventor applications to use the same data bucket, resulting in many possibilities for interesting and complex shared data projects. For example, multiple people can create different apps that visualize the same data set in different ways, an app can be created that summarizes several data buckets, and apps that give specific users limited access to select parts of a large data project can be customized. Additionally, this easy-to-parse design allows for future work in creating a console for viewing data and logging.

One detriment to this approach is that all of the data can be easily viewed and edited by anyone using CloudDB through MIT App Inventor. While FirebaseDB requires an authentication token so that each project is hardcoded to only be able to view the one data bucket tied to the current project (with the exception of creating a personal Firebase account, which is generally only used by advanced developers), CloudDB allows a developer to view and edit any CloudDB project created by any MIT App Inventor developer without permission. However, this feature also allows users to create multiple CloudDB instances in a single project that access multiple different project buckets simultaneously. We believe that the powerful and flexible data sharing capabilities that CloudDB currently provides will be extremely valuable to users. Privacy and security warnings will be given to users of CloudDB prior to use.

Additionally, the returned variables associated with this key-value implementation are potentially confusing. Suppose I call *CloudDB.StoreValue* with *tag* = *mostPopularMajors* and *valueToStore* = ("6", "2", "18"). Later on, I wish to modify the list by adding an element, "2A", without using the call *CloudDB.AppendValueToList* block.

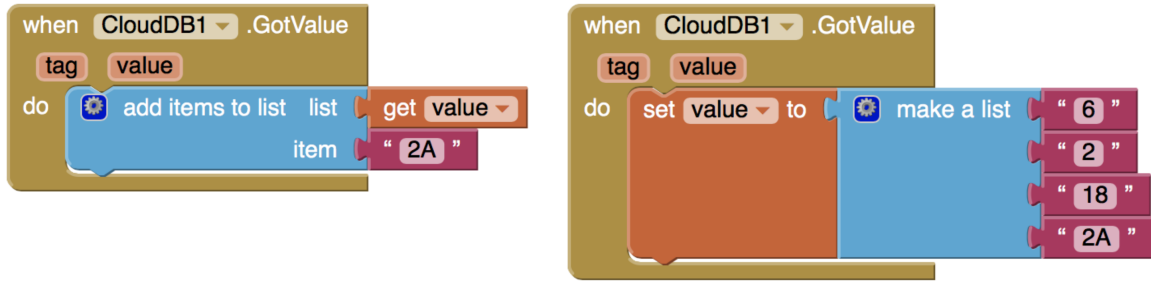


Figure 2-2: Although both of these blocks operate on the *value* returned by *when CloudDB.GotValue*, all edits are only local. Neither actually modify the list stored on the CloudDB server.

I first *call* *CloudDB.GetValue* with requested *tag* = *mostPopularMajors* and receive a response to the *CloudDB.GetValue* request through the *when CloudDB.GotValue* block. I then attempt to change the value in the *when CloudDB.GotValue* block by either directly setting the *value* that was returned or modifying it within the block (see Figure 2-2). However, the *value* returned by *when CloudDB.GotValue* is a copy of the actual value and not a pointer to the value itself. Thus, neither method will actually modify the list stored under the specified tag inside the CloudDB server—only the local *value* variable will be modified. Similarly, the *tag* returned by *when CloudDB.GotValue* is also just a copy of the actual tag. While this behavior is consistent with the rest of MIT App Inventor, it might cause some confusion in novice programmers.

2.4 Atomic operations were written in Lua script

It was fairly easy to set up basic CloudDB functionality, including key-value store and data changed event notifications, using Jedis’s built-in Pub/Sub features. However, atomic operations could not be used out-of-the box. The Redis database provides the **LPOP** *key* and **LPUSH** *key value [value ...]* commands, which atomically pop and push an item from or to a list stored in Redis [21]. However, in order to use these commands through the Jedis Java client, the list inputs needed to be standard Java lists. As explained previously, MIT App Inventor stores all such data as JSON Strings, which makes it incompatible with the Redis atomic list protocols.

Atomic operations such as list pop and push are fairly important for CloudDB functionality. Without such features, concurrency issues such as the readers-writers problem may arise when multiple clients attempt to change the same value [23]. Additionally, because FirebaseDB currently supports atomic list methods, we needed to support them in CloudDB as well.

In order to define atomic list operations in the context of MIT App Inventor, Lua scripts were written and evaluated using Redis's built-in Lua interpreter, **EVAL** [22]. Redis guarantees that Lua scripts are executed atomically. No other script or Redis command will be executed while a Lua script is being executed. Due to these properties, in order for atomic operations to not bog down the entire system, the scripts were written in a way that made them quickly executable. Two short Lua scripts were written, one for the atomic *AppendValueToList* block and the other for the atomic *RemoveFirstFromList* block (see Appendix A).

2.5 Semantic change to list: Lists stored in CloudDB are passed by value instead of by reference

Lists are the only MIT App Inventor datatypes with mutators, and as such, list objects are passed by reference [28]. Take a global variable in MIT App Inventor named list1 with the value ("A", "B", "C"). If another global variable named list2 is initialized and then set to list1, a reference to list1 is passed to list2. When a change is made to list2, the value of list1 will also reflect that change.

However, this rule does not apply to lists that are stored in CloudDB. When *CloudDB.StoreValue* is called, if the *valueToStore* is a list, then a copy of the list is stored instead of a reference to the original list. Taking the above example as an analogy in CloudDB, suppose *CloudDB.StoreValue* is called to store *tag* = list1 and *valueToStore* = ("A", "B", "C"). If *CloudDB.GetValue* is called to retrieve list1, and the value of list1 is stored as a global variable named list2, any changes to list2 will not be reflected in list1. When list2 was set to the value of list1, CloudDB passed it

a copy of the value instead of a reference to the original list. This semantic change to the list object in MIT App Inventor is due to basic limitations of databases, which also occurred in FirebaseDB.

2.6 Data is encrypted through a SSL proxy server that communicates with a local Redis server through TCP

Developers and users of CloudDB applications may upload and share personally identifying or potentially sensitive data. Thus, this component needs to both support transport level encryption to keep data transfer private and authenticate the Redis server instance to keep data secure. Redis inherently has a very weak security policy: it offers no transport level security and only provides server administrators with the option of using a password to authenticate clients communicating with the Redis server. None of the traffic sent to and from Redis is encrypted, so all of the data, including the password, can be easily sniffed as it travels over unsecured TCP connections. In order to increase data security and protect the Redis server from potential attacks, an SSL proxy server sitting on the same machine as CloudDB's Redis instance was built. The proxy accepts requests from the internet via SSL, and then forwards them via TCP over localhost to Redis. Redis was configured to only accept requests from localhost, protecting it from potential DDoS or other attacks. The proxy server was implemented using the Elixir package, which runs on the Erlang VM.

Erlang-based applications such as the proxy server are composed of processes, each with their own stack and heap, and which communicate via message passing. Hundreds of thousands of these lightweight Erlang processes can be spawned on a single server, which makes this system highly scalable and robust [26]. There are two types of processors: workers, which perform the computations that the application needs, and supervisors, which monitor and can restart worker processes [25]. For the

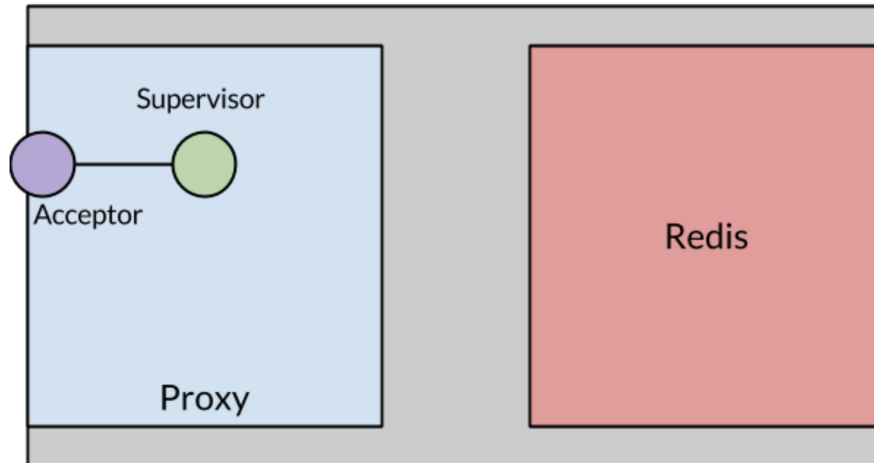


Figure 2-3: The proxy server at startup has one supervisor process and one outward-facing acceptor worker process.

rest of Chapter 2.5, I will present the hierarchical arrangement of processes used in this application in the form of a supervision tree.

When it is first spun up, the proxy server has only one supervisor process and one worker process. Specifically, the worker process is a SSL listener process that starts listening on the specified outward-facing port. It runs an infinite loop and waits for clients to attempt to connect to the server at its specific port. When a client first sends a request to CloudDB, it connects to this listener/acceptor process via the TLS protocol (see Figure 2-3).

When the client connects and the SSL handshake is performed, the acceptor establishes a socket and a new TCP connection to Redis for this client (see Figure 2-4).

Two workers are then spawned for the client request and added to the supervisor's list of children. One worker is a reader which reads from the client's TLS socket, and the other is a reader which reads from the new Redis socket (see Figure 2-5). The client-facing process waits for the client to send new requests to forward to Redis and the Redis-facing process waits for Redis to send a response to forward to the client. When Redis gives a response, the Redis handler reads a line of TCP from its read socket and writes that line to its write socket. The Redis handler then relays the response back to the client through a secure TLS connection. If the client has a

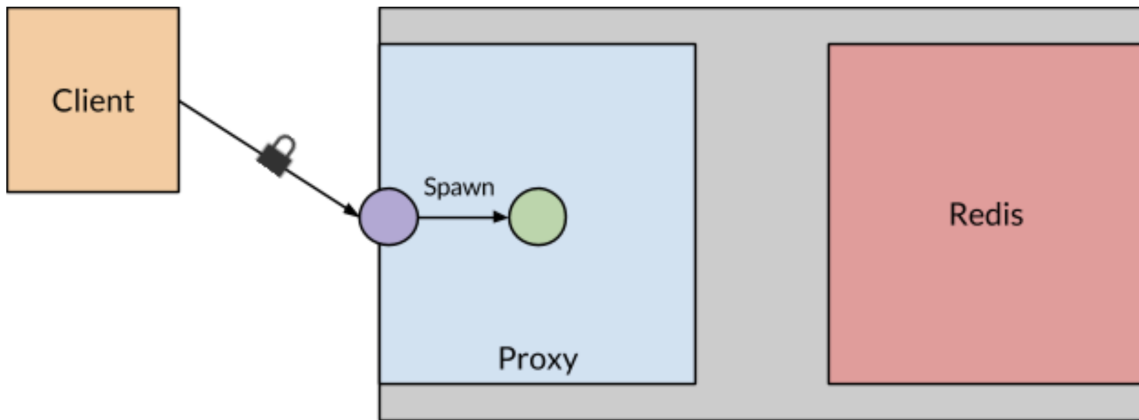


Figure 2-4: A client connects to the proxy service via TLS. The SSL handshake is performed and the acceptor process instructs the supervisor process to spawn a client handler process as well as a Redis handler process.

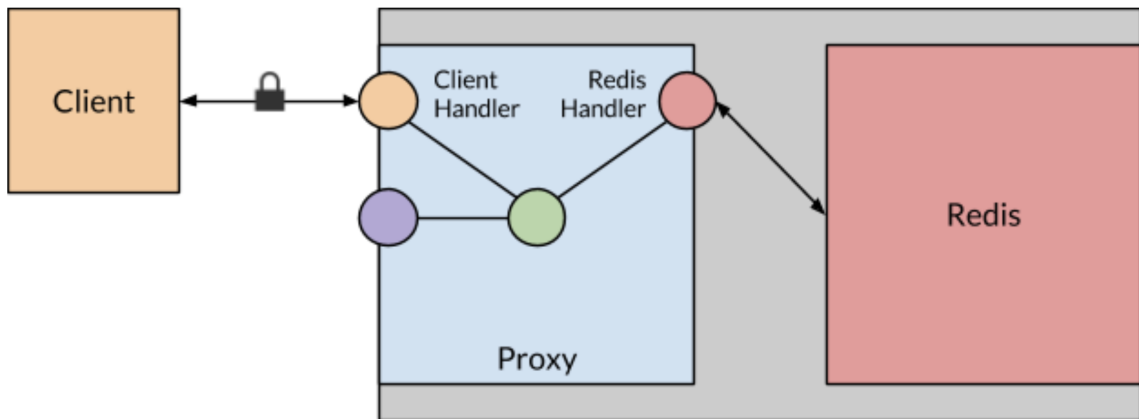


Figure 2-5: The client now sends information to the client handler process, which forwards it to Redis. The Redis handler listens for responses from Redis and forwards them back to the client. The acceptor process waits for other clients to attempt to make new connections to the proxy.

