Prison Garden Application

Final Report

Team Number:	may1713
Client:	Julie Stevens
Advisor:	Doug Jacobson
Leader:	Juan Venegas
Webmaster:	Andrew Dailey
Webmaster:	Jacob Stimes
Communications:	Matt Koenig
Concept Holder:	ChiaJun Tai
Concept Holder:	Jacob Anhorn
Team Email:	may1713@itstate.edu
Team Website:	https://may1713.sd.ece.iastate.edu
Revised:	23 April, 2017

Revised Project Design

Project Statement

The Prison Garden Application will be a tool for inmates to use in prisons across Iowa for educational and rehabitual purposes. It will provide inmates with skills they can use to re-incorporate themselves into society, while also giving them a sense of accomplishment. This web application will meet prison security standards regarding internet connectivity by being isolated from any external networks. Progress will be tracked by a database hosted either locally or on an adjacent system. Once an inmate makes satisfactory progress in the app then they will be rewarded a certificate for their progress and knowledge they have acquired.

Design

The Prison Garden Application is designed to be a web application for use on multiple tablet platforms. The tablet platform that we will use for the initial development is Android. This was chosen for a few reasons; Firstly, if other prisons wanted to use the tool they would not have to worry about exchanging the tablets they are using. Second, Android is an open source platform that was designed with development in mind, allowing for effective and intuitive application building.

Non-Function Requirements

- The tablet needed for the app shall be very durable as it may be dropped or even tampered with
- The OS on the tablet shall have its security options modified. This is so the the inmates will not be able to tamper with the app or tablet on a software level
- The app will have a high volume of content to educate users on gardening, which needs to be organized in a user-friendly manner
- The application shall be mobile friendly: specifically with tablets
- App shall support multiple user profiles, be able to handle different users signing into the app and sync their progress back to a database

