CSC 400
Software Requirements Specification and Design Document
MoRPi: A Mobile Application for the Raspberry Pi
By Eric Barbin

1. Introduction

MoRPi (Mobile Raspberry Pi) is a mobile application that allows users to control a Raspberry Pi over Wifi. It aims to simplify the process of developing Raspberry Pi projects while simultaneously serving as a learning tool for beginners. Through the use of this application, users are able to create, customize, and run Raspberry Pi projects directly from a mobile device without having to connect peripheral devices such as a monitor, keyboard, or mouse, to the Pi. MoRPi also offers a highly interactive interface which resembles the Pi breadboard and allows users to intuitively develop a graphical representation of a project configuration that exists on the physical breadboard of the Pi. Users are able to modify the functionality of any connected hardware component either through a Graphical User Interface (GUI) or by editing the Python project source code directly. Project configurations and modified project source code may be saved to the Raspberry Pi or to the public GitHub repository dedicated to Raspberry Pi projects.

In comparison with existing Raspberry Pi mobile applications, MoRPi offers the most features, is the most interactive, and is the most user friendly. Many top rated Raspberry Pi applications currently available from Google Play provide little more than static project diagrams and tutorials on how to use the Raspberry Pi. From a functional perspective, the RaspController application is the most similar to MoRPi as it allows the user to control the GPIO pins, access the Pi file system, and send shell commands to the Pi. However, the GPIO pins are unintuitively represented in a list, and hardware support is limited to just four sensors and the Pi camera. MoRPi has an advantage over RaspController as it allows users to dynamically manipulate wiring and hardware configurations, control the GPIO pins on an intuitive interface, modify project source code, and save projects to the Raspberry Pi or GitHub.

Potential users of the MoRPi mobile application are primarily Raspberry Pi users. They will benefit from this application as it allows them to quickly and easily create, or modify, Pi projects without the need for connecting peripheral devices other than the breadboard. This is especially useful if the Pi is operating inside of a larger structure and the USB ports are inaccessible. The ability to save project configurations is also beneficial to users as it allows them to reuse the Pi for multiple purposes without ever losing project configurations. Saving project configurations to GitHub also benefits the entire Raspberry Pi community as users can share their projects with others. Beginners, especially those unfamiliar with Python, will also benefit from this application as it allows them to immediately build and run Pi projects while learning Python at their own pace.

2. Architecture

MoRPi is an Android application meant to run on mobile devices such as smartphones and tablets. The architecture of this system is based on two distinct client-server relationships where the Raspberry Pi may act as a client or a server depending on the component with which it is interacting. The first client-server relationship is between the mobile device and the Raspberry Pi, where the mobile device is the client and the Pi is the server. The second is between the Pi and GitHub, where the Pi is the client and GitHub is the server.

Terminology: Throughout this document, the term *project configuration* refers to the graphical representation of the wiring and hardware configurations displayed on the physical breadboard of the Raspberry Pi. The term *project configuration files* explicitly refers to all files which contain the data needed to display the project configuration on the application interface. The term *project template* refers to a folder containing predefined project configuration files, pre-written project source code, and hardware support modules. A *blank project template* is a project template which contains project configuration files with no configuration data. The term *user project* refers to a folder containing project configuration files and project source code where at least one of the files has been modified by the user, and hardware support modules. A user project may also contain collected data.

Major Inputs and Outputs of the System
The major inputs and outputs of the system are described in the following section and illustrated in Figure 1.

Inputs to the Mobile Device

1. **From the User:** Inputs to the mobile device from the user include textual input when logging in or editing Python Source code, and touch when selecting menu options or customizing a project configuration.
2. **From the Pi:** Inputs to the mobile device from the Raspberry Pi include project configuration data and Python source code.

Inputs to the Raspberry Pi

1. **From Mobile Device:** Inputs to the Raspberry Pi from the mobile device include user login credentials, project configuration data, and Python source code.
2. **From GitHub:** Inputs to the Pi from GitHub include project templates and user projects.

Outputs of the Mobile Device Displayed to the User

1. **Mobile to User:** Outputs of the mobile device shown to the user include the project configuration display and the Python source code for editing.

Outputs of the Raspberry Pi to GitHub

1. **Pi to GitHub:** Outputs of the Pi to GitHub are user login credentials and user projects.

Outputs of the Raspberry Pi to the Pi Breadboard

1. **Pi to Breadboard:** Outputs of the Pi to the breadboard are the electrical signals which control the hardware components.