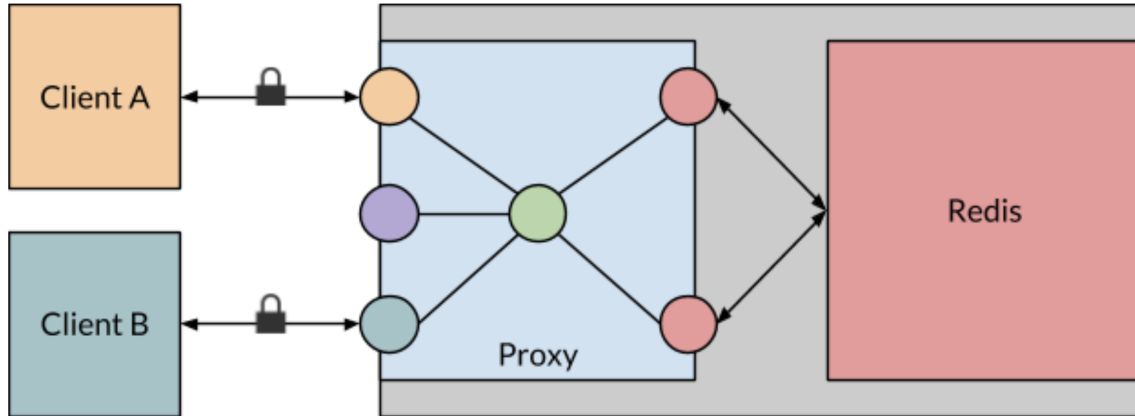


Figure 2-6: Multiple clients connecting simultaneously to the proxy service, each with their own individual client handler process and Redis handler process.

new request, the client handler reads a line of TLS from its read socket and writes it to its write socket to pass to Redis through TCP. When a client is finished and the connection closes, Elixir automatically sets its associated client and Redis handlers to die.

The first worker process (the listener/acceptor) continuously listens for new clients attempting to connect to Redis through its open TLS socket and repeats the processor spawning process for each new client (see Figure 2-6). This parallelism combined with Redis’s built-in concurrency allows for multiple devices to seamlessly communicate with CloudDB at once without collision. Every response from Redis is directly passed to the appropriate client handler and then the correct client. Additionally, the proxy server employs the one-for-one supervision strategy so that the supervisor automatically restarts any proxies that unexpectedly crash.

2.7 MusicShare: Example of a multi-functional app built using CloudDB

CloudDB is able to interface with various MIT App Inventor components to create complex apps that allow data sharing. One example of an app that used to be impossible to create in MIT App Inventor without private extensions or significant

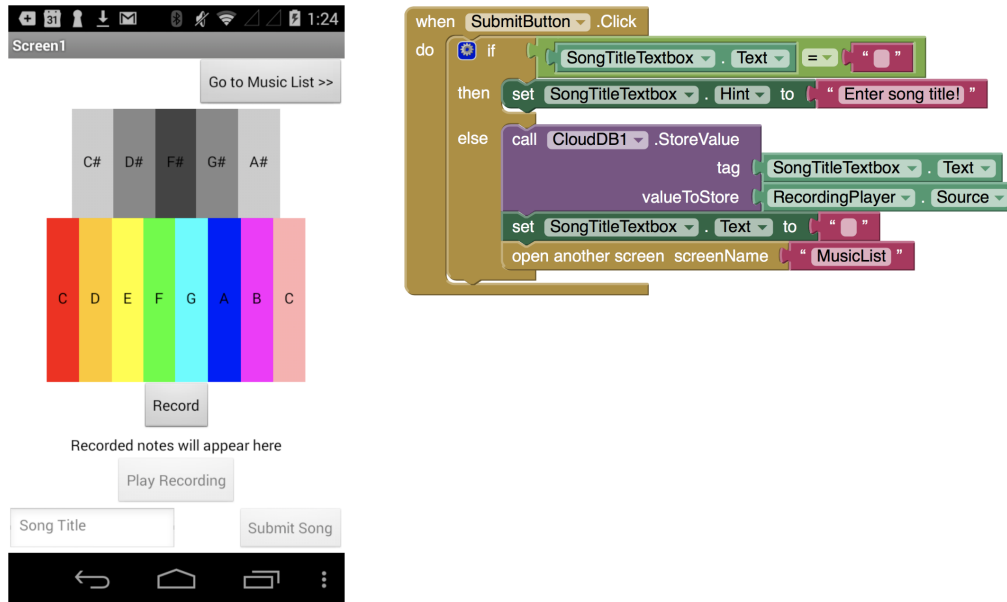
external programming is the MusicShare app. This app extends a basic piano app and allows users to play, record, and submit their own songs, as well as listen to songs that others users of the app have shared (see Figure 2-7).

The first screen of the MusicShare app has a standard piano and music recording interface. When a user wishes to submit a recording they have made, they enter a name for their song and click the "Submit Song" button. This triggers the *when SubmitButton.Click* block, which calls *CloudDB.Store Value* to store the song name as the *tag* and the source recording as the *value* (see Figure 2-7a). CloudDB is able to recognize media files such as sound, image, or video files. It stores these files as a byte array and encodes that array as a base64-encoded string. This built-in functionality was mostly re-used from the bincompfile extension written by Jeff Schiller.

If a user wishes to view and listen to all of the songs that have been submitted to the app, they can click the "Go to Music List" button from the first screen, which brings them to another screen displaying a ListView of all of the song titles that have been submitted to the project. The list is first populated using *call CloudDB.GetTagList* when the screen is initialized. It is then updated every time a song has been added or modified through the *when CloudDB.DataChanged* block. When a list element is picked, *CloudDB.GetValue* is called to retrieve the selected tag. When the get request has been completed through the *when CloudDB.GotValue* block, the media file's byte array is automatically decoded and a MIT App Inventor Player can directly play the *value* that was retrieved.

CloudDB infers that *RecordingPlayer.Source* is a media file.

A programmer would not have been able to build this app using FirebaseDB because media files, such as sound files, cannot be natively stored through the FirebaseDB component.



(a) MusicShare’s first screen, which provides a user interface for creating and submitting music. The only block in this screen that interfaces with CloudDB is *when SubmitButton.Clicked*.



(b) MusicShare’s second screen, which lists all of the songs that have been submitted to the repository. The list is automatically updated live. Tapping on a song automatically plays it.

Figure 2-7: Screenshots and CloudDB code from MusicShare, an example CloudDB app that lets users create, share, and listen to music.

Chapter 3

Teaching Methodology: Curriculum was created for a 6-lesson workshop for middle school students

In order to test the usability of CloudDB for the target audience and find out if middle school aged students can understand and create meaningful projects with shared data technology, I designed and ran a workshop for middle school and junior high school students. Ten students between the ages of 12 to 15 years old participated in the workshop. They were in grade seven, eight, or nine. The workshop was hosted every Saturday from 12:05 to 12:55 and lasted from February 18, 2017 to April 1, 2017, with the exception of March 11, 2017 due to a holiday. The total intervention time was around 5 hours. Throughout the lessons, students were introduced to the MIT App Inventor tool, were explicitly taught computational thinking concepts regarding the Cloud, learned how to use the CloudDB component in MIT App Inventor, and created independent final projects.

3.1 Participants in the workshop had little to no experience with programming

The workshop was given in the form of a class through HSSP, a semesterly program ran by the Massachusetts Institute of Technology Educational Studies Program for community service and outreach in the Boston and Cambridge area [17]. All of the lesson plans for this workshop were reviewed and approved by professional teachers from HSSP. Students pay \$40 to join the program and can register to lottery for up to four classes. According to the program guidelines, students are required to select classes without any outside or parental interference, which encourages them to pursue their personal interests. From basic information inferable about the general population of HSSP participants, it is likely that the subjects of my study are more scientifically-minded and more interested in technology than the average 12 to 15 year old student.

In the description for the workshop, I emphasized that students should sign up if they are interested in an opportunity to try novel technology and learn how to build mobile applications. I indicated that no prior experience with programming, MIT App Inventor, or mobile development was needed. Over 100 students lotteried for my class as their first choice, which was very high compared to other classes in the program, but due to constraints on room size and teaching staff, only ten students could be selected through the random lottery process. However, it is clear from the response rate that this topic is both relevant and highly interesting to the target audience.

Of the students selected for the class, six were male and four were female. Two students indicated that they had prior experience in blocks-based programming through Scratch and one other student indicated very minimal exposure to a text-based programming language. The rest of the students indicated that they had no prior programming experience. None of the students had previously built an application for mobile, web, or any other interface. As a part of the MIT Committee on the Use of Humans as Experimental Subjects procedures, the students and parents/guardians

had to sign minor assent and parental consent forms to allow their data to be collected for research and to be filmed (see Appendix B). Students were still encouraged to take the class if they did not agree to release their data for research, but all ten students and their parents/guardians signed the forms.

3.2 A lesson plan was created summarizing the schedule, concepts taught, and data collected

The six hour-long lessons were planned for 50-minute time increments to provide a buffer for late arrivals or unexpected delays. The goal of the first lesson was to teach the students how to use the MIT App Inventor tool and the lab tablets to create some basic apps (see Table 3.1).

Time	Activity
5 min	Class introductions: Name, grade, why are you taking this class, any programming experience you have (Scratch, Java...).
5 min	<p>What is App Inventor?</p> <ul style="list-style-type: none"> • A website that lets you design and make apps (show site). • Have the students go to the App Inventor website. • Have the students open a new tab and navigate to the development page. <p>Make sure everyone has downloaded the App Inventor Companion app on their mobile devices. Let students create accounts on the site and figure out how to use it at their own pace.</p>
40 min	Students go through a list of beginner tutorials. Encourage them to talk to other students in the class if they have questions. When they are done with the first app, teach them individually how to download the app through the QR code and APK method.

Table 3.1: Lesson 1 Plan, Introducing MIT App Inventor

The goal of the second lesson was to explain the concept of the Cloud to students, conduct a preliminary survey, and introduce the CloudDB component (see Table 3.2). Students were also presented with the coding challenge of modifying an App Inventor drawing app to allow multiple devices to draw on the same picture at the same time using CloudDB.

Time	Activity
5 min	<p>Introduce the concept of the Cloud:</p> <ul style="list-style-type: none">• Ask students what they do on their computers and phones.• If they mention anything (email, games with shared high scores), ask them how they think it works. <p>Explain the Cloud:</p> <ul style="list-style-type: none">• Tell them that the Cloud is basically a bunch of computers all over the world that talk to each other through the Internet. This lets them do things together that they couldn't do alone.• Ask students if they can think of any examples of the Cloud.

5 min	<p>Give the Google Docs example:</p> <ul style="list-style-type: none"> • Problem: You can't access your Word document on another computer so you use Google Docs. • In Google Docs, your file is not saved on your own computer like a Word file would be. • A Google Doc is saved on a computer somewhere else called the "Cloud," which is pretty much another computer somewhere far away that you can talk to through the internet. • This allows you to open up the same exact doc on any other computer by constantly talking to the "Cloud" computer. In fact, multiple people can access your file and edit at once. • If you and your friend are both on the same doc, the Cloud computer will constantly talk to both computers at the same time. • The computers will tell the Cloud computer what changes they want made on the doc, and the Cloud computer will update any connected computers.
5 min	Students fill out the Pre-questionnaire (see Appendix C.1).
15 min	Introduce the CloudDB component in App Inventor with a handout (see Appendix C.2). Explain that CloudDB allows you to store data onto a special Cloud computer for your apps.

20 min	<p>Present the Draw Together coding challenge:</p> <ul style="list-style-type: none"> • Give them the partially complete AIA for the app. It is without the CloudDB components and currently just draws and wipes by itself. • Pair up the students and have them do the Draw Together Worksheet (see Appendix C.3). • They should make a plan for how the drawing data is shared between devices.
--------	---

Table 3.2: Lesson 2 Plan, Introducing the Cloud

In the third lesson, the class went over the solutions to the Draw Together CloudDB coding challenge and was encouraged to expand on the basic app or try to make another CloudDB app (see Table 3.3).

Time	Activity
10 min	Give students the Draw Together Solutions and Extensions printout (see Appendix C.4). Ask them to try and understand the solution and use it to get their code working. If they already finished the app, ask them to look at the extension that let's the user paint in different colors.
5 min	Ask a pair with the best working solution to explain how it works to the rest of the class.

35 min	<p>Pose coding challenges around the Draw Together Extension and other apps. Students are free to work on them for the rest of the class with staff guidance:</p> <ul style="list-style-type: none"> • How would you implement a Draw Together app with multiple colors? • How would you build a Music Sharing app (given a starter AIA file)? • How would you make a game with a global high score (given a starter AIA file)?
--------	--

Table 3.3: Lesson 3 Plan, CloudDB Draw Together Wrapup

In the fourth lesson, the independent final design project was introduced. Students brainstormed both individually and in teams to design their final apps and had a bit of time to begin pair programming (see Table 3.4).

Time	Activity
5 min	<p>Introduce the design project.</p> <ul style="list-style-type: none"> • This project is an opportunity to create a mobile app of your own design and imagination. • Students will have two classes to make their projects with a partner and will do a short presentation during the final class. • There are two constraints: (1) The project's theme will be the ENVIRONMENT, so the app should have some relation to that theme, and (2) Everyone's projects must use the CloudDB component.