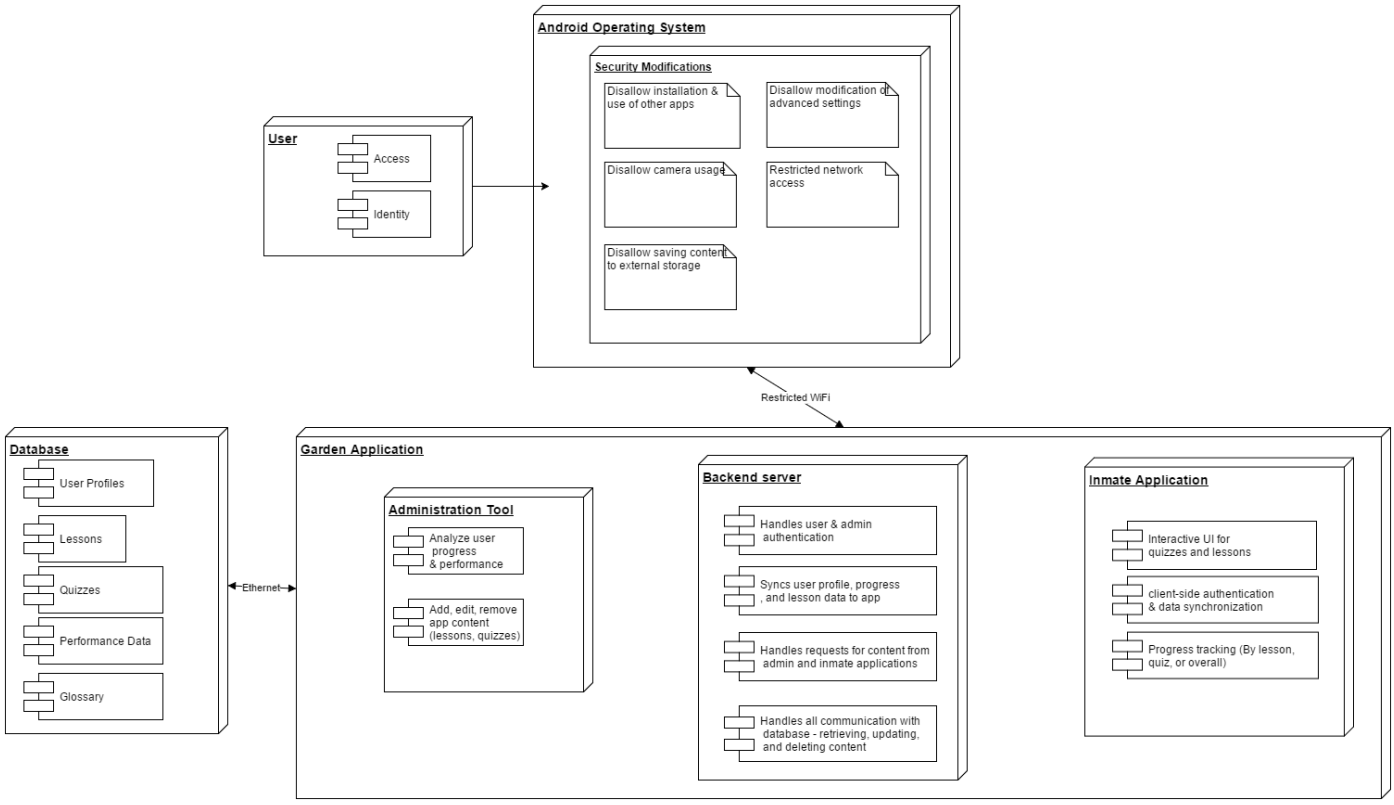
Functional

- The app shall prompt for credentials when it is opened. The app has to support multiple users and will need keep track of individual progress for each
- The app will display the progress/ the quiz scores that the user has taken on the home page once the user has logged in
- The app shall allow users to view lessons and instructions for furthering their progress within the prison's gardening program
- The app shall prompt the user for small objective quizzes after each educational course. After the user goes through the desired course, the user can take a quiz by

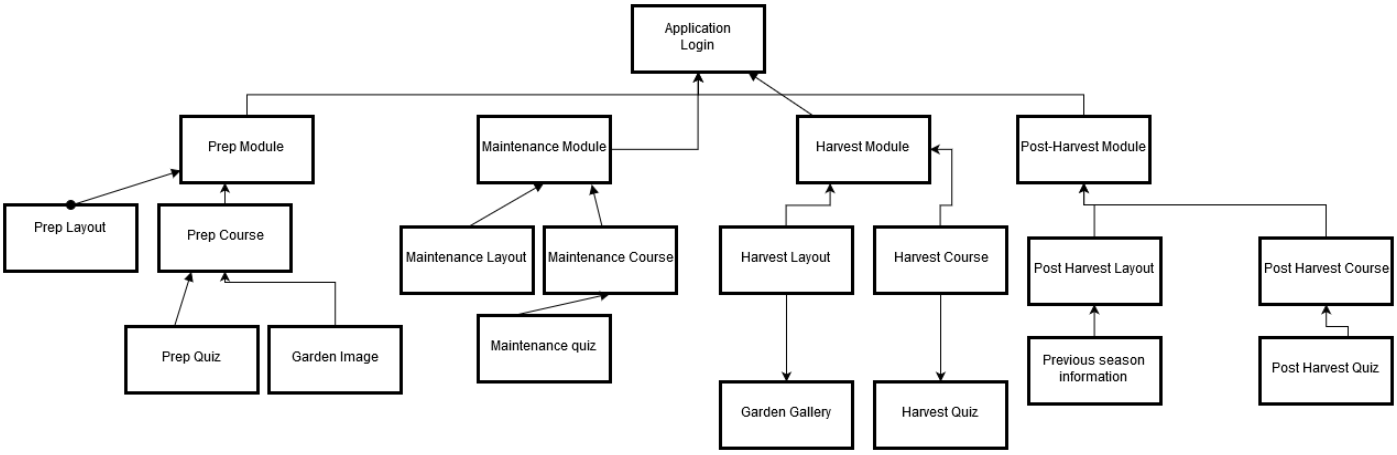
selecting the link/button. Once selected, the app will prompt the user with randomized multiple choice questions related to the course they took

- The app shall prompt the user if they are ready to log out. When the user clicks on the logout button, the app will log them out and notify them that they have been successfully logged out
- The OS shall restrict app access to only the gardening application created by our team
- The OS shall have a special, restricted profile to be used by the inmates. This profile will disallow the changing of settings and store no persistent data.
- The app shall allow admin users to upload future educational content or delete current content and the app will update the state of the app with the additional content or the removal of content
- The app shall allow admin users to view the progress of users and view and any content or information associated to users

Block Diagram



Functional Decomposition



Standards?