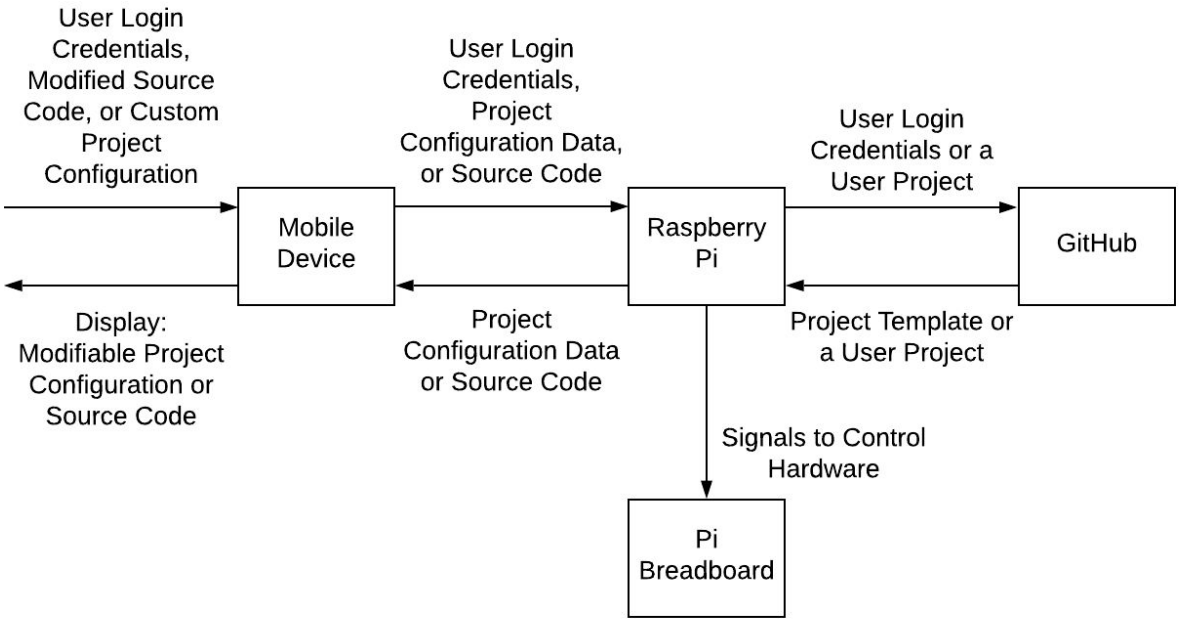

Figure 1: Data Flow of Major Inputs and Outputs

Description of User Interfaces

All user interfaces exist as Android Activities or Dialogs within the MoRPi mobile application.

1. **Breadboard Interface**

   The breadboard interface is the main interface of the application and is an Android Activity designed to replicate the breadboard of the Raspberry Pi. It contains functional representations of the GPIO pins and breadboard pinholes. Users may change the state of a pin or modify the project configuration by touching a pin, or pinhole, respectively.

2. **Wiring Configuration Interface**
   The wiring configuration interface is an Android Dialog which presents the user with a list of wiring configuration options such as selecting wire color, removing one end of a wire, or removing an entire wire.

3. **Hardware Selector Interface**
   The hardware selector interface is an Android Dialog which contains a list of all available hardware components a user may add to a project. Selecting a hardware component from this list adds it to the project configuration on the breadboard interface.

4. **Hardware Functionality Modifier Interface**
   The hardware functionality modifier interface is an Android Dialog that displays a list of predefined functional modification options for a selected hardware component. To open this interface, the user can simply touch a hardware component which has been added to the breadboard interface.

5. **Source Code Editor Interface**
   The source code editor interface is an Android Activity that displays the Python source code to the user and allows them to make modifications. Advanced users can open this interface from the main menu and freely edit the entire project source code. Beginners can open it through the hardware functionality modifier interface where they can also view helpful suggestions on how to modify component functionality in Python source code.

6. **Project Selector Interface**
   When a user opts to open a project, they are presented with the project selector interface which is an Android Activity that allows users to view a list of projects residing in the Raspberry Pi file system, or on GitHub.

7. **Login Interfaces**
   The Raspberry Pi and GitHub login interfaces are both Android Dialogs which accept the text input of the user's Pi and GitHub login credentials, respectively.

8. **Save Project Interface**
   The Save Project interface is an Android Dialog which accepts text input so the user may name the project.

MoRPi Directory Hierarchy

The MoRPi main application directory is created under the Pi's home directory the first time the user connects to the Raspberry Pi through the application. Subdirectories for project templates and user projects are also created in this directory. Initially, the project templates subdirectory is filled with a blank project template and three basic project templates. All other project templates pulled from GitHub are also stored here. User projects are added to the user projects subdirectory when a user creates a new project, or pulls an existing user project from GitHub.

3. Use Cases

All use cases assume that the Raspberry Pi intended for use with this application has already been configured for remote access over a Wireless Local Area Network and has a static IP address. While this initial setup is critical for application use, it is not a considered a main interaction directly between the user and the application and as such, the steps for this configuration have been omitted from this document. For further information on Raspberry Pi configuration, users are referred to official Raspberry Pi documentation.