10 min	Students complete the Individual Design Worksheet (see Appendix C.5).
15 min	<p>Have students talk to at least 3 other people in the class about their ideas. At the end, have students find a partner to do their project with based on mutual interests (or group of 3 if class has an odd number). Once students finish pairing up, have each group do the Group Project Design Worksheet (see Appendix C.6).</p> <ul style="list-style-type: none"> • Check with every team to make sure they have a reasonable project plan. • Make sure the projects can be completed in a reasonable time.
5 min	Pair up teams as they finish to do the Design Peer Feedback Worksheet (see Appendix C.7). Teams should take turns talking about their projects and take notes on what their peers say.
15 min	Students begin to pair program their app, starting by trying to implement the CloudDB portion first. Collect their AIA files at the end of the class.

Table 3.4: Lesson 4 Plan, CloudDB Independent Design Project Workday 1

Students were given the entire fifth class to work on their final projects (see Table 3.5).

Time	Activity
45 min	Students finish pair programming their projects. Remind students to switch roles every 8 minutes or so if needed. Collect their AIA files.
5 min	Explain the final project presentations next week. Each team will need to make a 3-minute Google Slides presentation about their project with a 1-minute question and answer session after.

Table 3.5: Lesson 5 Plan, CloudDB Independent Design Project Workday 2

The goal of the sixth and final lesson was to wrap up the final projects and have students give presentations (see Table 3.6).

Time	Activity
30 min	Allow students to make and practice their presentations, as well as make any final touches to their apps. Collect AIA files.
15 min	Student groups give their final project presentations.
5 min	Post-questionnaire (see Appendix C.8).

Table 3.6: Lesson 6 Plan, CloudDB Independent Design Project Final Presentations

3.3 Data Collected

Three types of data were collected during the workshop: written, audiovisual, and MIT App Inventor project files. The written data collected was in the form of worksheets given in class. The worksheets included students' answers to concept questions about CloudDB to ensure understanding, plans for how they could potentially use CloudDB in their apps, their app design process, and peer feedback based on interview questions. All of the worksheets given to the students directly assisted in their learning and completion of in-class activities except the Pre-questionnaires and the Post-questionnaires, which were given solely for research purposes. Students were told that the two questionnaires were entirely anonymous and did not reflect their work or abilities in any way.

The Pre-questionnaire was given during Lesson 2 of the workshop (see Appendix B.1). There were four 5-point Likert scale questions gauging how important students perceived the Cloud to be and how comfortable they were with the idea of it. There were four additional free response questions that tried to probe at the details of students' understanding of the Cloud's role in society and their relation to it. This

questionnaire was given after the concept of the Cloud was briefly explained to students so that it would not be unfairly biased against students who had not heard of the term before but conceptually understood the ideas surrounding shared data.

The Post-questionnaire was given at the very end of the workshop as the final activity of Lesson 6 (see Appendix B.8). The four Likert scale questions from the Pre-questionnaire were repeated to gauge how students' perception and understanding changed throughout the course. One more Likert question was added to the Post-questionnaire specifically probing at how students personally felt about the efficacy of the workshops in improving their understanding of shared data. The four free response questions from the Pre-questionnaire were also repeated. Additionally, there were three new free response questions that specifically focused on students' experiences while working on their final design projects.

All of the students had turned in their assent and consent forms by the third lesson, so we began to film one pair of students using a videocamera. This pair of students sat in the corner of the room. They were chosen due to the availability of a power source nearby and the tripod being mostly out of the way in the corner compared to elsewhere in the limited class space. The camera was pointing from behind the students so that it could capture the state of their computer monitors. At the end of the final lesson, when student groups presented their final design projects, the videocamera was shifted to point at each of the presenting students.

The MIT App Inventor project files (AIAs) were also collected at the end of the class for the students' final projects. This file can be opened on the developer website to view both the Designer and Blocks Editor portions of a specific app. Additionally, if the teaching staff heard any interesting quotes from the students during class, they would note them down.

Chapter 4

Results: Data from the workshop shows that young students can quickly understand shared data

The workshop involved ten students between the ages of 12 and 15. The measurements taken during the workshop can be parsed to answer two separate research questions: (1) "To what extent can middle school students understand the Cloud?" and (2) "Can young people build meaningful technologies using the Cloud that could affect their communities?"

It is important to note that the number of participants in the study was insufficient to make definitive conclusions about the CloudDB system or the curriculum. Thus, we present these results as an exploration to inform future experiments.

In order to help us answer the first question in the context of MIT App Inventor and CloudDB, I analyzed students' Pre-questionnaire responses, students' Post-questionnaire responses, and the adeptness of their technical usage of the CloudDB component in their final design projects.

Question two was answered by analyzing students' Post-questionnaires, individual and group project design worksheets, video of students' final project presentations, and the level of completion and quality of these final projects.

Relevant quotes were collected during in-class discussions and students' free work

time that supported both research questions. Students' Pre-questionnaire and Post-questionnaire responses were also compared to gather how effective the workshop intervention was at teaching shared data concepts.

4.1 Students quickly understood high level ideas and strong role of shared data in modern society

The concept of shared data was first introduced in Lesson 2 through a brief explanation. Only one student said that they had heard of "the Cloud" before the lesson, but that student could not provide an explanation and did not know what it was. The specific example of Google Docs was used as a case study to relate the concept to a popular modern technology that is commonly used. The students seemed to catch on quickly and did not have any clarifying questions after the concept was introduced. One student's comment hinted at young students' ability to quickly gain a wider perspective on how data sharing works in the Internet ecosystem:

"Isn't this just the Internet? ...Internet servers store data and share and save it."

— **Student A**

Immediately following the explanation of shared data, students were given Pre-questionnaires to measure their understanding and attitudes regarding the Cloud. There were four 5-point Likert scale questions and four free response questions. When analyzing their responses to the four Likert scale questions, I coded **Strongly Agree** as 5 and **Strongly Disagree** as 1. The mean score and range of scores are displayed in Table 4.1 below:

#	Question	Mean	Range
1.	I use the Cloud in my daily life.	4.5	4-5
2.	I understand how the Cloud stores and shares data.	4.125	3-5
3.	I feel comfortable making apps that use the Cloud.	3.875	2-5

4.	I can think of ways that the Cloud can solve problems in my everyday life.	4.25	3-5
----	--	------	-----

Table 4.1: Mean and range of responses for Pre-questionnaire Likert scale questions on a 1-5 scale.

Students rated question 1 very high, all agreeing or strongly agreeing, which indicates that they felt that data sharing technology was highly relevant in their daily lives. This was perhaps influenced by the previous in-class discussion regarding how Cloud data sharing is used in popular modern technologies such as Google Docs. The mean score on question 2 was slightly higher than **Agree**, indicating that students generally felt like they understood how the Cloud works. Question 3 had the lowest score and largest range. On average, students were **Neutral** to **Agree** on feeling comfortable making apps that use the Cloud. A lower score is to be expected given that none of the students had previously created an app with shared data technology. The larger variation in the responses to question 3 may have also reflected students' differing comfort levels with using the MIT App Inventor tool after the previous lesson. Perhaps some students found MIT App Inventor to be easy to use and inferred that Cloud data sharing in MIT App Inventor would not be too difficult. Students gave question 4 high scores, saying that they generally agreed with being able to think of ways that the Cloud could solve problems in their daily lives. This is later reflected in students' abilities in thinking up a wide range of potential app ideas that use CloudDB in the final project phase.

The four free response questions provided deeper insight into students' levels of understanding. Question 1, *In your own words or with pictures, describe what you think the Cloud is and what it does. Try not to use the examples from today*, had two levels of answers. Students either described the Cloud as a data storage mechanism or as a mechanism for storage, sharing, and editing. An answer of the first, simpler type of response was: *"I think Cloud is somewhere store the data or somewhere everyone put things there."* Figure 4-1 illustrates an answer of the second, more complex variety.



Figure 4-1: A student's response to Pre-questionnaire, *In your own words or with pictures, describe what you think the Cloud is and what it does. Try not to use the examples from today.*

The student drew the Cloud almost as an intermediary message broker for several different devices, which is a crucial part of many implementations of Cloud platforms. This student also demonstrated understanding that Cloud data can be shared across different device types. There was one student who left this question blank, indicating a lack of understanding.

Question 2 of the free response asked: *Do you think the Cloud has an important role in today's society? Why or why not?* All students responded positively. However, all of their explanations focused on document sharing or online collaboration, indicating that they were heavily influenced by the in-class example of Google Docs and may not have understood the full capacity of Cloud data. An answer representative of the general spirit of the responses was: *"Yes! It allows sharing important documents + work to go more smoothly. Also, it helps connect the world so people around the globe can all communicate together."*

Question 3, *What are some problems you see with the Cloud?* had an interesting breadth of responses. Note that issues with and drawbacks of shared data were not at all discussed during class. A few students responded with no ideas. The others wrote:

- *"It uses an external source to store the file."* This answer doesn't explain why an external storage is problematic, but we can infer that the student thinks this is bad, perhaps due to common concerns such as security, ease of access, and speed.

- *"May be too slow."* This is a very prevalent problem—speed is one of the top areas of optimization for Cloud services and products.
- *"Sometimes when the whole class is looking at the same doc it will kick some of the [students] out."* This response focuses on the Google Doc example of shared data, but touches on real-life challenges. With too many concurrent users, Google Docs, which uses the operational transformation algorithm, will be unable to enforce consistency across all edits due to server overload [24]. While this student did not have knowledge of these technical details, thinking up this problem case encouraged the student to initiate a conversation with me after the class. The student was able to think up the generally correct explanation that *"the Cloud computer can't handle talking to too many people at once."*
- *"If you don't have internet (wifi) you wouldn't be able to share something."*
- *"If the computer 'Cloud' were to crash or stop working, the data would be lost & people would not be able to work on it together."*
- *"What if the Cloud overloads or is hacked."*

The last three responses are all common issues that Cloud service providers struggle with. Within just a few minutes of learning what shared data was generally about, the vast majority of students in the class were able to think critically about data sharing and the problems that can come with it in the context of today's Internet landscape.

Students did not generate novel answers to question 4, *How might you use the Cloud in the future to create something that solves a problem you see in the world?* The few students who did write responses all wrote about applications in collaborative work and were clearly influenced by the Google Docs example, once again.

The Pre-questionnaire indicated that the vast majority of students were able to grasp the big ideas around shared data quickly (with just 10 minutes of explanation). They were generally confident in their understanding and showed some ability to

think critically about shared data in the larger context of the Internet. However, they were unable to think of use cases aside from the example given in class which shows a lack in depth of understanding and critical thinking.

4.2 Creating a class-wide shared drawing application successfully helped most students learn CloudDB

Students began using the CloudDB component in their own MIT App Inventor projects at the end of Lesson 2, when the DrawTogether class-wide app building activity was introduced (refer to Chapter 3.2). At the beginning of the activity, all students were sent a premade AIA file that created a working drawing app. The pre-made app allowed the user to draw on a picture on their own device. Students were tasked with figuring out how to use CloudDB to modify the app so that everyone in the class can draw on the same picture at the same time.

The final expected result was for students to use *CloudDB.StoreValue* and *when CloudDB.DataChanged*. Whenever they drew a line, that line would be pushed to CloudDB as four coordinate values in a list. The *when CloudDB.DataChanged* block would be triggered whenever anyone stored anything onto the CloudDB project. The value returned from that block would represent other users' new lines and would be automatically rendered on the local device. A worksheet (see Appendix C.3) served as a guide for students to think about how to proceed.

Step 2 of the worksheet asks students to consider the properties of CloudDB with respect to the app they are trying to build: *Do the AccountName and ProjectID have to be the same or can they be different for each app? Why?* After several minutes of free work time, the teacher asked students to volunteer their answers. Three students responded:

- **Student B:** *"Yes, same so everyone can see the same drawing."*
- **Student C:** *"No, because that's how the Cloud works."* When the teacher asked for elaboration, the student responded: *"Everyone has a different computer that*

talks to the Cloud computer."

- **Student D:** *"Absolutely yes, because if it's different, then the drawing will be stored on different Cloud computers."*

The correct answer was that all of the apps should have the same *AccountName* and *ProjectID* so that every app can push their changes and pull the changes made to the drawing by other apps. Student C seemed to have trouble understanding how the independent clients communicate with the common server through the CloudDB mechanism. However, after Student D's explanation, Student C's understanding was improved. When asked to explain their realization, the student again used the analogy of talking to the Cloud computer instead of the data buckets analogy that was explained in class:

"The AccountName and ProjectID are like the name of the Cloud computer."

— **Student C**

While this explanation isn't technically accurate from an implementation point of view, this level of abstraction was reasonable and appropriate for students' interaction with CloudDB. From my observations, it seems that children are often able to explain complex ideas to their peers more easily than adults. In this particular classroom setting, when one student understood and explained a concept using kid-friendly language, it helped others understand it better.