DANK Standards

Implementation Details

User Interface & Front End

The technologies we used in our front end and user interface implementation were Javascript, HTML5, Bootstrap, JQuery, and CSS. We set up templates for displaying the pages with dynamic data retrieved from our database, so admin users would be able to add, edit or remove content in the future without needing technical skills to create or remove the actual pages themselves.

The gardening content consists of a set of units, each of which contain a set of lessons. The app's navigation menu is built dynamically from the list of units in the database, and the lesson options on the unit pages are in turn built from the list of lessons for that unit. In addition, there is a quiz for each unit users can navigate to from the unit page or any lesson page. Quizzes consist of ten questions randomly selected from a master list of questions for the chosen unit, stored in the database. Each attempt by a user is stored with the permutation of questions and the answers that were selected. Admins can later review these attempts, as well as see statistics about how well users are doing on quizzes overall.

Also included in the menus are three static options: Glossary, Garden Planning and Records. The glossary contains a list of keywords along with their definitions. Garden Planning is where users can map out each of their gardens into square plots and plan what plant to place in each plot, and see previous plans. In the Records section, users can log their daily

activities, yearly yields and make comments about observations made in the gardens. Users submit forms containing the relevant information and data is publicly displayed for all users and admins to view. Daily records can not be edited or deleted once submitted to the database and are tagged with the user's ID and the given date. Previous records from all users are displayed on the main page of the records section, sorted descending by entered date.

Standard users only have access to the gardening application, whereas admins also have access to the admin tool, a second application hosted on the same server. This tool is accessible through a web browser and is where admins will be able to see user progress data and manage content. They have the ability to add, edit, and remove units, lessons, quizzes and supplemental content such as PDF files and videos.

We created a set of uploading guidelines, specifying how content should be organized for uploading. For example, we decided upon a specific directory structure we expect units and lessons to be in (described in detail in Appendix I), and we have a procedure for generating correct hyperlinks from an uploaded html file (Appendix IV). The team creating the gardening content uses a software called Adobe InDesign, which can export to HTML. However, since they are not aware of where resources reside on our server, we came up with specifications for them to designate where links should go and what they link to, and our uploading takes care of replacing those comments with the correct hyperlinks.

Database & Server

For our server we utilized Django and its built in user authentication. We also added a few python classes ourselves such as a content manager for adding, editing and removing content, and a data access class for interfacing with our data base. The server takes requests from the front end pages and translates them into requests for our database. Content from the lessons and supplemental content is stored on our server in a file system, so we don't need to worry about converting the files to and from a binary format to store in the database.

The database is implemented in MySQL, and stores information about users, units, lessons, quizzes, garden plans, daily work logs and various details tracked for us by Django like user authentication and permissions. The data access class is solely responsible for opening and closing connections to the database, and defines the format for how data is passed to and from the database. Tables that have information connected to other tables, like lessons contained in specific units, are linked via foreign keys and make use of cascade triggers, from delete and update events, to ensure the information stored is up to date.

Testing Process & Results

Frontend

The project frontend is broken up into a few main parts, all with different expected user demographics. Half of it is comprised of administrative tooling and progress tracking while the other side is a mobile-friendly interface to be used by the prison's inmates. Therefore, the client-facing screens had an implicit requirement to be more intuitive and simple to use. The admin tools, on the other hand, were more complicated in terms of their capabilities which led to a design that was more focused on practicality than aesthetics.