Use Case Diagrams

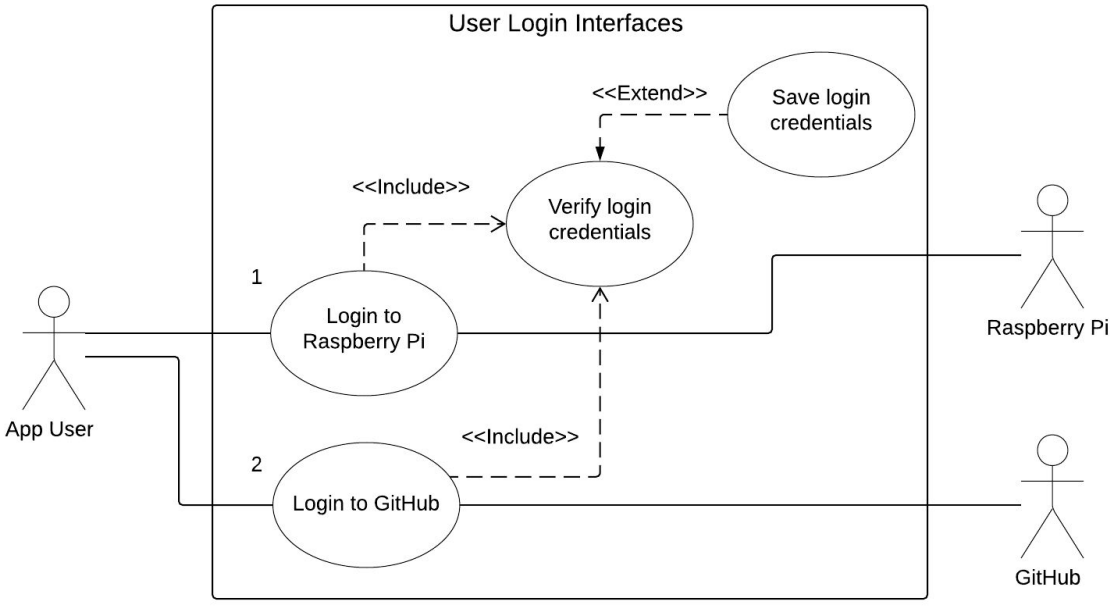

Figure 2: Use case diagram for user login

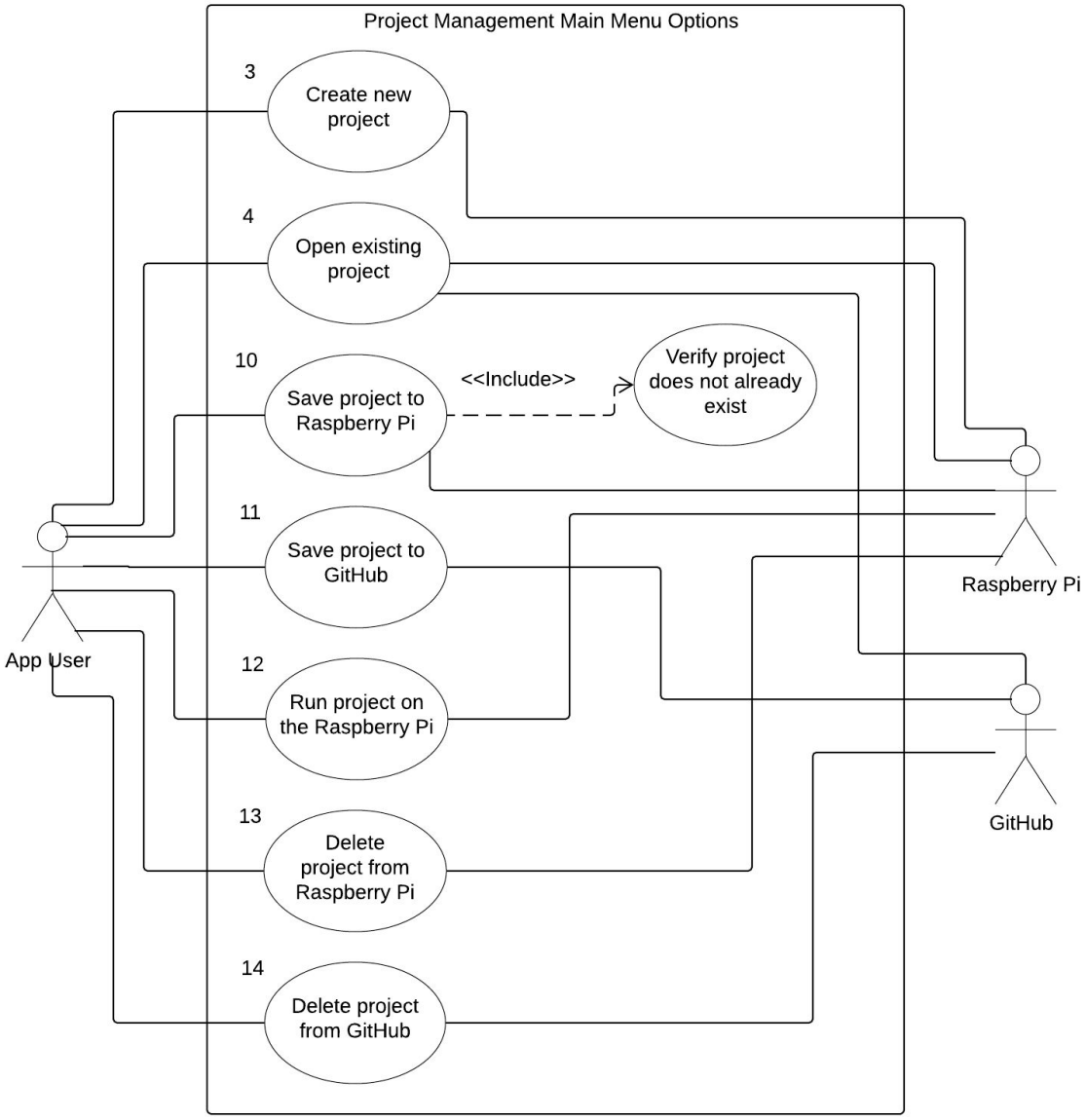

Figure 3: Use case diagram for project management

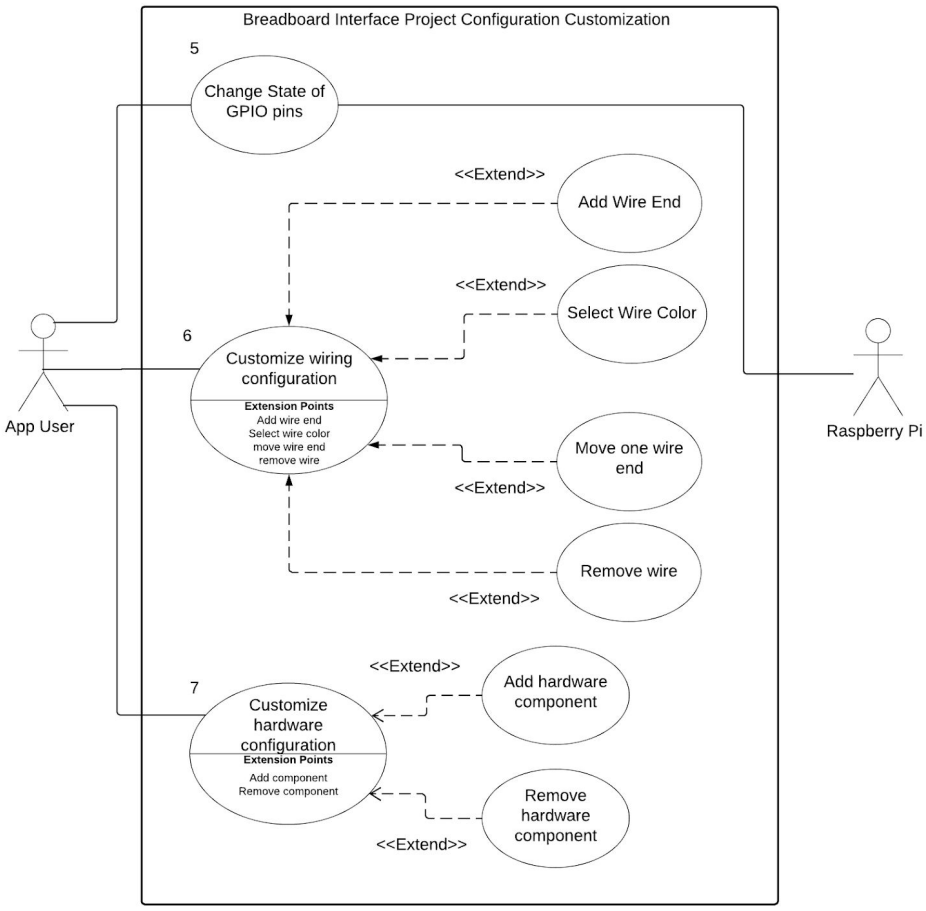

Figure 4: Use case diagram for custom project configuration

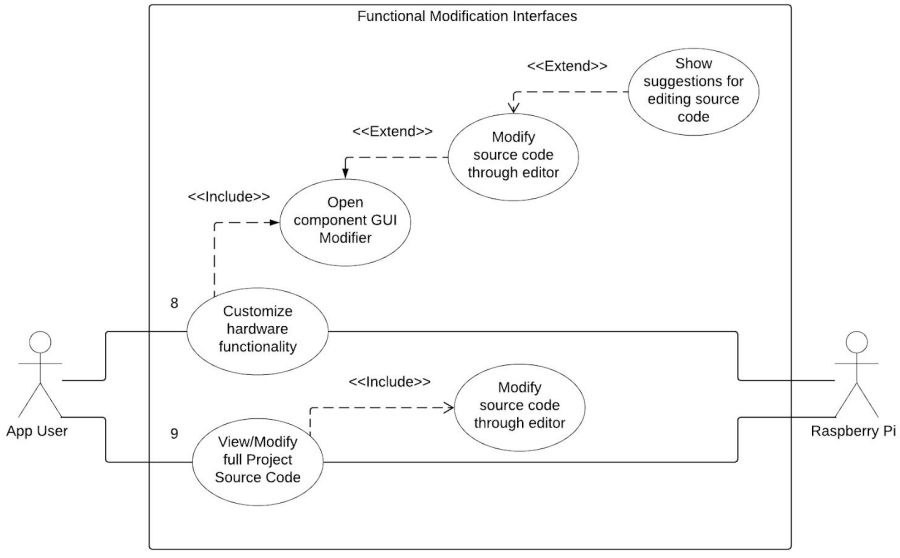

Figure 5: Use case diagram for functional modifications

Use Cases and Scenarios

1) Login to Raspberry Pi through mobile device.

The user is required to login to the Pi to open, save, or run a project.

Steps common to all scenarios:

1. The user obtains the static IP address of the Pi.
2. The user selects the option to login to the Pi.
3. The user enters their Pi login credentials and the IP address of the Pi.

Scenario 1: Successful login.

4. The system notifies the user that they are logged into the Pi.
5. The system notifies the user that their login credentials and the Pi IP address may be stored for future use.

Alternative 1: The user opts to save the login information.

1. The information is saved to the mobile device.

Alternative 2: The user opts not to save the login information.

1. The information is not saved.

Scenario 2: Unsuccessful login attempt.

4. The system notifies the user that a connection has not been established.
5. The system provides the user with troubleshooting tips such as ensuring that the correct credentials and IP address were entered and ensuring that the Pi is correctly configured for remote access.

Scenario 3: First login through the MoRPi application.

4. Successful as in scenario 1. (Assume)
5. The system creates the MoRPi directory and subdirectories, and pulls project templates from the public GitHub repository for Raspberry Pi projects.

2) Login to GitHub.

A user is only required to login to GitHub if they want to push their project to GitHub. The user is not required to login to GitHub to open a project from GitHub.

Steps common to all scenarios:

1. The user selects the option to login to their GitHub account.
2. The user enters their GitHub login credentials.

Scenario 1: Successful Login.

3. The system notifies the user that they are logged into GitHub.
4. The system notifies the user that their login credentials may be stored for future use.

Alternative 1: The user opts to save the login information.

5. The information is saved to the mobile device.

