Office Door Kiosk

Team 28

Dr. Thomas Daniels

Weston Morgan - Product Lead

Jacqueline Johnson - Project Manager

Eric Rysavy - Technical Documentation Lead

Chris Duncan - Software Architect

Evan Foley - UI/UX Lead

Peter Laurion - QA

sdmay18-28@iastate.edu

sdmay18-28.sd.ece.iastate.edu

Revised: 4/22/2018

Contents

1 Introduction

The office door kiosk provides a quick and convenient means of communication for students trying to make a face to face connections with their professor, and for when professors want to display relevant information to people who come by their office.

2 Project Design (What it does)

2.1 Kiosk Application

The end product of this project is an office door kiosk application. This is an application that can run on any mobile device. The application has the following three modes

> Administrative mode: where professors can login and update their kiosk. This can be done both through the kiosk itself as well as through the professor's personal devices.
>
> Kiosk mode: where users at the kiosk can interact with professors and view content.
>
> Student mode: where the user can access the information form their own phones, but with limited access to the functionality of the kiosk mode.

The application also includes several “widgets” that the professor can configure and use to communicate with students via the kiosk/phone. Some of these widgets include, a display of office hours, a calendar, and a doorbell widget that a student at the kiosk can use to signal to the professor that he/she is at the kiosk and wants to converse.

2.2 Intended Users and Uses

This product is used by two different classes of user - the kiosk owner and the kiosk user - in a university setting this equates to a professor and a student, respectively. The kiosk owner can display personal and interesting information on the kiosk, such as pictures, schedules, contact info, notes, etc. The kiosk owner will also use the kiosk as a way for others to communicate with them. Kiosk users can use the kiosk as a way to learn about and communicate with the kiosk owner in a semi restricted manner, ie. only using video chat or sending a notification of their arrival at the kiosk.

2.4 Functional Requirements

- Build a secure enclosure for the kiosk device
- Remotely update the information displayed on the kiosk
- Display professor's customizable data on the kiosk
- Leave notes for students to view
- Do not disturb mode for times professor doesn’t want to be contacted
- Update/Display the professor’s calendar events
- Update/Display the professor’s office hours
- “Doorbell” that alerts the kiosk owner when someone is present
- Video Chat between kiosk and professor

2.5 Non-Functional Requirements

1. The kiosk must be resistant to thieves and trolls
2. The kiosk must be able to be removed by those authorized.
3. The UI must be responsive
4. User accessible design
5. Any number of Professors should be able to have accounts
6. No color based input elements

2.6 Existing Widgets/Functions

This section will briefly list and explain currently functional, implemented widgets in the application.

2.6.1 Notes

This widget allows for the admin to write and remove notes from the kiosk. This may include things such as “Ran out for lunch, be back at 1” or “I am currently in my office, but please do not bother me unless something **important** is on fire.”

2.6.2 Profile

This widget allows the admin to set a picture and name for the kiosk. This information is displayed, and can be viewed on the kiosk or by “student mode.” This widget also is how the admin retrieves the Kiosk/Student codes, which are used to access the kiosk and student modes.

2.6.3 Doorbell

This widget allows users of the kiosk to alert the administrator that they are at the kiosk. It is intended for professors who are nearby to their office to be alerted that someone is waiting on them. This alert goes to the administrator's phone.

2.6.4 Calendar

This widget allows the admin to set whatever events they care to inform others of. Events with specific hours on specific dates can be used. For example, a professor may want to set an event for "Out of town" for all of Thursday and Friday of a specific week, informing students and those who use their kiosk that they will be unavailable, despite what other sources (such as office hours) say.

2.6.5 Office hours

The office hours are a standard time during which professors are hypothetically in their office. This information is set by the administrator and can be viewed by the or kiosk or student versions of the application.

2.6.6 Settings

This widget, which is only available to administrators, allows for the enabling/disabling of other widgets, including setting time constraints. For example, an administrator may choose to not receive alerts when they are teaching a class, so they can disable the doorbell for those times. From this widget they can also enable do not disturb mode, which deactivates the kiosk until do not disturb mode is disabled.