Frontend testing was comprised almost entirely of user experience feedback. All of the project's functional logic is implemented server-side, which would have rendered frontend logic testing somewhat redundant. The more applicable circumstance for testing the administrative and client pages was in-person meetings with Julie and other members of her design team. In our opinion, the most valuable evaluation we received came in the form of direct verbal comments and critiques from those who have substantial experience with design concepts.

We met with the design team as often as our schedules would allow (typically weekly). Most meetings followed a standard agenda: updates, demo, and feedback. Our team would update theirs on what developmental progress had been made the previous week. Details explained included new features, bug fixes, and future plans. We then performed a small showcase of these aforementioned changes. Lastly, Julie's team conveyed concerns and recommendations with respect to how features looked and operated. From this feedback, our team amended our tasks for the upcoming week. Any functional or visual defects in the project's frontend came to be ironed out through this weekly process.

Backend

The project server (written in Python with help from Django) contains all of the business logic for our application. Events such as user authentication, lesson retrieval, and quiz submissions are all initiated by REST calls and handled by our backend code. Fortunately, this created a separation of concerns between user interfaces (views) and application functionality (models and controllers).

From the start, we designed our code to be testable by adhering to two primary ideas: modularity and leveraging Django's pre-packaged features. Python allows for the development of code modules with well-defined borders. By separating our features into individually-packaged and independent segments, testing became an already divided and conquered procedure. The scope for unit testing any particular part of the application became clear and intuitive: does this module successfully and accurately perform its single responsibility? By writing small and concise classes and modules, the overall project complexity was able to remain stamped down.

Moving forward from the unit testing phase, we began assembling these logical modules into a cohesive collection. Testing this assembly came in the form of use case definitions and executions. We asked questions similar to:

- How does an inmate login?
- How does an inmate select a lesson?
- How does an inmate take a quiz?
- How does an administrator add new course content?
- How does an administrator review inmate progress?

For each question like this, we discussed the steps and checked how natural their actual process was. Any use case that failed to satisfy the acceptance criteria of both our team and the design team was sent back to the task board for revisions. It is from this process that we kept the application's usability and correctness moving in a direction that was ideal for all invested parties.

Appendix I - Operational Manual

There are four main components to the operation of our app. The first is running a server to host our content. The second is installing a client-side version of our app on a tablet. Third is the utilization of the admin tool. The final component is the utilization of the app from the non-admin user's perspective.

Server setup

Since this is a web-based app, running the project requires a server. Client devices such as tablets send requests to the server to retrieve webpages and data. For development and testing, we used a local-server that can easily be run on our own devices. For deployment, we have been given a server from the Corrections Facility to use for hosting the app.

Prerequisites

Regardless of whether the app is to be run locally or on a production server, these steps must be completed on the machine that is to be used for a server before it can run our application.

1. Retrieve our codebase from this git repository (download as zip or use 'git clone' in a git-terminal):
   https://github.com/obijuankenobi/Prison-Garden-Application.git

2. Install python 2.X
   https://www.python.org/downloads/

3. Install virtual environment by entering this command:
   ```
   pip install virtualenv
   ```

For reference:
http://python-guide-pt-br.readthedocs.io/en/latest/dev/virtualenvs/

4. Create and activate a virtual environment
   ```
   virtualenv venv
   ```

5. With the virtual environment activated, enter these commands:
   ```
   a. pip install django
   b. pip install openpyxl
   c. pip install mysql-python
   ```

Running locally

1. In a terminal, navigate to the directory containing the virtual environment and the application code

2. Activate the virtual environment by entering this command:
   ```
   workon venv
   ```

3. Start the server by entering this command:
   ```
   python manage.py runserver
   ```

4. In a web browser, navigate to:
   localhost:8000

This shows the homepage of our application, and the entire app can be used, navigated, and tested.