Alternative 2: The user opts not to save the login information.

5. The information is not saved.

Scenario 2: Unsuccessful Login attempt.

3. The system notifies the user that the login was unsuccessful and prompts them to re-enter their login credentials.

3) Create a new Raspberry Pi project.

By default, the user can create a new project with a custom configuration immediately upon opening the application.

Scenario 1: Create a new project configuration.

1. The user opens the application or selects the new project option from the main menu.
2. The default blank project template is opened.
3. The main application interface shows a graphical representation of an empty, Raspberry Pi, breadboard.
4. The user can create a new project configuration by adding graphical representations of Raspberry Pi wires and hardware components to the breadboard, and by turning Raspberry Pi GPIO pins on or off.

4) Open existing Raspberry Pi project.

The steps outlined in this use case assume that the user is logged into the Raspberry Pi as specified in use case 1.

Steps common to all scenarios:

1. The user selects the option to open an existing project from the main menu.
2. The system prompts the user to open a project from the Raspberry Pi, or from GitHub.
3. The user selects the source from which to open the project.
4. In either case, the system prompts the user to select a project template or a user project.
5. The user selects the type of project (template or user) to open.
6. In either case, the system shows the user a list of available projects of the selected type, which reside on the selected source.
7. The user selects a specific project to open.

Scenario 1: Successfully open an existing project configuration stored on the Raspberry Pi.

8. The system loads project configuration data and the project configuration is displayed on the main breadboard interface.

Scenario 2: Successfully open an existing project from GitHub.

8. The system checks to see if the project is already stored on the Pi.

Alternative 1: The project is saved on the Pi.

1. The system prompts the user to either open the project from the Pi or to download the project from GitHub and overwrite the existing project.

Alternative 1 A: The user opts to open the project from the pi.

a. The project is then loaded as in scenario 1.