All of the students were able to complete the class-wide DrawTogether app by the end of Lesson 3, with various levels of teacher assistance. Most students quickly understood that the *CloudDB.StoreValue* block should be used to push their changes to the Cloud, but only one student was able to figure out that the *when CloudDB.DataChanged* block could be used to constantly listen for changes made by everyone else. Most students attempted to use the *CloudDB.GetValue* blockset instead. Some tried to put the *CloudDB.GetValue* command under where the *CloudDB.StoreValue* is called and others attempted to tell CloudDB to run the *CloudDB.GetValue* command every second or so (while this is possible with the Timer component, it is not the most elegant

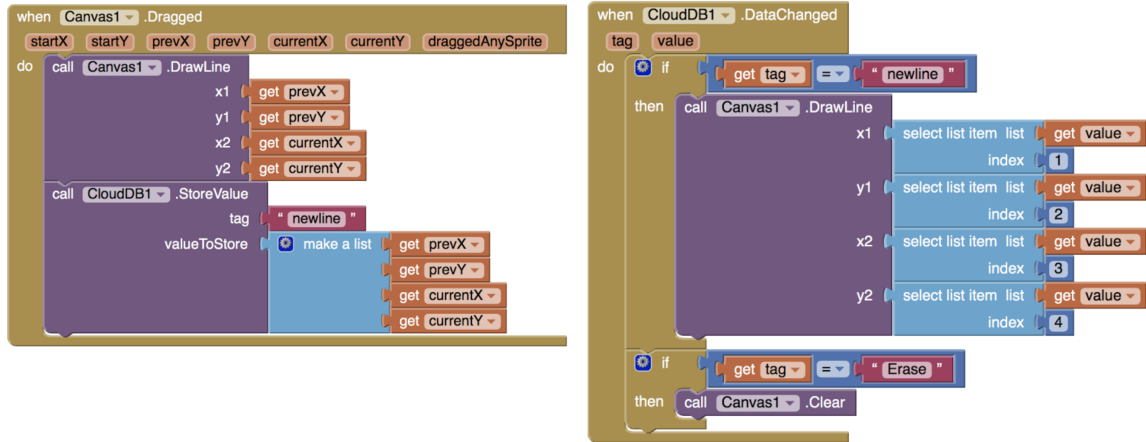


Figure 4-2: Code from the finished DrawTogether class-wide drawing app

or the easiest solution to implement).

I feel that this confusion mostly stems from an unfamiliarity with CloudDB blocks and their functions rather than a general lack of understanding regarding shared data. After finishing the core DrawTogether app, half of the class was able to implement the multi-color DrawTogether extension with varying degrees of success. The extension involved adding an extra element to the lists being stored and retrieved that held the value of the color that the new line should be drawn in.

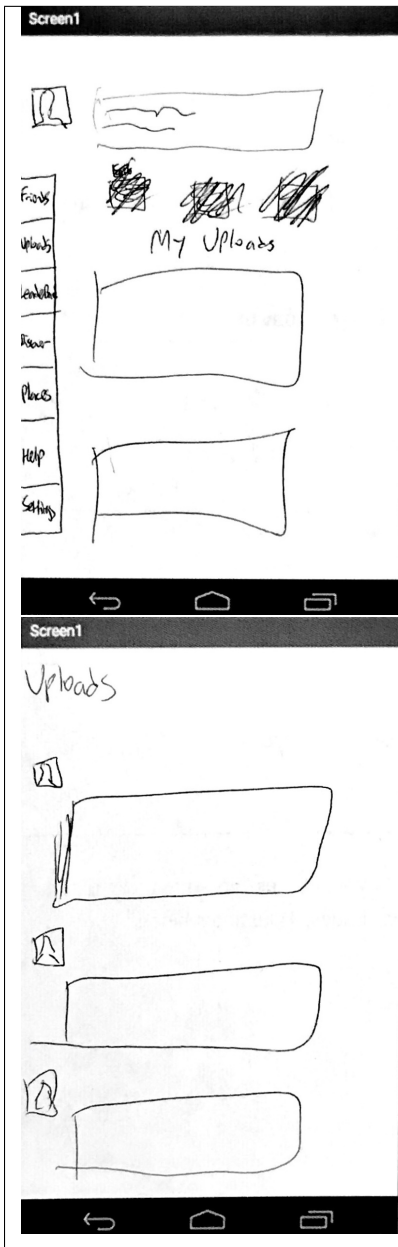
While students generally felt like they understood how to use the main functions in CloudDB through this activity, there were several pain points with respect to working with data structures. First, students needed to understand how the Canvas component handles line drawing with four parameters: **prevX** -> **x1**, **prevY** -> **y1**, **currentX** -> **x2**, **currentY** -> **y2** (see Figure 4-2). Then they needed prior knowledge of lists in MIT App Inventor so that they could store all of the values under a single tag to push to CloudDB (again, there are other solutions that are more complex and less elegant). Finally, they had to use built-in list functions to extract the elements needed to reconstruct a line.

There was only one introductory lesson prior to this activity when students could directly explore the MIT App Inventor website. During Lesson 1, students were encouraged to follow a variety of basic app tutorials of their choosing. Many did not come into contact with the Canvas or list components during that lesson. In

the future, if given more time, it might be good to make sure all of the students are exposed to creating apps that use Canvas and list prior to this activity so that students can remain focused on implementing CloudDB functionality instead of trying to figure out how the other features of the code works. Despite the struggles with Canvas and list, students were exposed to simple data structures through this activity, which is valuable for reasoning about abstraction with regard to shared data.

4.3 The final project design process indicated increased understanding of capabilities of shared data and strong interest in creating apps that influence the community

The final design project was centered around solving a problem related to the environment. Students were first asked to individually brainstorm problems that they were passionate about and draw screenshots of a few mobile app ideas based on their problem statements. Students were verbally enthusiastic in class about the idea of designing and creating their own apps, and the topic of the environment seemed to be something that all of the students cared about. There was an large variety of initial problems that students wrote out, including pollution, extinction, deforestation, and oil spills. This led to an interesting spread of app designs, a few of which I present in Table 4.2 below:



Two of three drawings for an app idea that would allow users worldwide to share pictures of beauty in nature through CloudDB. This student was inspired by the problem of littering because *"it is harmful for birds/other animals."* Users would be able to upload their own photos, tag photos by location, and view and favorite others' photos. This app idea is similar to the popular photo sharing social media apps of today.

	<p>The design for a two-player game that uses CloudDB. Two devices would communicate with each other through the Cloud during a fight sequence between a human and a tiger. This student wanted to combine three problems that they considered very important: littering, deforestation, and climate change. At the end of the game, the players would be shown an educational message about how deforestation <i>"will cause animals to die and ruin the earth's ecosystem"</i> or that <i>"animals are losing their habitats and humans are responsible."</i></p>
	<p>One of several similar drawings for an initial design of a simulation app that would show how <i>"oil spills effect many animals and poison their water, making the environment non-livable"</i> over time through a scrolling bar. Due to coming to class late, this student skipped the step in the worksheet asking them to discuss how CloudDB can be incorporated into their app ideas with a partner. However, the student seemed very passionate about educating others about this environmental problem.</p>

Table 4.2: App designs from individual design project worksheets.

From the individual design project worksheets, we can see that young students know a fair amount about a wide range of problems in the world. They are able to communicate on why they care about those problems and are familiar enough with technology and mobile capabilities to understand how apps can fit in with solving these problems. I also felt that students were able to expand their understanding of the capabilities of CloudDB and data sharing during this activity. By first starting

out with problems they want to solve and then discussing with peers and teachers about how data sharing could be incorporated into apps to solve these problems, students realized that data sharing is not limited to Google Docs-style collaboration as most of them indicated in the Pre-questionnaire. Students were able to imagine broader applications of CloudDB to a wider range of real-life issues by first trying to solve a problem that they were individually passionate about.

After students completed the individual brainstorming worksheet, they shared their ideas with at least two other students in the class and formed groups of two to three based on shared interests for the final group project. Students were given half of Lesson 4, all of Lesson 5, and half of Lesson 6 to work. Four groups were formed. Two groups created educational games, one group made a social networking type app, and one group tried to create a meta collaboration app that would help multiple people work on an MIT App Inventor project together.

4.4 Case study of one group's progression of constructing knowledge about CloudDB and shared data

Due to limited videocamera resources and classroom space, only one group near the corner of the classroom was filmed. This pair of students decided to make a two-player battle game focusing on deforestation and extinction. Analysis of the students' discussions provided insight into how their understanding of data sharing, data structures, and the client-server model was constructed over the course of their work.

First, the students tried to figure out how a player would be assigned to play as a certain character. They wanted the first player who opens the game to play as the human and the second player to play as the animal guardian of the forest. However, after several minutes of discussion, they decided that the process of entering the game was not as important as the game itself, so they chose to put aside the problem of implementing the lobby system after they coded the game mechanics.

The second decision the group made was with regard to how the game is played. The pair quickly agreed that, similar to an old-style arcade game, buttons would be used to attack on both sides. The pair had an interesting debate around whether to use turn-based versus continuous battle mechanics. Ultimately, the group decided to implement simultaneous button-mashing gameplay to deal damage and determine the winner. The conversation that led to this decision was centered around the abilities of CloudDB. A snippet of their conversation is transcribed below:

Student D: *"Um, I was thinking both attack. It's like turn-based. Because moving might be a little bit..."*

Student E: *"Laggy."*

Student D: *"Yeah it'd be kinda choppy."*

This pair clearly understood the problem with lag in data sharing, possibly from their experience with the DrawTogether app, which they reference later on in the conversation. They made a key decision regarding their app with that issue in mind, which demonstrates understanding and critical computational thinking.

The students then set up the non-CloudDB parts of their app, which took a significant amount of time. When the group was ready to add CloudDB functionality to their game, they referenced the code from the DrawTogether activity as well as another sample CloudDB AIA game to start out:

Student D: *"We can Cloud everything just like the paint app."*

Student E: *"We can use that one for reference as well. Except it's easier because we don't have to have the canvas."*

Student D: *"Well actually, the key is no matter what we do we have to start with the exact same code right?"*

Student E: *"Wait you can open up the app right? And play it correct? And the background will be the same. All we need to do is stream the damage... Like all we need to do is stream changes."*

This group successfully reused and remixed concepts about CloudDB that they learned during the DrawTogether activity regarding when and what data should be streamed. They concluded that not all of the game and app data should be sent to CloudDB—only the changes were necessary—and the changes would be reflected in the damage variables which are triggered by pressing the attack buttons.

Since the group wanted two players on two different devices to be running the same app with the same code instead of two apps with different "Tiger" and "Human" code, they needed to figure out how to store health and attack values for each player. The pair eventually reached an understanding of how to use two different global variables for a two-player game on one app, which is a relatively complex idea often presented at the beginner college computer science level:

Student D: *"When attack dot click do... We should make a variable called damage. A global name."*

Student E: *"We don't want one global damage, that'll be for both of them!"*

Student D: *"Global damage ONE and global damage two."*

Student E: *"Exactly, that's what I'm saying. Because then you would be hurting yourself AND your opponent."*

This group also had an interesting conversation over what data should go into the cloud and what data should only be stored locally. Initially, **Student D** said that *"We can use one CloudDB for practically anything"* and tried to put all of their app properties in CloudDB instead of just the ones that needed to be shared. They thought that if they put everything in the Cloud, then that would allow all of the devices running their app to see the same screen. While trying to code such an app later on, they quickly realized that storing the static parts of their app in the cloud causes the code to become more complicated than just leaving those portions locally on the client. They were able to instinctively learn a little bit about client-side versus network programming.

To end the game, the students realized that they could use a Cloud variable as an indicator:

Student E: *"One thing that's kind of important is that we need to tell [CloudDB] that if that value right there gets to zero," <points to health bar value> "it needs to tell you that you've lost or you won."*

Student D: *"So how do we make it stop once it gets to zero?"*

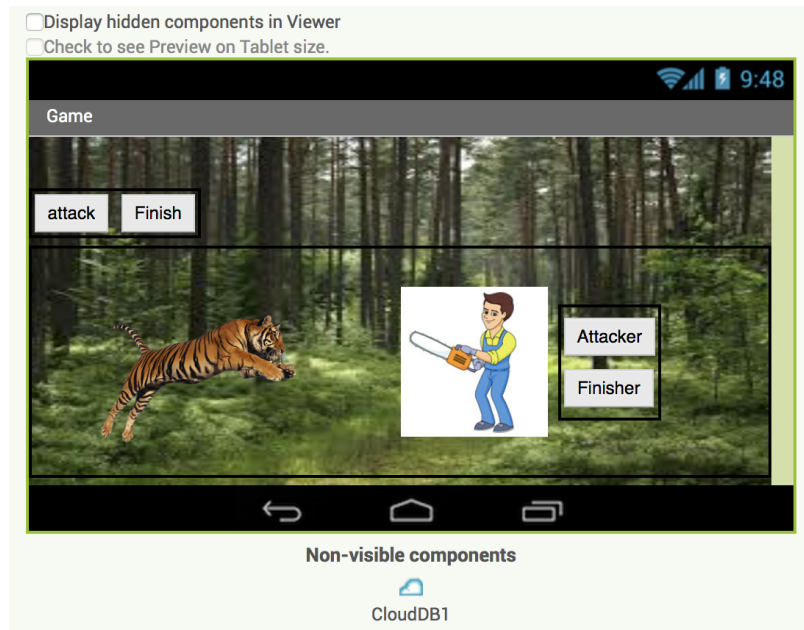
Teacher: *"In this if statement you can do an if."*

Student 2: *"I was gonna say if tag is greater than zero..."*

The students correctly modeled CloudDB as an intermediary messenger that could trigger events (in this case, ending the game) and send that information to two separate clients, who can process the same data in different ways.

By the end of the class, this group had created a multi-screen app that successfully decremented HP values of characters over CloudDB when attack buttons were pressed. The app also showed the players an environmental fact when the game ended. The students were beginning to build the code required to set up lobby management functionality but ran out of time. The CloudDB-relevant portions of their Designer screen and code are shown in Figure 4-3.