Deploying to a server

1. On the server, navigate to the 'server' directory of our code and run the 'setup.sh' script with this command:
   ```
   ./setup.sh
   ```

2. In a terminal, navigate to the directory containing the virtual environment and our code

3. Activate the virtual environment by entering this command:
   ```
   workon venv
   ```

4. Start hosting and listening for requests by entering this command:
   ```
   python manage.py runserver 0.0.0.0:8000
   ```

Now, devices with a connection to that server can make requests for our application. The server will require access points to be configured at the locations where users are expected to be using the tablets.

App-installation on Tablets

We have a small Android app that is to be installed on devices used for this application. This is because our app is intended to be used on tablets, but we can not allow the users to directly access a web-browser. Contained in our codebase is an APK file, which is an Android app installer.

Installation

1. Retrieve our codebase from this git repository (download as zip or use ‘git clone’ in a git-terminal):
   https://github.com/obijuankenobi/Prison-Garden-Application.git

2. Locate the APK file under the ‘client’ directory and move the file to the Android device (either by downloading it or by transferring it via USB)

3. On the Android device, go to settings, then 'Security', and turn on 'Unknown sources'

4. On the Android device, find the directory where you downloaded or copied the APK file, and tap the file

5. Follow the on-screen installation instructions

Upon completion, there should be a new app icon on the device, which launches our app once tapped.

Admin Tool

Creating Admin User

1. In a terminal, navigate to the directory containing the virtual environment and the application code

2. Enter the following command to create an admin account:
   ```
   python manage.py createsuperuser
   ```

3. Fill out the information as prompted and once successful the terminal will display a message confirming the success

4. Admins can also use the application normally like a user but also have access to the Django admin system and the Admin Tool

Using Django admin

1. Make sure the server is running
2. On a computer open your browser and type (ip address of server)/admin
3. Check with your your local IT admin for the ip address of the server

4. Login in using the credentials made in the step above. This is the Django Admin home page where admins can reset the password of users or can restrict/disable a user if needed
5. Clicking on users will show all user that have created an account
6. Clicking on a specific user will display their information

Using PGA Admin tool

1. The application-specific admin tool is located at: (ip address of server)/pgaadmin
2. If the server is running, you can log into the tool by entering the credentials created in 'Creating Admin User'
3. On the home page, there are two options
    a. User progress
        i. Gives access to a list of users and the courses they've completed
        ii. Can also see a list of quizzes, and selecting a quiz allows one to view specific statistics on what answers have been given
    b. Course management
        i. Initially presents a list of units that can be renamed, deleted and clicked on
        ii. Clicking a unit navigates to a page listing the lessons in the unit as well as supplementary materials used in that unit. These can similarly be edited and deleted. Each unit has a link to its accompanying quiz
        iii. On the quiz page, quiz questions & answers can be edited, questions can be added and deleted
        iv. On each of these pages, there is an upload button which allows the admin to select a file to upload to that section. The uploading specifications are described next.

Uploading Specifications

New units (aka courses) can be uploaded to the application as a zipped directory containing all of its necessary content (individual lessons, supplementary materials, and a quiz). The directory structure required for this process is predefined in order to be consistent. To start out, an example unit is displayed below.

- 5_Harvesting_Plants/
    - Picking_Carrots/
        - carrots.html
    - Picking_Tomatoes/
        - tomatoes.html
    - Washing_Plants/
        - plants.html
    - materials/
        - carrot.jpg
        - tomatoes.jpg
        - picking.mp4
        - washing.pdf
    - quiz/
        - quiz.xlsx

This sample is comprised of the following details:

- The unit is named “Harvesting Plants”
- It the 5th unit in the gardening program
- It has 3 lessons:
    - “Picking Carrots”
    - “Picking Tomatoes”
    - “Washing Plants”

The unit structure has a few guidelines that must be adhered to:

- All spaces must be represented as underscores (they will appear as spaces when live)
- Unit and lesson names are defined by their directory names
- The number before the unit directory name corresponds to its order in the program
- Individual lesson HTML files must end with “.html” (only 1 is expected per lesson)
- The unit’s quiz must be placed into a directory named “quiz”
- The unit’s quiz must end with “.xlsx”
- For each quiz question, there should be one row, where the first cell contains the question text, the next cell contains the correct answer, and all following cells are additional answer options (Quizzes are multiple choice format only)
- Additional materials must be placed into a directory named “materials”