Alternative 1B: The user opts to open the project from the Github and overwrite the existing project.

a. The project is pulled from GitHub and overwrites the existing project on the Raspberry Pi.
b. The project configuration is then loaded as in scenario 1.

Alternative 2: The project is not saved on the Pi.

1. The project is pulled from GitHub and stored on the Raspberry Pi.
2. The project configuration is then loaded as in scenario 1.

Scenario 3: Unsuccessful attempt at opening an existing project stored on the Raspberry Pi.

8. The system notifies the user that an error occurred while trying to open the selected project and recommends that the user open the project from GitHub, if a repository for that project exists.

Scenario 4: Unsuccessful attempt at opening a project from GitHub.

8. The system notifies the user that the project repository may have been moved or is no longer available.

5) Change state of GPIO pins.

Scenario 1: Change state of pin.

1. The user touches a graphical representation of a GPIO pin on the application breadboard interface.

Alternative 1: The pin is off.

1. The corresponding, physical pin on the Raspberry Pi is turned on.

Alternative 2: The pin is on.

1. The corresponding, physical pin on the Raspberry Pi is turned off.

6) Customize wiring configuration on the application breadboard interface.

Scenario 1: Connect one end of a new wire to an empty, breadboard pinhole.

1. The user touches a pinhole on the application breadboard interface for which there is no associated wire color or wire number.
2. The system prompts the user to select a wire color.
3. The user selects the wire color.
4. A graphical representation of one end of a wire is displayed on the pinhole, in the selected color.

Scenario 2: Connect the second end of a wire to an empty, breadboard pinhole.
(Assume step 1 or step 4 has taken place.)

1. The user touches a pinhole on the application breadboard interface for which there is no associated wire color or wire number.
2. A graphical representation of a wire, of the selected color, is drawn between the two pinholes.

Scenario 3: Changing the color of a wire

1. The user touches a graphical representation of a pinhole on the application breadboard interface for which there is an associated wire number or graphical representation of a wire.
2. The system displays a list of wiring configuration options.
3. The user opts to change the color of the wire.
4. The wire is displayed in the newly selected color.

Scenario 4: Changing the position of one end of a wire.

1. The user touches a pinhole on the application breadboard interface for which there is a wire number, or graphical representation of a wire.
2. The system displays a list of wiring configuration options.
3. The user opts to remove the selected end of the wire.
4. The graphical representation of the wire is removed.
5. The user proceeds as in Scenario 2.

Scenario 5: Removing a wire.

1. The user touches a graphical representation of a pinhole on the application breadboard interface for which there is an associated wire number or graphical representation of a wire.
2. The system displays a list of wiring configuration options.
3. The user opts to remove the wire.
4. The pinholes are reset to default color with no numbering, the graphical representation of the wire is removed, and all wire numbers corresponding to wires of the same color are adjusted based on current wire count.

7) Customize hardware configuration on the application breadboard interface.

Scenario 1: Add a new piece of hardware:

1. The user selects the option to add a new piece of hardware to the project configuration.
2. The system presents the user with a list of available hardware components.
3. The user selects a hardware component.
4. The system places the hardware in a predefined location on the breadboard interface.

Scenario 2: Change the location of an existing piece of hardware:

1. The touches the hardware component and is presented with a list of configuration options.
2. The user opts to change the location of the hardware component.
3. The user touches a new location on the screen.
4. The hardware component is placed at the new location.

Scenario 3: Remove an existing piece of hardware:

1. The user touches a hardware component.
2. The system presents the user with a list of hardware configuration options.
3. The user selects the option to remove a hardware component.
4. The hardware component is removed from the interface along with any connected wires.

8) Customize hardware functionality.

Scenario 1:

1. The user touches a hardware component.
2. The system presents the user with a list of hardware configuration options.
3. The user opts to edit component functionality.
4. The system displays a list of functional modification options, specific to the physical capability of the component.
5. The user selects a modification option based on the functionality they want to change.
6. The system presents the user with a GUI (an Android Dialog) for modifying a specific functionality. On the GUI, there is also an option for the user to view the source code.

Alternative 1: The user modifies functionality through the GUI.

1. The user modifies component functionality through methods of interaction such as integer input or text input.

Alternative 2: The user opts to modify functionality through the source code.

1. The system displays the portion of the source code pertaining to the selected functionality.
2. The user may opt to view a helpful suggestion on modifying the source code.
3. The user modifies and saves the source code.

9) Modify Project Source Code.

This option gives advanced users the ability to view and edit the entire source code.

Scenario 1:

1. The user selects the option to modify the project source code from the main menu.
2. The system displays the modifiable project source code.
3. The user modifies the source code and saves it to the Pi.

10) Save a project to the Raspberry Pi.

Steps common to all scenarios

1. The user selects the option to save the project to the Raspberry Pi.
2. The system prompts the user to name the project.
3. The user enters the name of the project to be saved.
4. The system checks to see if a project with that name already exists.

Scenario 1: A project with that name does not exist

5. The project is saved to the Raspberry Pi.

Scenario 2: A project with that name does exist.

5. The system notifies the user and prompts them to change the name of the project or overwrite the existing project.

Alternative 1: The user changes the name of the project.

1. The user opts to change the name of the project.
2. The system prompts the user to change the project name.
3. The project is saved to the Raspberry Pi.

Alternative 2: The user overwrites the existing project.

1. The user opts to overwrite the existing project.
2. The project is overwritten.

11) Save project to GitHub.

Steps common to all scenarios:

1. The project is first saved to the Raspberry Pi as in use case 10.
2. The system verifies that the user is logged into GitHub.

Scenario 1: The user is not logged into GitHub.

3. The system prompts the user to login to GitHub.
4. The user must login to GitHub as in use case 2.

Scenario 2: The user is logged into GitHub.

3. The system adds the user's GitHub username as a prefix to the project name.
4. The system pushes the project to GitHub.

12) Run the project on the Raspberry Pi.