While this group constructed quite a lot of knowledge about data structures and the workings of CloudDB and was able to use MIT App Inventor relatively successfully to create a mostly functional app within the short amount of time they were given, I observed one main point of confusion. Both students consistently confused *tag* and *value*, seemingly using the term *tag* to refer to both parameters. When the students first began coding with CloudDB, they initially left the *value* field as an empty String and just set the *tag* to the values they wanted to share. While this is a valid way to use CloudDB when coupled with the *when CloudDB.DataChanged* block, these students were trying to use it in such a way that required differentiation. Anytime an attack button was hit by either player, only the *tag* would be changed to the appropriate amount of numerical attack damage being dealt, with no indication of which character



(a) Designer layout of the main gameplay screen of the app

```

initialize global Hp1 to 200
initialize global Hp2 to 150
initialize global damage1 to 0
initialize global damage2 to 0

when attack.Click
do
  set global damage1 to 15
  call CloudDB1.GetValue
  tag Hp2
  valueIfTagNotThere 150

when attacker.Click
do
  set global damage2 to 20
  call CloudDB1.GetValue
  tag Hp1
  valueIfTagNotThere 200

when finisher.Click
do
  set global damage2 to 50
  call CloudDB1.GetValue
  tag Hp1
  valueIfTagNotThere 200

when Finish.Click
do
  set global damage1 to 45
  call CloudDB1.GetValue
  tag Hp2
  valueIfTagNotThere 150

initialize global gameover to 0

when CloudDB1.GetValue
tag value
do
  if get tag == Hp2
  then
    if get value > 0
    then
      call CloudDB1.StoreValue
      tag Hp2
      valueToStore get value - get global damage1
    else
      set global gameover to 1
    else if get tag == Hp1
    then
      if get value > 0
      then
        call CloudDB1.StoreValue
        tag Hp1
        valueToStore get value - get global damage2
      else
        set global gameover to 2
  if get global gameover ≥ 1
  then
    open another screen screenName Fact
  
```

(b) Blocks code for communicating with CloudDB to enforce consistency in HP based on attack damage for two players of the game

Figure 4-3: Designer screen and CloudDB code from a two-player fighting game that was created for a group's final project

is attacking which. This made it impossible to determine which player lost HP. The students fixed this issue with the help of the teaching staff but continued to refer to the *value* as the *tag*. It was unclear whether this was due to a continuous lack of understanding or just using the terms interchangeably.

It is interesting to note that both students were able to create and use global variables in MIT App Inventor correctly. Perhaps it would improve understanding to teach students about variables first and then introduce the parallel of CloudDB's *tag* as some variable name and *value* as some variable's value.

4.5 Students were generally enthusiastic and optimistic about creating apps that would be used by their peers

While I only analyzed one group's app creation process in-depth, all four groups were very enthusiastic and motivated about their final projects. One group even exchanged contact information so that they could continue working on their project together after the workshop. Unfortunately, winter storm Theseus hit the Cambridge area on the day of the last presentation-focused lesson (April 1, 2017), which prevented half of the class from coming to MIT. Luckily, at least one person from each group was present, but only one group had full access to their code. Two other groups had some previous version of their code and one group had no access to their code—all of their code was on a missing group member's MIT App Inventor account and that individual did not share any backups. In any case, all of the groups gave a short presentation on what their group made and why:

1. Deforestation Game (see Figure 4-3 in Chapter 4.4)

Student D: *"So our idea is focusing on deforestation for kids our age because it's more of a video game than an information app."*

Student D: *"Well deforestation, it's destroying forests for humans and this wrecks the forest ecosystem and eventually since the human population keeps*

increasing, deforestation keeps on growing. And if we keep on deforestating we eventually run out of trees and our oxygen supply is limited and mass extinction."

Teacher: *"So what do you hope your users will get out of this app?"*

Student D: *"The consequences of deforestation because we have two characters, one is an animal protecting the tree and the other is a human trying to cut it down. If the human cuts it down it says congratulations you win but you run out of oxygen and the entire human race dies."*

Student E: *"And maybe have some fun. It doesn't really make sense but a player can win and the world goes bang!"*

Student D: *"That's actually what's going to happen. You win! But you destroyed the world!"*

Student E: *"And they learn you don't want to do that."*

2. MIT App Inventor Project Collaboration Tool

Student F: *"Our idea was to create a digital web board where various people who were connected to the same account could collaborate on one of their ideas. We didn't get that much done but we were trying to have one device where you could edit something and it sends it to the cloud and it displays it on all the other devices."*

3. Social Media/Best Cities in America (see Figure 4-4)

Student G: *"We wanted to do kind of like a social media service where you could post pictures of your city and the app will tell you the best cities in America environmental wise. I had to start over today. We made a list of the best cities and the worst cities and we wanted to use CloudDB to upload pictures."*

Teacher: *"So you were going to use CloudDB to store the ratings people had and aggregate them?"*

Student G: *"Yeah."*

4. Oil Spill Cleanup Game (see Figure 4-5)

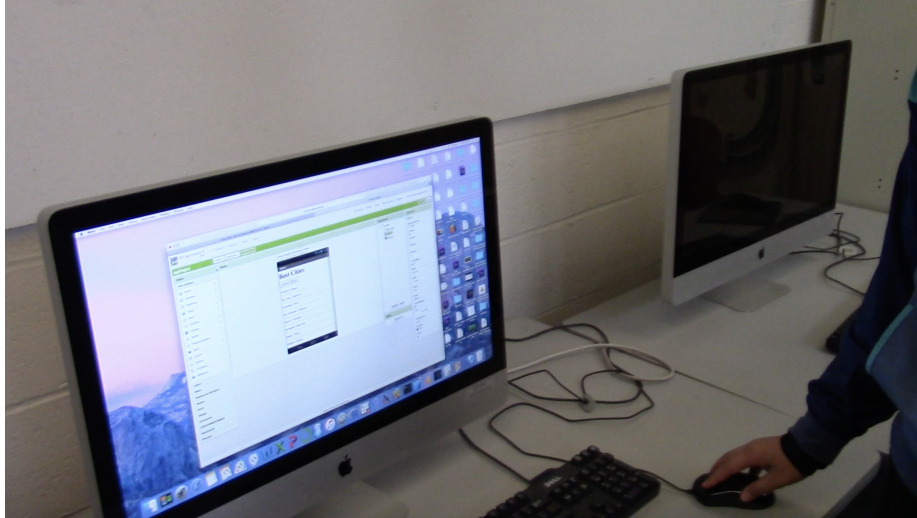


Figure 4-4: Student presenting final project on social media picture sharing and best city ranking app

Student H: *"Me and my partner made an app about oil spills, but unfortunately she wasn't here today and all our work was on her email so I just made a rough sketch of what it's like. So first you choose a team, team one or team two. And then you go to instructions to tell you what to do. And then when you press start it counts three two one and then it goes to the game. And you have three levels and on the first one you have a certain amount of time to clean up an oil spill and you can see how fast the other player is cleaning up. On the other screen like that one is yours and that one is the [other player's]. There are three levels and whoever wins two or three wins the whole game."*

Many students based their app ideas off of other apps they've used. During the group brainstorming session, students showed each other apps on their personal mobile phones or tablets to more clearly explain their ideas. Group 3's app had many similarities with Instagram combined with a ranking app [27]. Group 1 and 4's apps all had components from popular multi-player online or phone games. Group 2 was inspired by the difficulties they had being unable to work on the same MIT App Inventor project at the same time and wanted to create a CloudDB-based tool that could give MIT App Inventor Google Docs-like collaboration features.



Figure 4-5: Student presenting final project on oil spill cleanup game

Most of these apps targeted audiences in the same age group as the designers and also had an educational component. Students were engaged while listening to other groups' presentations and were enthusiastic about trying out the apps that their peers had made.

4.6 Post-questionnaires indicated high levels of satisfaction with making independent, creative apps

After the final presentations, students were asked to take an anonymous Post-questionnaire to measure what they took away from the final projects and whether their understanding and attitudes regarding the Cloud had evolved at all during the workshop. First, there was a block of five 5-point Likert scale questions: the first four were the same as the Likert scale questions in the Pre-questionnaire and the fifth was a new question measuring students' thoughts about the workshop. Second, there was a block of three free response questions about the final project. Finally, there was a block of four free response questions about the Cloud which were the same as the free response questions on the Pre-questionnaire. As in the Pre-questionnaire, when analyzing their responses to the Likert questions, I coded **Strongly Agree** as 5 and **Strongly Disagree** as 1. The mean score and range of scores are displayed in Table

4.3 below.

#	Question	Mean	Range
1.	I use the Cloud in my daily life.	4.5	4-5
2.	I understand how the Cloud stores and shares data.	4	4
3.	I feel comfortable making apps that use the Cloud.	3.778	3-4
4.	I can think of ways that the Cloud can solve problems in my everyday life.	4.167	3-5
5.	My understanding of the Cloud improved through these workshops.	4.5	4-5

Table 4.3: Mean and range of responses for Post-questionnaire Likert scale questions on a 1-5 scale

The class only had 10 students to begin with, and due to the additional decrease in attendance caused by snow storm Theseus, the final survey results may not be accurate in representating the whole class; a single student's response could easily skew the whole data set. We can generally see that the responses for the first four questions did not differ much from that of the Pre-questionnaire, which were already relatively high. The responses for the final question were also high, in the range of **Strongly Agree** to **Agree**, indicating that the workshop was generally a positive learning experience.

The first set of three free response questions showed that students unanimously enjoyed and felt confident about the final project. Question 1 asked: *How do you feel about the app you made?* All of the students gave very positive responses, such as *"I think we had a good idea and there was a lot of potential,"* and *"I think it came out well for the time we had."* These responses also generally indicated that students did not feel like they had enough time to carry out their app vision.

Question 2 of the free response asked: *What was hard about developing the app?* One student left this question blank, possibly implying that they didn't have much trouble. The pain points that most of the other students mentioned had to do with

operating the MIT App Inventor tool. Since students only had two 50-minute lessons where they could directly interact with the MIT App Inventor website prior to the final project, it is unsurprising for them to still have difficulties using all of the components and blocks properly, especially since many of the students needed to use more complex features of MIT App Inventor to make the apps that they had imagined. One student said: *"I think there was a big learning curve to using the app inventor."* Another student said that *"arranging the pictures"* was difficult. None of the students mentioned Cloud concepts as something that was difficult to understand or code.

Question 3, *What were some things you didn't expect?* had an interesting breadth of responses:

1. *"How long some parts might take."*
2. *"I didn't expect how much work went into something as simple as a button on an app."*
3. *"I didn't expect how long it would take to implement the app with code."*
4. *"I expected that App Inventor would be more like Scratch but it wasn't."*
5. *"It was so much easier."*
6. *"That we could use one Cloud block for all our damage."*

3 responses (the first, second, and third) indicated that students felt that creating an app was harder than they thought it would be. 2 responses (the fifth and sixth) indicated the exact opposite. This dramatic variation in experiences may have been due to either pre-existing notions about app programming or the difficulty of the final project app that the student's group had been trying to make.

The next block of free response questions was a repeat of the shared data knowledge evaluation questions from the Pre-questionnaire. For question 1, *In your own words or with pictures, describe what you think the Cloud is and what it does*, all of the responses were picture-based. In the Pre-questionnaire, most of the responses

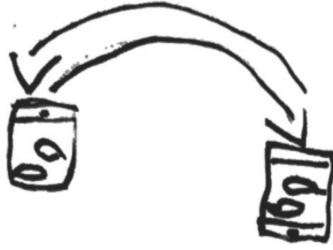


Figure 4-6: A student's response to Post-questionnaire, *In your own words or with-pictures, describe what you think the Cloud is and what it does. Try not to use the examples from today.*

had been text-based—only two students answered using pictures. Most of the pictures were similar to Figure 4-1 from the Pre-questionnaire shown in Chapter 4.1. One interesting variation was that one student drew two mobile devices sending data back and forth. This picture completely abstracts out the intermediary Cloud server that we taught during class and likely better reflects how the student's interaction with the CloudDB code helped them construct how data sharing is applied.

Question 2 asked: *Do you think the Cloud has an important role in today's society? Why or why not?* Students unanimously replied with "Yes" like in the Pre-questionnaire. However, while the majority of the responses in the Pre-questionnaire mentioned Google Docs-like functionality to explain why shared data is important, there was a wider variety of explanations in the Post-questionnaire. Two examples of responses were:

- *"Yes, it is very helpful for passing data."*
- *"Yes, it is used in various google software and in other electronics."*

These responses are much broader than just the application of Google Docs and indicates that students have a better understanding of the capabilities of shared data and how it fits in within the larger Internet ecosystem.

Question 3, *What are some problems you see with the Cloud?* had several different responses compared to in the Pre-questionnaire. Four particular responses were:

1. *"Data can be easily manipulated which can be bad for people who use the cloud for bad things."*

2. *"Data could be changed in the Cloud."*
3. *"It is hard to use."*
4. *"If somebody gets a hold of the original project they could manipulate it."*

The first, second, and fourth responses were likely conclusions formed from the students' interactions with CloudDB. While not all Cloud data sharing systems are easily accessible or hackable, many are. CloudDB projects can be accessed by any MIT App Inventor developer by design. This encourages cross-project and cross-developer usage of data for projects, and makes creating specific data-facing apps much easier than with FirebaseDB. However, it is important that students are able to recognize the security and data management issues that come with this feature. The first response also implies that the student found working with CloudDB to be easy, which is a plus for CloudDB's usability. However, the third response suggested the opposite.