Disabling/Deleting a user

1. Log into the Django Admin System
2. Click on users and find the user that you want to disable/delete
3. Once the user is selected, there will be a prompt at the bottom of the screen that will give you the option of deleting or disabling the user
    a. NOTE: that some of the users info such as entries in Records may still be viewable

Users

Users can create their own accounts. In order to do so they need to click on the register link below the submit button on the login page. They fill out a form and once filled out correctly they will automatically logged in and redirected to the user home page. Users will have access to all lessons and aren’t restricted to any particular lesson. Some lessons may be “under construction”, but there are tools the user can utilize.

Users can plan out what their garden will look like by accessing the garden image tool in the Preparation section. Here users can drag and drop vegetables into a grid so they can visualize what their garden can look like. Also, they have the option of saving their images for future use. Users can also log their activities as well as any observations they had on their gardens. They can do this by navigating to the Records course and selecting Daily Logs. Once selected, users will be able to see a table of logs made by other users ordered from most to least recent. Users cannot delete logs once they have been created. In order to add an entry to the table, users will fill out a form, supply the necessary information, and submit the form.

When learning a lesson, users will have the option to take a quiz to show what they have learned. The quizzes consist of 10 random multiple choice questions that pertain to what they have learned. Users simply click on what they think is the best answer and select submit once finished. Lastly, it should be noted that users need to log themselves out. The app does keep track of who was last logged in and as such they need to log out by selecting the logout tab in the navigation menu. The app will notify the user if they have successfully logged out and can log back in by selecting log in in the navigation menu.

Appendix II - Ideas and Roadblocks

In the initial version of the project we were considering making an application built in Android. We came to this decision based on the nature of the application, being that this application would be used on a tablet and an Android device would be more readily available to us. Upon consoluation with the IT administrator at the prison it sounded as if an Android exclusive application might not suit all their needs, as they liked the idea of cross-compatibility.

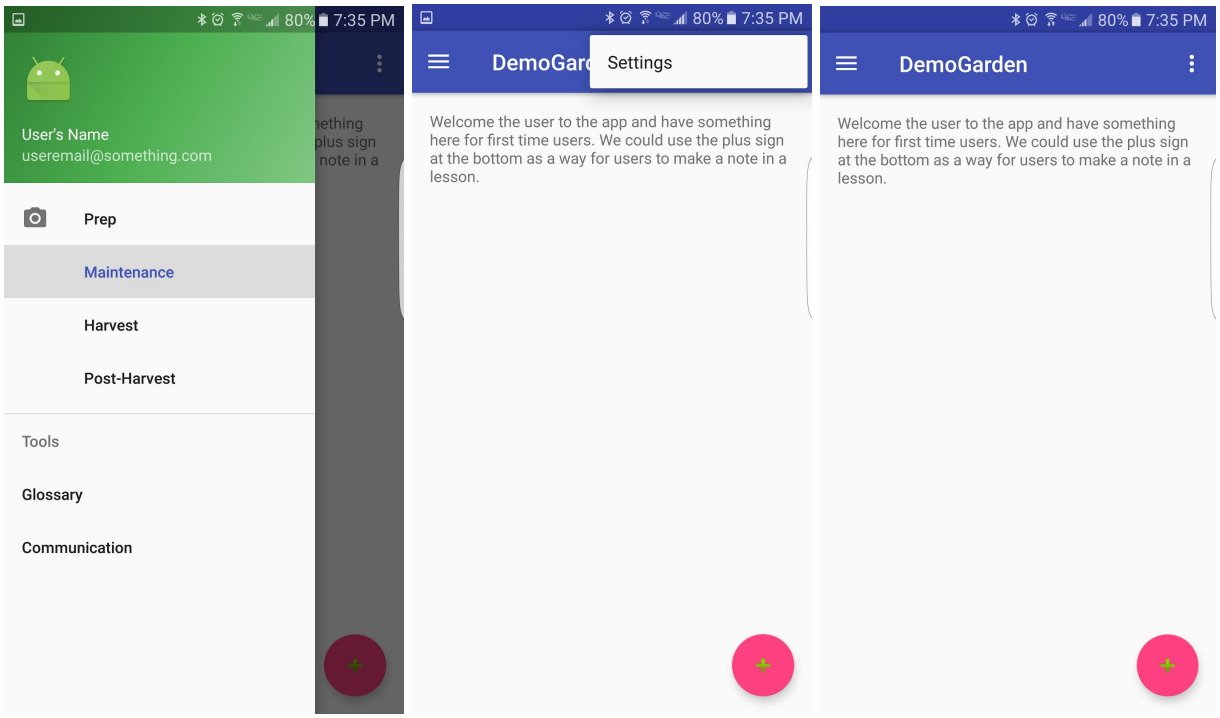
When we discovered this we began revising our design to suit the need for cross-compatibility. Our first idea was to potentially use a tool like Xamarin to stick with the idea of an Android/iOS centric application. As we began to evaluate this design further we began to think about moving away from the Android/iOS application and move toward the design we chose, a web-based application utilizing the Django framework .