Steps common to all scenarios:

1. The user selects the option to run the project. (The user must be logged into the Pi.)

Scenario 1: The project runs successfully.

2. The system runs the currently loaded project (the project which corresponds to the configuration displayed on the application breadboard interface).

Scenario 2: The project does not run.

2. The system notifies the user that there was an error running the project and advises them to ensure that they are logged into the Pi and that the project has been saved.

13) Delete a project from the Raspberry Pi.

Scenario 1:

1. The user selects the option to delete a project from the Pi.
2. The system verifies that the user is logged into the Pi.
2. The system displays a list of projects on the Pi.
3. The user selects a project to delete.
4. The system deletes the project.

14) Delete a project from GitHub.

Scenario 1:

1. The user selects the option to delete one of their projects from GitHub.
2. The system verifies that the user is logged into GitHub.
3. If they are not currently logged in, the system prompts them to do so as in use case 2.
4. Once the user is logged in, the system displays a list of their project repositories.
5. The user selects a project to remove.
6. The selected project repository is removed from GitHub.

4. Structural Design

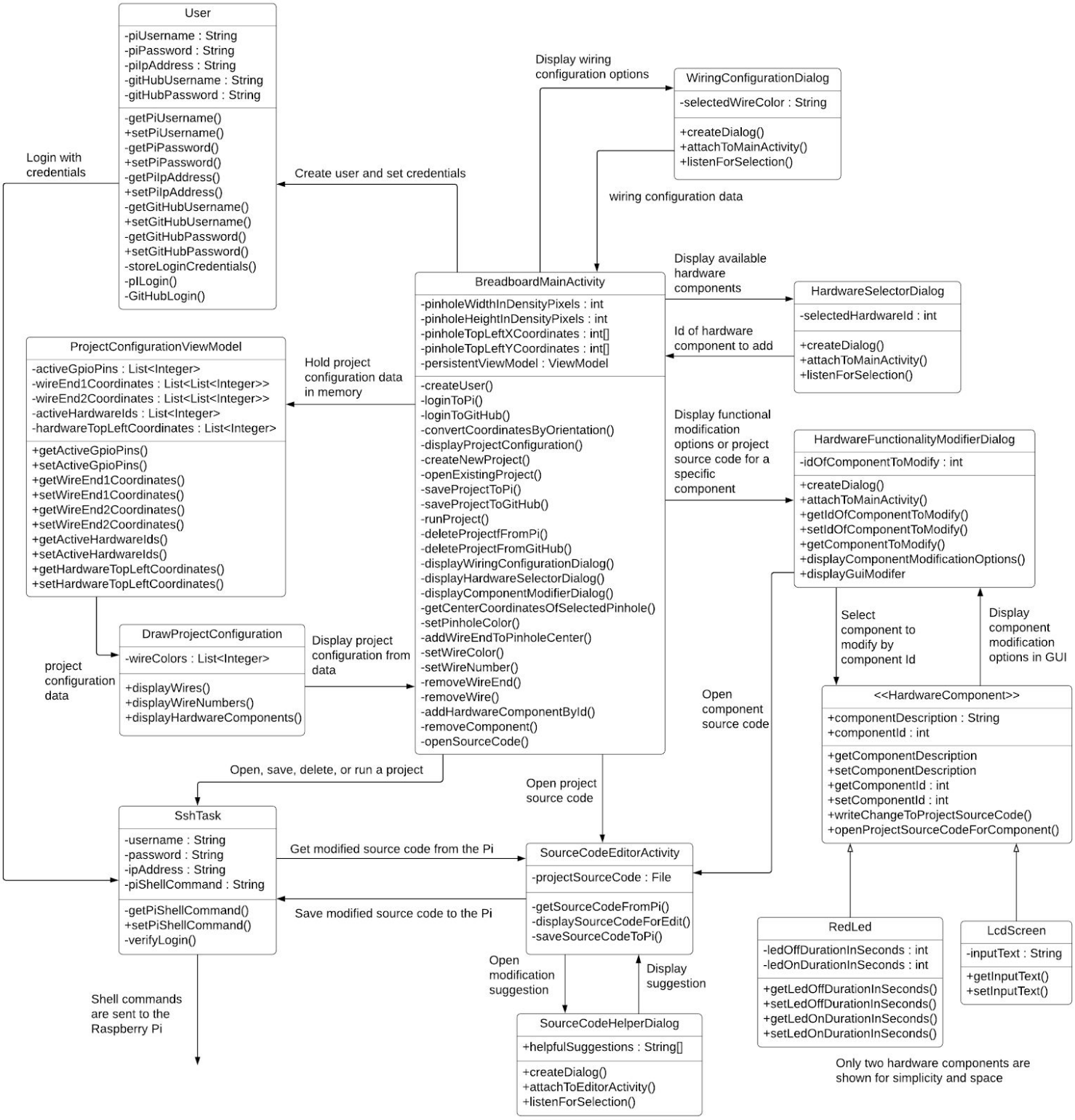

Figure 6: Class Diagram

5. Data

All persistent data is stored within the project configuration files of each project subdirectory. Persistent data primarily consists of the screen coordinates and IDs of wire ends and hardware components, and the unique pin numbers of active GPIO pins. In order to store this data, each project configuration subdirectory contains multiple text files, each containing the output stream of the serialized ArrayList or HashMap which holds the data during runtime. Since the data is stored as an output stream, it is easily read back into these data structures as an input stream when a project is opened. SFTP is used for the transfer of all files.

A total of twenty files may be dedicated to the wiring configuration alone, as ten files are potentially needed to hold the wire end coordinates, and ten files may be needed to hold the wire end IDs (also referred to as pinhole IDs). Twenty files are potentially needed as MoRPi offers five wire colors based on those that come standard with the official Raspberry Pi kit, and the IDs and coordinates of each end are stored separately. This data is initially used to display the the wiring configuration when a project is opened. One file is dedicated to holding the pin numbers of active GPIO pins so that their states can be changed to on when a project is opened. There are also two files dedicated to the project hardware configuration where one file contains the unique identification numbers of hardware components and a corresponding file contains the screen coordinates of the top left corner of each component. This information ensures that the correct hardware components are placed in the correct position when a project is opened. All coordinates are stored with respect to vertical screen orientation. Conversion between coordinates in vertical and horizontal orientation is handled programmatically.