Question 4, *If you were to use Cloud technology in the future to create something that solves a problem you see in the world what would it be?* generated several creative responses, compared to the total lack of novel responses in the Pre-questionnaire. Two particularly interesting responses were:

- *"Using Cloud tech to create a car that will learn driving routes and store them (basically making a self-driving car that doesn't need a gps)"*
- *"We could use it to inform people about a problem so they can work together to solve it."*

The first response shows that the student is able to think of shared data not only in the context of software but also in the context of hardware systems that can interact with the physical world. The second response demonstrates an interest in motivating people and helping them organize to solve problems in the world, which shows that the student is at least somewhat interested in using Cloud technology to have a positive effect on their community.

The Post-questionnaire suggests that the workshop as a whole and the final project in particular was a positive learning experience for students. Students had more varied and more complex responses to the same free response questions as the Pre-questionnaire, which also indicates that this workshop improved their knowledge and critical thinking skills with regard to the Cloud. They demonstrated that they understood the role of shared data in society in a broader sense (not just as a way to make Google Docs work) and showed expanded ability in reasoning about shared data both in the context of networking and in the context of how it can be used to improve the world.

4.7 Workshop was generally effective but could have been longer to allow for more complete final projects

As the differences between the Pre- and Post-questionnaires showed, the workshop was generally an enjoyable and effective way to teach students about shared data. However, some key suggestions for improvement were collected from the students, teaching staff, and researchers over the course of this project.

First, students did not have enough time to introduce or familiarize themselves with MIT App Inventor and its wide array of components. Since CloudDB is a relatively advanced component and is always used in conjunction with other components in order to manipulate or display the shared data, being familiar with MIT App Inventor was almost a pre-requisite. Either students should have much more time to learn MIT App Inventor's other components, or advanced MIT App Inventor users should be chosen for such a workshop in the future.

Second, while students were able to quickly grasp the general idea behind shared data and CloudDB, they had slightly less success at understanding the detailed workings of CloudDB. The DrawTogether activity helped quite a lot in improving students' knowledge of how specific blocks are used, but more hands-on examples may have been helpful. Additionally, as seen from the DrawTogether unit, it may be useful to give

students a quick lesson on basic data structures in MIT App Inventor so that they are able to organize their data through CloudDB.

Finally, the amount of time given for students to complete their final projects was not nearly enough. While most groups almost had a working minimum viable version of the app that they had envisioned, it is likely that giving them one or two extra lessons worth of time would have resulted in complete first drafts of working apps. However, we also noticed that most of the time students spent building their apps on MIT App Inventor was focused on the non-CloudDB-related parts, including aesthetics, game mechanics, and basic interactions. For example, students could not find the if/then blocks and didn't know how to navigate across screens. Due to unfamiliarity with MIT App Inventor, students spent a lot of time trying to code the rudimentary framework required before the CloudDB component could even be added. This pain point might be alleviated when running this workshop with advanced MIT App Inventor users instead of students who had never even seen the tool before.

Chapter 5

Discussion: The Cloud has powerful implications for computational thinking education for young students

The core vision of this work is to empower young students to become creators of technology instead of just consumers of it, specifically focusing on helping them create mobile applications that harness the power of the Cloud. To achieve this, I: (1) built CloudDB, a data sharing component for MIT App Inventor, a popular blocks-based mobile application programming tool commonly used in lower education, (2) designed a six-week workshop to teach mobile application creation using shared data through the MIT App Inventor tool, and (3) ran this workshop on a cohort of ten middle school students with minimal prior coding experience. My key findings are summarized below:

- **CloudDB is able to securely store, retrieve, and modify data through a Redis server.** It has two properties and fourteen blocks that allow complex commands such as a *when DataChanged* listener and atomic list operations on various data types, including text, list, and media files.
- **Middle school students can easily pick up how to use the CloudDB component.** All students were able to complete the DrawTogether class-wide

drawing app within 50 minutes of learning the basics of CloudDB. While mastery of the component may involve prior knowledge of data structures, experience with MIT App Inventor, and some extra practice, the *CloudDB.StoreValue*, *CloudDB.GetValue*, and *CloudDB.DataChanged* blocks were easily understood and applied as seen through quick and correct usage by MIT App Inventor amateurs.

- **Young people are passionate about designing their own technological solutions to problems they see in the world.** When the final design project was introduced during class (using CloudDB to create an app that is related to the environment), students displayed immediate enthusiasm. Their worksheets indicated a breadth of understanding with respect to common environmental problems, as well as creative app solutions that spoke well to audiences in their age range. Students were motivated through the project building process and some even wanted to continue after the workshop was over.
- **Students were able to build reasonably complex and complete apps given a short timeframe.** With just 100 minutes of coding time, all groups had developed semi-working apps based off of their initial vision.

We have seen through these workshops that it is fully possible to teach students as young as middle school aged about shared data at a level of abstraction where they can apply it to real app development problems. Additionally, young students seem to be naturally interested in designing and implementing technology that can improve their communities and lives. Since it is clear that students are able to understand and use shared data, even with minimal programming experience, it would perhaps be appropriate to introduce the CloudDB component at the tail end of MIT App Inventor classes in schools.

Being able to share and manipulate data is a powerful ability for young app developers. It gives students the potential to create applications that can have a large reach in connecting devices to people in their communities and all across the world. Through the Post-questionnaire, we saw that just from working with CloudDB

in MIT App Inventor for 150 minutes, students were able to imagine abstract and creative ways to build new technologies and contribute using shared data, such as by programming a GPS-less self-driving car or crowdsourcing problem-solving. I believe that today's primary school aged students, who have grown up surrounded by technology and mobile apps, will be able to understand complex concepts such as shared data much more quickly, and be able to contribute to the world through technological creation more adeptly.

Chapter 6

Future Works: Improving security, providing access restrictions to project data, developing log viewing capabilities, and extensions to educational research

There are many extensions to this work on the technical side of improving CloudDB and many further questions that can be asked in educational research. First, although the current level of security in CloudDB ensures that random adversaries on the Internet will have trouble accessing Redis server data, anyone who does use the CloudDB extension in MIT App Inventor will be able to gain access to any user's project data. Currently in CloudDB, all data is stored as key-value pairs. The key consists of the *AccountName* property, the *ProjectID* property, and the *tag*. As discussed in Chapter 2.3, any developer can create a project and set the properties of a CloudDB component to reflect those of a project they wish to view or modify. In the future, it would be wise to allow developers to have more control over how and with whom their project data is shared. Such a system would need to enforce access separation

Which computer do you like better? » Cloud data history				
User	Data Name	Action	Data Value	Time
Sorrelwhisker	☞ Windows	set_var	9	1 month, 3 weeks ago
Sorrelwhisker	☞ Macintosh	set_var	2	1 month, 3 weeks ago
Sorrelwhisker	☞ Windows	set_var	8	1 month, 3 weeks ago
PinotPie	☞ Windows	set_var	7	2 months, 1 week ago
PinotPie	☞ Windows	set_var	6	2 months, 1 week ago
goldfish678	☞ Windows	set_var	5	4 months ago
ClayGuy	☞ Windows	set_var	4	4 months, 1 week ago
kyle7	☞ Windows	set_var	3	4 months, 2 weeks ago
woofwoof2	☞ Windows	set_var	2	4 months, 2 weeks ago
sabynlove	☞ Macintosh	set_var	1	4 months, 2 weeks ago
Pokecollets	☞ Windows	set_var	1	4 months, 2 weeks ago

Figure 6-1: A sample cloud data log in Scratch, which displays when, how, and by whom a cloud variable was modified [29].

based on client ID and would require client identity validation. Two potential ways to implement this feature would be to (1) create a new identity management service for clients, and (2) take advantage of Redis’s ability to isolate applications’ keyspaces on a per-user level and a per-application per-user level. Data sharing settings could be set through interaction with the identity management service, which would permit collaboration on public data buckets and restrict access to private data buckets.

Second, log viewing capabilities for CloudDB would be useful both as a function and for conducting research in educating students about Cloud data. I plan to build a webpage that app developers can use to view their CloudDB data buckets. For example, the webpage could show the list of tags and values for each of the developer’s buckets as well as a last changed timestamp and a brief history. A similar feature was implemented for Scratch Cloud variables by Sayamindu Dasgupta [29]. The feature was called "Cloud Data History" and displayed public logs showing the time cloud data was modified, who modified the data, the current value of the data, and the type of action of the data modification (see Figure 6-1). Through looking at these logs, Scratch programmers were able to find instances of cheating in the games that they had created.

On the education side, my experience from this workshop indicated that it might be better to conduct the CloudDB app development study with students who have already had a decent amount of experience using the MIT App Inventor tool, as the learning curve for the tool itself is somewhat steep due to the massive number of components available. If the workshop was offered as an optional extension at the end of an MIT App Inventor curriculum already taught in schools, students may be able to develop more complex final projects in a shorter time frame due to accumulated knowledge about the non-CloudDB components in MIT App Inventor. And obviously, a larger sample size would lead to more conclusive results.

Additionally, MIT App Inventor is currently working on collaborative programming, which can be used for group projects so that all members have access to the code and can work concurrently. It might be interesting to run the workshop again using the new collaboration tools to see how simultaneous programming instead of pair programming influences the work process and helps students understand concepts.

Appendix A

Lua code for CloudDB atomic list
functions

```

private static final String APPEND_SCRIPT =
    "local key = KEYS[1];" +
    "local toAppend = ARGV[1];" +
    "local currentValue = redis.call('get', key);" +
    "local newTable;" +
    "if (currentValue == false) then " +
    "  newTable = {};" +
    "else " +
    "  newTable = cJSON.decode(currentValue);" +
    "  if not (type(newTable) == 'table') then " +
    "    return error('You can only append to a list');" +
    "  end " +
    "end " +
    "table.insert(newTable, toAppend);" +
    "local newValue = cJSON.encode(newTable);" +
    "redis.call('set', key, newValue);" +
    "return redis.call('get', key);";

@SimpleFunction(description = "Append a value to the end of a list atomically. " +
    "If two devices use this function simultaneously, both will be appended and no " +
    "data lost.")
public void AppendValueToList(final String tag, final Object itemToAdd) {
    checkAccountNameProjectIDNotBlank();

    Object itemObject = new Object();
    try {
        if(itemToAdd != null) {
            itemObject = JsonUtil.getJsonRepresentation(itemToAdd);
        }
    } catch(JSONException e) {
        throw new YailRuntimeError("Value failed to convert to JSON.", "JSON Creation Error.");
    }

    final String item = (String) itemObject;
    final String key = accountName + projectID + tag;

    Thread t = new Thread() {
        public void run() {
            Jedis jedis = getJedis();
            try {
                jedis.eval(APPEND_SCRIPT, 1, key, item);
            } catch(JedisException e) {

            } finally {
                jedis.close();
            }
        }
    };
    t.start();
}

```

```

private static final String POP_FIRST_SCRIPT =
    "local key = KEYS[1];" +
    "local currentValue = redis.call('get', key);" +
    "local decodedValue = cJSON.decode(currentValue);" +
    "if (type(decodedValue) == 'table') then " +
    "  local removedValue = table.remove(decodedValue, 1);" +
    "  local newValue = cJSON.encode(decodedValue);" +
    "  redis.call('set', key, newValue);" +
    "  return removedValue;" +
    "else " +
    "  return error('You can only remove elements from a list');" +
    "end";
@SimpleFunction(description = "Return the first element of a list and atomically remove it. " +
    "If two devices use this function simultaneously, one will get the first element and the " +
    "the other will get the second element, or an error if there is no available element. " +
    "When the element is available, the \"FirstRemoved\" event will be triggered.")
public void RemoveFirstFromList(final String tag) {
    checkAccountNameProjectIDNotBlank();

    final String key = accountName + projectID + tag;

    Thread t = new Thread() {
        public void run() {
            Jedis jedis = getJedis();
            try {
                FirstRemoved(jedis.eval(POP_FIRST_SCRIPT, 1, key));
            } catch (JedisException e) {

            } finally {
                jedis.close();
            }
        }
    };
    t.start();
}

```


Appendix B

Assent and Consent forms for data release to research

B.1 Minor Assent Form

ASSENT TO PARTICIPATE IN RESEARCH

Using Cloud Databases in App Inventor

1. The researcher responsible for this study's name is Natalie Lao, a student at the Massachusetts Institute of Technology (MIT).
2. We are asking you to take part in a research study because we are trying to learn more about how to teach programming and a new technology called the "Cloud" to students in a better way, using a blocks-based coding language.
3. If you agree to be in this study we will ask you to do one or more of the following things:
 - Answer written questions about what you've learned during the workshops.
 - Use a computer program that was designed to make Android mobile apps.
 - Build apps and discuss your app-building process with your peers.