The reason we chose to move towards a web application using Django was due to the built in administrator tool, as well as many of the team members having previous Python experience. One final design revision that needed to be done was to serve the requirement that users are restricted to the use of the application and the application alone. In order to serve this requirement we created an Android application that acts as a webview that exclusively points to our application.

We were going to implement a feature where admin users could be emailed by the application when a user has failed a quiz for the 3rd time. But since we are running this on an isolated server with no internet access, we will not be able to connect to gmail to send the email.

Julie and her team also requested the ability to use the camera to take pictures of plants and work being done, but we were unable to get our app to access the camera and not allow on-device storage.

Older App Ideas



Appendix III - Supplemental Content

Lessons Learned:

Since we were building the application by using our laptops to simulate the users experiences on the tablets, we did not frequently test the application until very few weeks before the final presentation. We faced some issues of mapping the touch events to the mouse events on our tablet. It caused failures of some of the features that we had implemented and postponed the timeline of other plans as we need to find other alternative solutions to solve the problems. So, we should always start early to test our application on the actual device to reduce the time and cost spent on debugging the application.

Appendix IV - Link Guidelines

The team creating the gardening content uses a software called Adobe InDesign, which can export to HTML. So we allow them to upload HTML files for their lessons. However, since they are not aware of where resources reside on our server, we came up with specifications for them to designate where links should go and what they link to, and our uploading takes

care of replacing those comments with the correct hyperlinks. These are the link specifications:

For each link you would like on a page, insert a line like this where you want the link:
<!--LINK: <filename>, type: <type>, text: <text>-->

<filename>, <type>, and <text> should be replaced as follows
(and inside of double quotes):

<filename> - This is the name of the resource you are linking to. For example, if you are linking to a video named "gardening.mp4", that's what you would put here. If it's a file (video, pdf, etc.) make sure to include the file extension (.mp4 or .pdf). If you are linking to another lesson, just type the lesson name itself. On the admin site, you will be able to view all resources and lessons that are uploaded, so you can check the names from there.

<type> - This specifies the type of resource you are linking to. Possible values are "video", "Pdf", "lesson", or"subsection". More details on subsections below.

<text> - This is simply the text you'd like to be shown for the link. For example, for that gardening video example, the text would probably be something like "Click here to watch a video".

A full example:

<!--LINK:"gardening.mp4", type: "video", text: "Click here to watch a video"-->

For linking to subsections (to navigate to a different part of the same page):

Use 'subsection' for the linktype.
To designate where the subsection starts, insert this line immediately before the subsection:

Replace"subsection_id" with whatever you'd like to use to identify this subsection. Use the exact same id for the <filename> part of the link. So for example a page with subsections may look like this:

<!--LINK:"section1", type:"subsection", name:"Go to section 1"-->
<!--LINK:"section2",type:"subsection", name:"Go to section 2"-->
...

...(section1 content)...

...(section2 content)...