Users are also able to store collected data. Each time a user adds a hardware component capable of reading in information, such as a temperature sensor, a subdirectory specific to that component is created under the parent project directory. A text file is then created in the component subdirectory where collected data is stored. In order to prevent the overwriting of collected data, a new text file is generated each time the project is run and named using the date and time.

6. User Interface Design

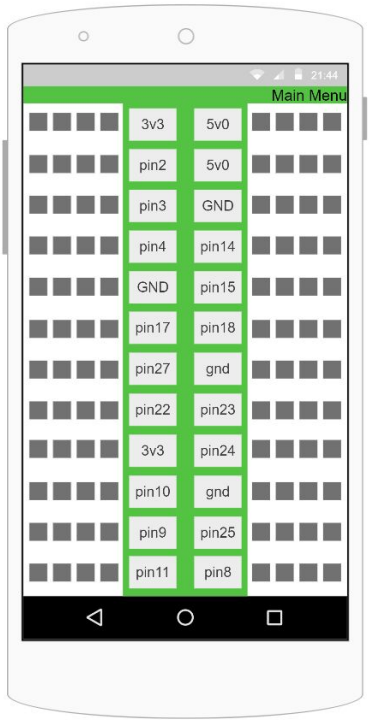

Figure 7: Breadboard (partial view)

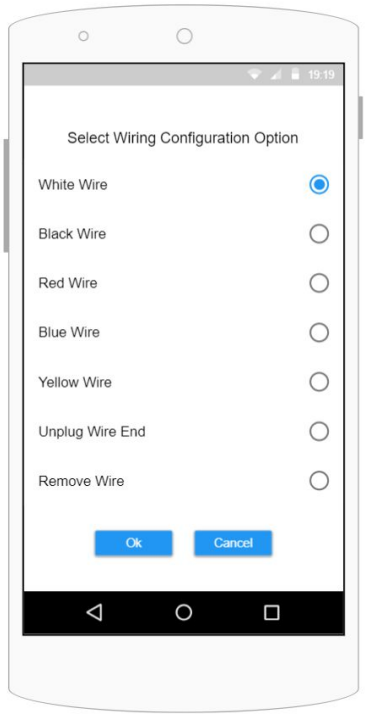

Figure 8: Wiring Configuration Options (from Figure 7)

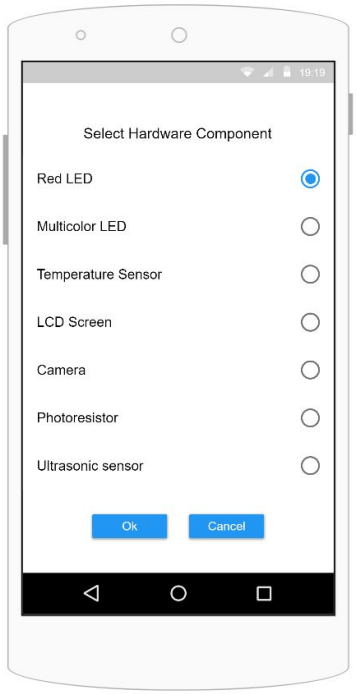

Figure 9: Select Hardware Component to add to breadboard (from Fig 7)

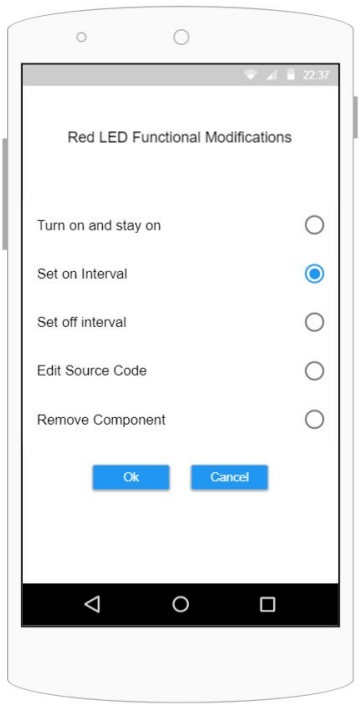

Figure 10: Component Modifications (from Figure 7)

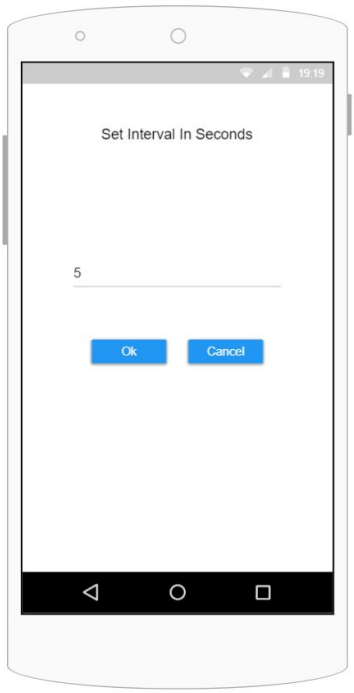

Figure 11: Modify Functionality in GUI (from Figure 10)

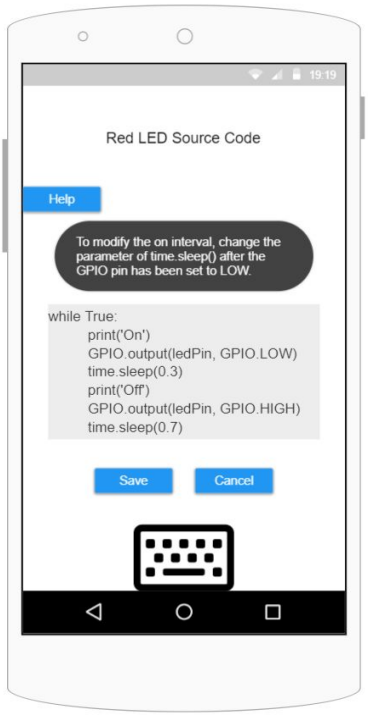

Figure 12: Edit functionality through source code with a suggestion (from Fig 10)