 - You will also have the option to agree to be video recorded during the workshop.
4. We do not believe that there are any significant risks to you if you participate in the study. We will make sure that your personal information is not shared with others, and will never use your name when we talk about this research study.
5. We hope that participating in this study will give you the abilities to make basic mobile apps for Android, which may be a fun experience.
6. Please talk this over with your parents before you decide whether or not to participate. We will also ask your parents to give their permission for you to take part in this study. But even if your parents say "yes" you can still decide not to do this.
7. If you don't want to be in this study, you don't have to participate. Remember, being in this study is up to you and no one will be upset if you don't want to participate or even if you change your mind later and want to stop.
8. You can ask any questions that you have about the study now. If you have a question later that you didn't think of now, you can call the lead researcher, Natalie Lao, at 617-866-8304, email her at Natalie@csail.mit.edu, or ask me next time. You can also call the Chairman of the Committee on the Use of Humans as Experimental Subjects at M.I.T. at 1-617-253 6787 if you feel you have been treated unfairly.
9. Signing your name at the bottom means that you agree to be in this study. You and your parents will be given a copy of this form after you have signed it.

ASSENT TO PARTICIPATE IN RESEARCH

Please check the boxes next to the things that you are agreeing to participate in

- The general study, which includes:
 Answering written questions about what you've learned during the workshops.
 Using a computer program that was designed to make Android mobile apps.
 Building apps and discuss your app-building process with your peers.

- Being video recorded during the workshop.

Name of Subject (Student's Name)

Student's Signature

Date

SIGNATURE OF INVESTIGATOR

In my judgment the subject is voluntarily and knowingly giving informed consent and possesses the legal capacity to give informed consent to participate in this research study.

Signature of Investigator

Date

B.2 Parental Consent Form

**CONSENT TO PARTICIPATE IN
NON-BIOMEDICAL RESEARCH**

Using Cloud Databases in App Inventor

Parent/Guardian of Student Consent Form

Your child is asked to participate in a research study conducted by Natalie Lao from the App Inventor Laboratory at the Massachusetts Institute of Technology (MIT). Your child was selected as a possible participant in this study because he/she is between the ages of 11-15 and is currently a middle school or high school student. You should read the information below, and ask questions about anything you do not understand, before deciding whether or not your child should participate.

• **PARTICIPATION AND WITHDRAWAL**

Your child's participation in this study is completely voluntary and he/she is free to choose whether to be in it or not. The choice to participate or not will have no impact on your child's grades. If he/she chooses to be in this study, he/she may subsequently withdraw from it at any time without penalty or consequences of any kind. The investigator may withdraw your child from this research if circumstances arise which warrant doing so.

• **PURPOSE OF THE STUDY**

The purpose of this study is to develop and test a new mobile application coding tool, which uses a blocks-based language to make mobile Android apps that use the Cloud. It is our hope that this tool will support students interesting in programming and app development.

• **PROCEDURES**

If your child volunteers to participate in this study, we would ask him/her to do one or more of the following things:

- Answer written questions about what you've learned during the workshops.
- Use a computer program that was designed to make Android mobile apps.
- Build apps and discuss your app-building process with your peers.

Your child will also have the option to agree to be video recorded during the workshop.

The workshops will take place in a location reserved at MIT for either 50 minutes or 2 hours.

• **POTENTIAL RISKS AND DISCOMFORTS**

We do not foresee any risks, beyond minimal, associated with participation in this study.

• **POTENTIAL BENEFITS**

We believe that your child could learn how to make basic mobile apps for Android, which may be a fun experience. However, we cannot promise any direct benefits directly associated with participation in this study.

If successful, this project will lead to the development and release of a free new coding tool that interfaces with the Cloud for the MIT App Inventor mobile application development interface. Our hope is that we can significantly improve student learning and innovation in technology, and

CONSENT TO PARTICIPATE IN NON-BIOMEDICAL RESEARCH

create fun, effective tools for students to use.

- **CONFIDENTIALITY**

Any information that is obtained in connection with this study and that can be identified with your child will remain confidential and will be disclosed only with your permission or as required by law. Your child's name and identifying information will not be disclosed to any other agency or group.

For dissemination purposes such as publications, technical reports and conference presentations, all names will be given a pseudonym/ID number and other characteristics that may identify your child will be changed.

All research data will be stored in the researchers' office at MIT in a locked storage locker and/or password protected hard drive. Data will only be accessible by authorized personnel.

- **IDENTIFICATION OF INVESTIGATORS**

If you have any questions or concerns about the research, please feel free to contact Natalie Lao, the lead researcher for this study. Natalie can be reached at 617-866-8304 or at natalie@csail.mit.edu.

- **EMERGENCY CARE AND COMPENSATION FOR INJURY**

If your child feels that they have suffered an injury, which may include emotional trauma, as a result of participating in this study, please contact the person in charge of the study as soon as possible.

In the event your child suffers such an injury, M.I.T. may provide itself, or arrange for the provision of, emergency transport or medical treatment, including emergency treatment and follow-up care, as needed, or reimbursement for such medical services. M.I.T. does not provide any other form of compensation for injury. In any case, neither the offer to provide medical assistance, nor the actual provision of medical services shall be considered an admission of fault or acceptance of liability. Questions regarding this policy may be directed to MIT's Insurance Office, (617) 253-2823. Your insurance carrier may be billed for the cost of emergency transport or medical treatment, if such services are determined not to be directly related to your child's participation in this study.

- **RIGHTS OF RESEARCH SUBJECTS**

Your child is not waiving any legal claims, rights or remedies because of their participation in this research study. If you or your child feels that they have been treated unfairly, or you or your child have questions regarding your child's rights as a research subject, you may contact the Chairman of the Committee on the Use of Humans as Experimental Subjects, M.I.T., Room E25-143B, 77 Massachusetts Ave, Cambridge, MA 02139, phone 1-617-253 6787.

SIGNATURE OF RESEARCH SUBJECT OR LEGAL REPRESENTATIVE

**CONSENT TO PARTICIPATE IN
NON-BIOMEDICAL RESEARCH**

I understand the procedures described above. My questions have been answered to my satisfaction, and I agree to allow my child to participate in this study. I have been given a copy of this form.

I consent to allow my child to participate in the following activities (**please check boxes next to activities you will allow**)

- The general study, which includes:
 Answering written questions about what you've learned during the workshops.
 Using a computer program that was designed to make Android mobile apps.
 Building apps and discuss your app-building process with your peers.

- Being video recorded during the workshop.

Name of Subject (Student)

Name of Legal Representative (Parent/Guardian)



Signature of Legal Representative (Parent/Guardian) _____
Date

SIGNATURE OF INVESTIGATOR

In my judgment the subject is voluntarily and knowingly giving informed consent and possesses the legal capacity to give informed consent to participate in this research study.

Signature of Investigator _____
Date

Appendix C

Supplemental Workshop Handouts

C.1 Pre-questionnaire

Survey

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree
I use the Cloud in my everyday life					
I understand how the Cloud stores and shares data					
I feel comfortable making apps that use the Cloud					
I can think of ways that the Cloud can solve problems in my everyday life					

Free response

In your own words or with pictures, describe what you think the Cloud is and what it does. Try not to use the examples from today.

Do you think the Cloud has an important role in today's society? Why or why not?

What are some problems you see with the Cloud?

How might you use the Cloud in the future to create something that solves a problem you see in the world?

C.2 CloudDB Handout

CloudDB MIT App Inventor Components

Properties

CloudDB1

AccountName

ProjectID

The AccountName and Project ID that you want to access on the Cloud computer.

The **AccountName** should be the **email** of the person who owns the data and the **ProjectID** should be the name of their project.

call CloudDB1 .StoreValue
tag
valueToStore

When you want to share data with other people by putting it on the Cloud computer, use the **StoreValue** block.

The **tag** should be a **text** describing the data you are sharing (ex: myDrawing). The **valueToStore** should be the **data you want to share** (ex: text, picture, sound).

call CloudDB1 .GetValue
tag
valueIfTagNotThere

When you want to get data from the Cloud computer, use the **GetValue** block. Put the **tag** of the data you want to get. Fill in the **valueIfTagNotThere** with the value you get if the tag you want is not in the Cloud computer.

when CloudDB1 .GotValue
tag value
do

After you use the **GetValue** block, make sure to include the when **GotValue** block. This block will tell you the **tag** and **value** of the data that you requested when you used the **GetValue** block once the Cloud computer sends you that data.

call CloudDB1 .ClearTag
tag

When you want to **delete** a **tag**, use the **ClearTag** block and tell it which tag to delete.


```
when CloudDB1 .DataChanged
  tag value
do
```

When anything on the Cloud computer changes, the **DataChanged** block will tell you **which tag was changed** and **the value that it was changed to**.

This happens when you store a tag/value pair and when you clear a tag.

```
call CloudDB1 .GetTagList
```

Use the **GetTagList** block when you want a list of all the tags in your project.

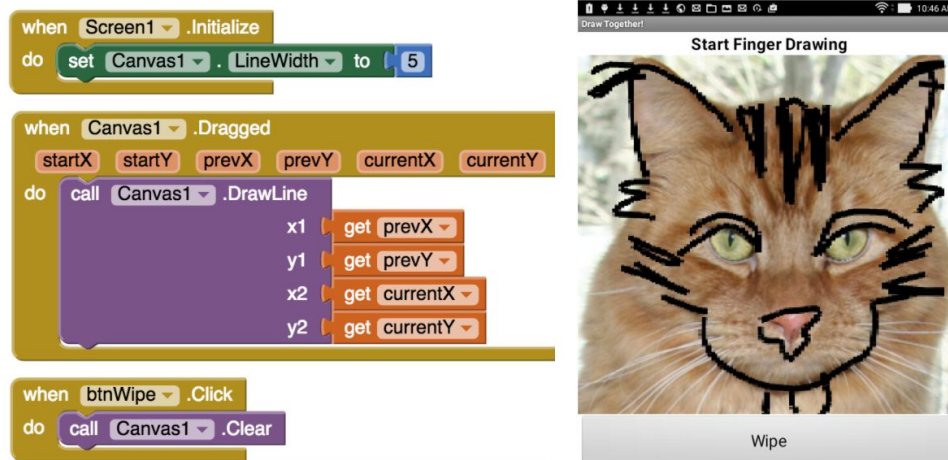
```
when CloudDB1 .TagList
  value
do
```

After using the **GetTagList** block, use the **when TagList** block to see the list of tags in your project. The list can be accessed in the **value** part of this block.

C.3 Draw Together App Design Worksheet

Draw Together App Design

1. We want to make the simple Draw Together app, which lets you and your friends draw in black and to erase all the markings. Open the AIA project in App Inventor. It is currently just a single-player app that is not connected to the Cloud. Play with it and look at the blocks.

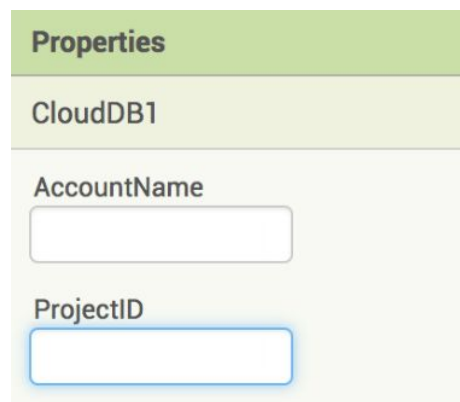


The image shows the App Inventor interface. On the left, there are three event-driven code blocks:

- when Screen1.Initialize**
 - do **set Canvas1.LineWidth to 5**
- when Canvas1.Dragged**
 - startX, startY, prevX, prevY, currentX, currentY
 - do **call Canvas1.DrawLine**
 - x1: **get prevX**
 - y1: **get prevY**
 - x2: **get currentX**
 - y2: **get currentY**
- when btnWipe.Click**
 - do **call Canvas1.Clear**

On the right, a mobile app preview is shown. The app has a title bar "Draw Together" and a main title "Start Finger Drawing". The screen displays a photo of a ginger cat with black hand-drawn outlines for its ears, whiskers, and eyes. At the bottom, there is a button labeled "Wipe".

2. You will need to use CloudDB to make this app multi-player. If everyone in the class is building an app to draw together, do the AccountName and ProjectID have to be the same or can they be different for each app? Why?



The image shows the "Properties" panel for a CloudDB1 component in App Inventor. It contains two text input fields:

- AccountName**: A text input field with a light gray border.
- ProjectID**: A text input field with a light blue border.