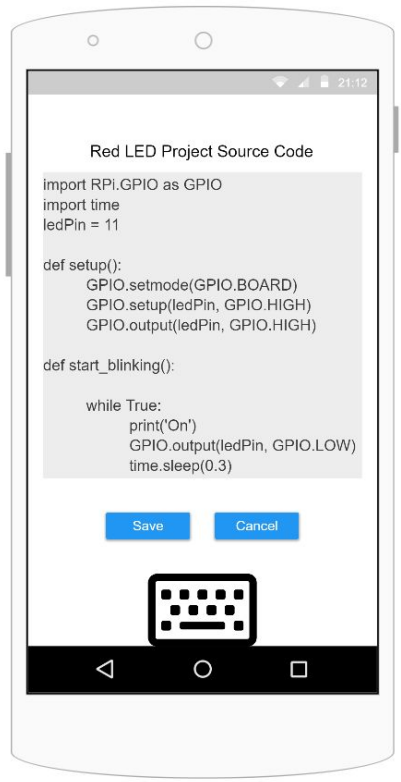

Figure 13: Edit entire Source Code (from Figure 7)

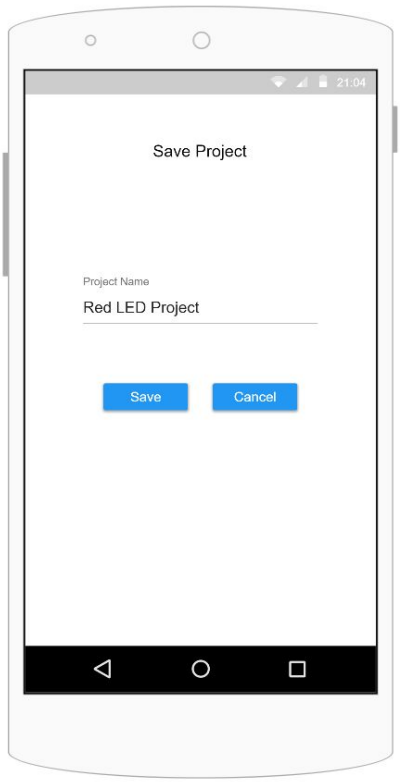

Figure 14: Save and Name Project (from Figure 7)

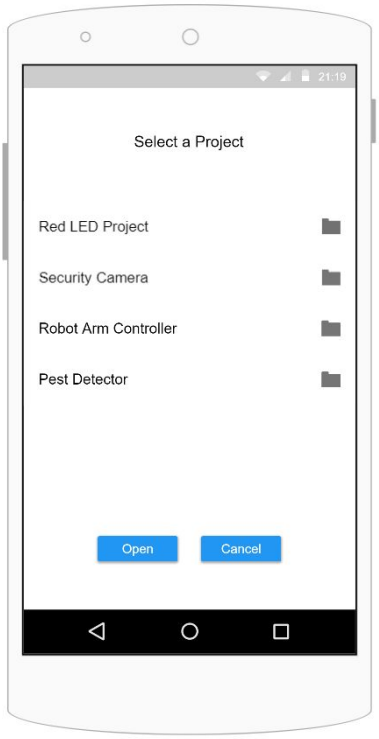

Figure 15: Select project to open (from Figure 7)

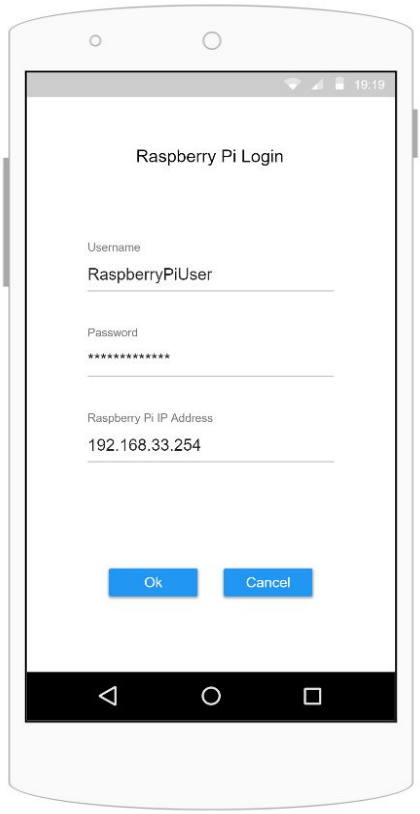

Figure 16: Raspberry Pi Login (from Figure 7)

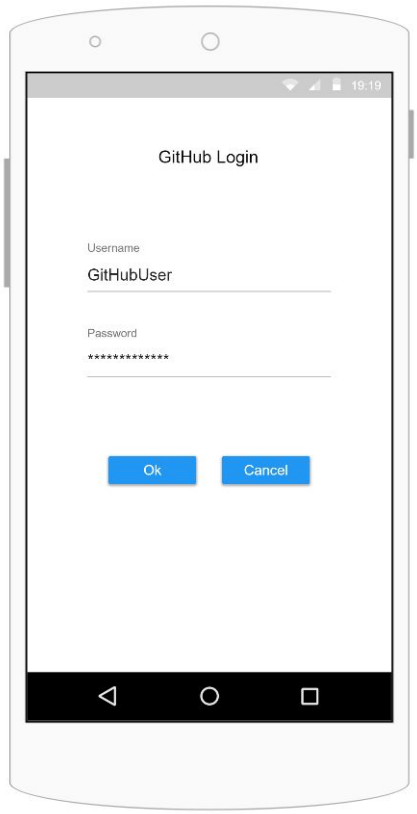

Figure 17: GitHub Login (from Figure 7)

7. Detailed Workplan