3. Make a plan for how you might code this app to share drawing data. How do you think you can use these blocks? Where should they go? What information do you need to pass to CloudDB? In what format and how often? Write or draw out your ideas in the space below. (Hint: Look at the CloudDB handout to figure out what blocks to use)

C.4 Draw Together Solution and Extension

Drawing Together App Solution

```
when Canvas1 .Dragged
  startX startY prevX prevY currentX currentY draggedAnySprite
do
  call Canvas1 .DrawLine
    x1 get prevX
    y1 get prevY
    x2 get currentX
    y2 get currentY
  call CloudDB1 .StoreValue
    tag "newline"
    valueToStore make a list
      get prevX
      get prevY
      get currentX
      get currentY
```

```
when CloudDB1 .DataChanged
  tag value
do
  if get tag = "newline"
  then
    call Canvas1 .DrawLine
      x1 select list item list get value index 1
      y1 select list item list get value index 2
      x2 select list item list get value index 3
      y2 select list item list get value index 4
  if get tag = "Erase"
  then
    call Canvas1 .Clear
```

```
when EraseButton .Click
do
  call Canvas1 .Clear
  call CloudDB1 .StoreValue
    tag "Erase"
    valueToStore "Erase"
```

Hint for Adding Multiple Colors: This is what should be changed in the Blocks space

```
when Canvas1 .Dragged
  startX startY prevX prevY currentX currentY draggedAnySprite
do
  call Canvas1 .DrawLine
    x1 get prevX
    y1 get prevY
    x2 get currentX
    y2 get currentY
  call CloudDB1 .StoreValue
    tag "newline"
    valueToStore make a list
      get prevX
      get prevY
      get currentX
      get currentY
```

```
when CloudDB1 .DataChanged
  tag value
do
  if get tag = "newline"
  then
    set Canvas1 . PaintColor to select list item list get value
      index 5
    call Canvas1 .DrawLine
      x1 select list item list get value
        index 1
      y1 select list item list get value
        index 2
      x2 select list item list get value
        index 3
      y2 select list item list get value
        index 4
  if get tag = "Erase"
  then
    call Canvas1 .Clear
```

C.5 CloudDB Individual Design Project Worksheet

CloudDB Individual Design Project Worksheet

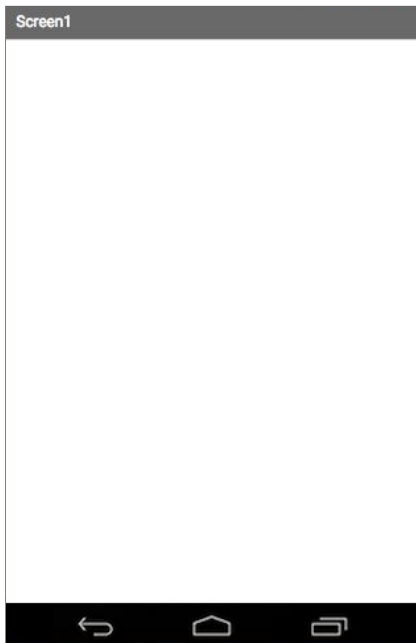
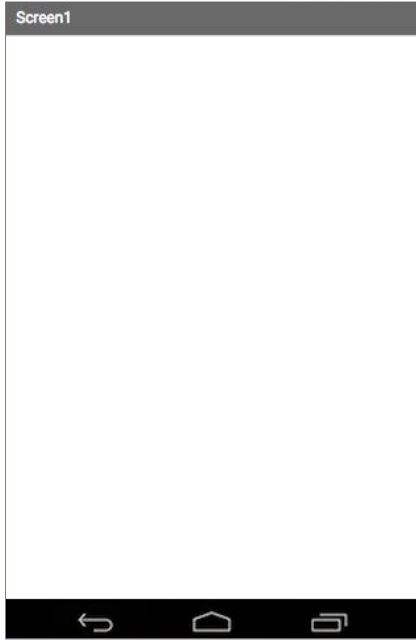
You will design and make an app that can be used in the ENVIRONMENT. This worksheet will help you think about what kind of app you would like to make.

1. Come up with at least 3 problems having to do with the ENVIRONMENT that you are *passionate* about.

Problem statement	Why I care about this problem

2. Share your problem statements with the person next to you. Discuss possible ideas for simple apps that would use CloudDB/data sharing to solve these issues. Take notes below:

3. On the following page, draw ideas for possible apps you could make that would use the CloudDB component somehow to solve problems. How would someone use your app?



C.6 CloudDB Group Design Project Worksheet

CloudDB Group Design Project Worksheet

Now that you've formed a group, your group needs to decide what you want to make! All group members can use this worksheet to write down ideas, but only one needs to be completed.

1. Take turns discussing the ideas that you came up with individually. When it's your partner's turn, write down specific things s/he talked about that interested you the most:

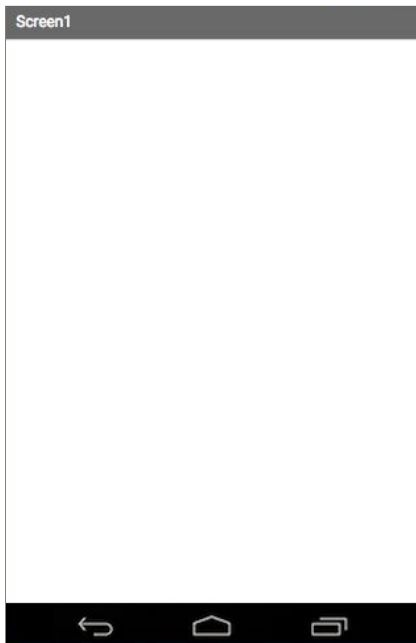
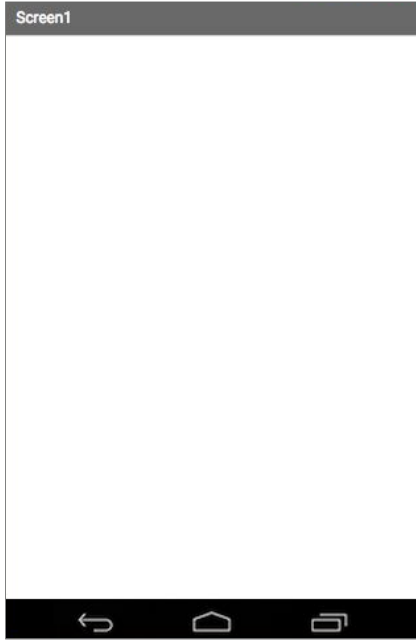
2. Decide which problem your group wants to tackle for this project. Talk about ideas for what app you want to make to solve the problem. Keep in mind that you have limited class time to code, so try to make sure that you can finish your app on time (with help from teachers)!

What is the problem you are trying to solve? _____

How will your app solve it/What will your app help people do? _____

How does your app use the CloudDB component? _____

3. On the back of this page, draw some ideas for what your final app will look like. Try to mark what buttons users would press to do something and how the app would respond to these user actions.



4. Now, let's focus on how you will use and code the CloudDB portion of your app. Think about what the CloudDB blocks you've worked with so far can do and how you can use them for your project. Fill out the following:



When storing data in CloudDB:

What my **tags** will look like: _____

Why/What they are used for: _____

What my **valuesToStore** will look like: _____

Why/What they are used for: _____

When getting data from CloudDB:

What my **tags** will look like: _____

Why/What they are used for: _____

What my **valuesToStore** will look like: _____

Why/What they are used for: _____

When data is changed in CloudDB:

My app will do _____ with the **tag** and

_____ with the **value**.

C.7 Design Project Peer Feedback Worksheet

Design - Peer Feedback Worksheet

First, tell your audience the purpose of your app and what it's supposed to do. Then, interview the other group following the 5 questions below and write down all the feedback you receive (even if you disagree with what they say):

1. According to your understanding, what environmental issue are we trying to solve and who is the target audience?

2. In your own words, what does the app that we proposed do?

3. What do you like about our app idea? How do you think it is effective in solving our problem?

4. How does the app plan to use CloudDB? Does that use case make sense?

5. What are some parts that you think may not work too well? What could we add or change to make it better?

After both teams are done with the interviews, go back to your original seats and discuss the feedback that you received with your partner. What changes will you make to your project design based on the feedback?

C.8 Post-questionnaire

Survey

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree
I use the Cloud in my everyday life					
I understand how the Cloud stores and shares data					
I feel comfortable making apps that use the Cloud					
I can think of ways that the Cloud can solve problems in my everyday life					
My understanding of the Cloud improved through these workshops					

Free response

How do you feel about the app you made?

What was hard about developing the app?

What were some things you didn't expect?

In your own words or with pictures, describe what you think the Cloud is and what it does.

Do you think the Cloud has an important role in today's society? Why or why not?

What are some problems you see with the Cloud?

If you were to use Cloud technology in the future to create something that solves a problem you see in the world what would it be?

Bibliography

- [1] Valerie Barr and Chris Stephenson. 2011. Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? In *ACM Inroads 2, 1* (February 2011), 48-54. DOI=<http://dx.doi.org/10.1145/1929887.1929905>

- [2] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. In *ACM Inroads 2, 1* (February 2011), 32-37. DOI=<http://dx.doi.org/10.1145/1929887.1929902>

- [3] Wing, Jeannette M. "Computational thinking and thinking about computing." In *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences* 366.1881 (2008): 3717-3725.

- [4] Brennan, K., and Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Annual American Educational Research Association Meeting*, Vancouver, BC, Canada.

- [5] MIT App Inventor. Massachusetts Institute of Technology, 2015. Web. 03 Aug. 2016. <<http://appinventor.mit.edu/explore/>>.

- [6] MIT App Inventor. "Experimental Components - App Inventor for Android." MIT App Inventor. Massachusetts Institute of Technology, n.d. Web. 03 Aug. 2016. <<http://ai2.appinventor.mit.edu/reference/components/experimental.html#FirebaseDB>>.

- [7] Sayamindu Dasgupta. 2013. From Surveys to Collaborative Art: Enabling Children to Program with Online Data. In Proceedings of the 12th International Conference on Interaction Design and Children (IDC '13). ACM, New York, NY, USA, 28-35. DOI: <http://dx.doi.org/10.1145/2485760.2485784>
- [8] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: programming for all. *Communications of the ACM*, 52(11):60-67, November 2009.
- [9] T. Stern. NetScratch: A networked programming environment for children. Master's thesis, Massachusetts Institute of Technology, 2007.
- [10] Bray, Tim. "Identifying App Installations." *Android Developers Blog*. N.p., 30 Mar. 2011. Web. 03 Aug. 2016. <<http://android-developers.blogspot.com/2011/03/identifying-app-installations.html>>.
- [11] Jeannette M. Wing. 2006. Computational thinking. In *Commun. ACM* 49, 3 (March 2006), 33-35. DOI=<http://dx.doi.org/10.1145/1118178.1118215>
- [12] Grover, S., and R. Pea. "Computational Thinking in K-12: A Review of the State of the Field." *Educational Researcher* 42.1 (2013): 38-43. Web.
- [13] Duncan, Caitlin, and Tim Bell. "A Pilot Computer Science and Programming Course for Primary School Students." *ACM Digital Library*. ACM, 11 Nov. 2015. Web. 19 Apr. 2017.
- [14] Grover, S. and Pea, R. 2013. Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students. In *Proceeding of the 44th ACM technical symposium on Computer science education (2013)*, 723-728.
- [15] Microsoft Corporation. "What Is Cloud Computing? A Beginner's Guide." *Microsoft Azure*. N.p., 2017. Web. 19 Apr. 2017. <<https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/>>.

- [16] Morgan, Timothy Prickett. "How Long Can AWS Keep Climbing Its Steep Growth Curve?" The Next Platform. N.p., 03 Feb. 2016. Web. 19 Apr. 2017. <<https://www.nextplatform.com/2016/02/01/how-long-can-aws-keep-climbing-its-steep-growth-curve/>>.
- [17] MIT Educational Studies Program. "HSSP." MIT ESP - HSSP. Massachusetts Institute of Technology, 26 Apr. 2017. Web. 03 May 2017. <<https://esp.mit.edu/learn/HSSP/index.html>>.
- [18] Bender, Matthias, Sebastian Michel, Sebastian Parkitny, and Gerhard Weikum. "A Comparative Study of Pub/Sub Methods in Structured P2P Networks." Databases, Information Systems, and Peer-to-Peer Computing Lecture Notes in Computer Science (n.d.): 385-96.
- [19] "Redis Pub/Sub." Redis.io. RedisLabs, n.d. Web. 10 May 2017. <<https://redis.io/topics/pubsub>>.
- [20] "Redis." Redis.io. RedisLabs, n.d. Web. 10 May 2017. <<https://redis.io/>>.
- [21] "Command Reference - Redis." Redis. RedisLabs, n.d. Web. 15 May 2017. <<https://redis.io/commands/>>.
- [22] "EVAL Script Numkeys Key [key ...] Arg [arg ...]." Redis. RedisLabs, n.d. Web. 15 May 2017. <<https://redis.io/commands/eval>>.
- [23] P. J. Courtois, F. Heymans, D. L. Parnas, "Concurrent control with 'readers' and 'writers'", Commun. Ass. Comput. Mach., vol. 14, pp. 667-668, Oct. 1971.
- [24] Baumann, Tim. "Operational Transformation." Operational Transformation - OT Explained. Github, n.d. Web. 14 May 2017. <<https://operational-transformation.github.io/>>.
- [25] "Elixir." Elixir-Lang. Plataformatec, n.d. 01 April 2017. <<http://elixir-lang.org>>

- [26] "Supervision Principles." Erlang. Ericsson, n.d. 01 April 2017. <http://erlang.org/documentation/doc-4.9.1/doc/design_principles/sup_princ.html>
- [27] Instagram, Inc. "Instagram on the App Store." App Store. Apple Inc., 15 May 2017. Web. 16 May 2017. <<https://itunes.apple.com/no/app/instagram/id389801252>>.
- [28] "Mutators." MIT App Inventor. Massachusetts Institute of Technology, n.d. Web. 20 May 2017. <<http://appinventor.mit.edu/explore/ai2/support/concepts/mutators.html>>.
- [29] Scratch. "Cloud Data: Cloud Data History." Scratch Wiki. MIT Media Lab, 23 Apr. 2017. Web. 21 May 2017. <https://wiki.scratch.mit.edu/wiki/Cloud_Data#Cloud_Data_History>.