2.6.7 Idle Page

After a set amount of time, the kiosk displays a generic idle page, showing the profile information of the administrator, as well as office hours. This information was selected as it will answer the most common questions that those walking by the kiosk will have.

3 Implementation Details

3.1 Implemented Design

For our final design, we implemented a React-Native application that will run on mobile devices and we used tablets to act as kiosks. All of our functionality of our application is decoupled into widget-like components that the professor can access and update remotely. Students can view this information and use certain functionality from either a tablet kiosk located outside the professor's office door, or from anywhere using a mobile device.

The professor can customize his/her own kiosk by restricting access to functionality. The professor can change the settings to only allow access to parts of the application based on the time of day.

This design allows for future expansion with new widget designs being easy to integrate. The application connects to a Node.js server using a combination of HTTP requests and WebSockets and is hooked up to a MongoDB database. The WebSocket implementation allows the professor to update the kiosk in real time.

3.2 System Block diagram

Below, Figure 3.1, is our final block diagram. It contains five major parts; tablet, phone, server, OneSignal API, and database. The phone can be any kind of cellular device and the application runs on any number of devices. The tablet works the same way, any kind of tablet can run our application. The server works as the connection between the tablet(s) and phone(s) and the database or each other. The phones are able to communicate with the tablet and update its display almost instantly, and the tablet is able to send communications to the phone. The database holds the information that the kiosk owner wants to display on their kiosk and various account information things needed.

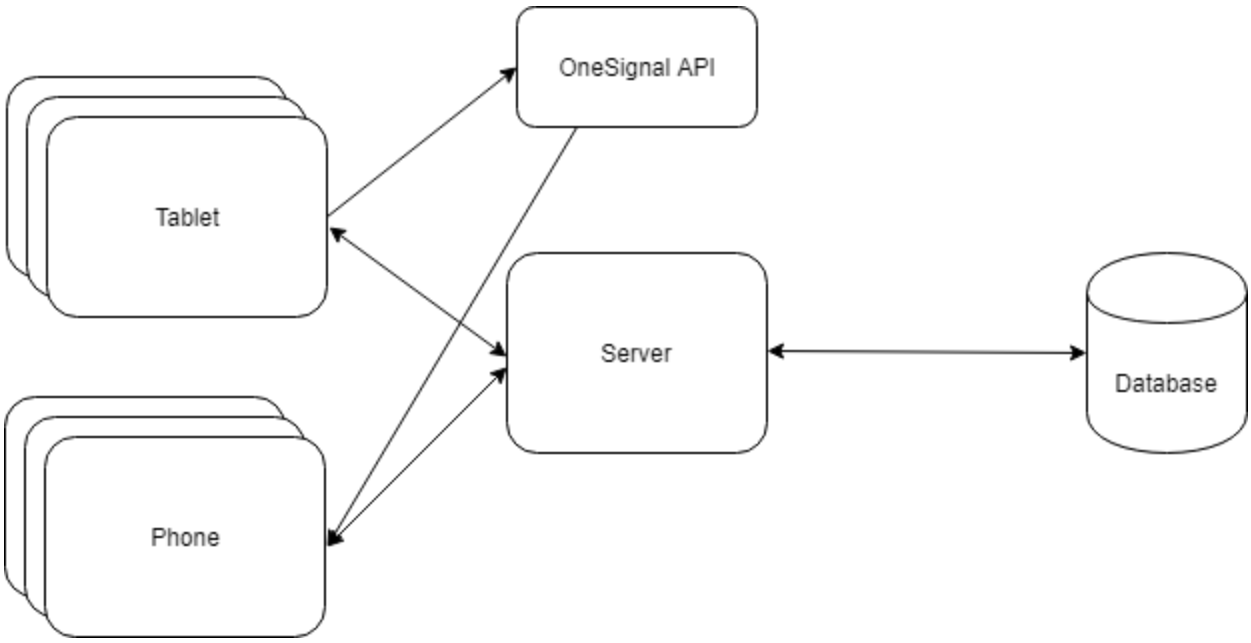

Figure 3.1: System Block Diagram

3.3 Functional Decomposition

Below, Figure 3.2 depicts our system's functional decomposition. Our application is decomposed into different widgets that are then decomposed into two sets of functionality: once set of functionality for the professor and another for the students.

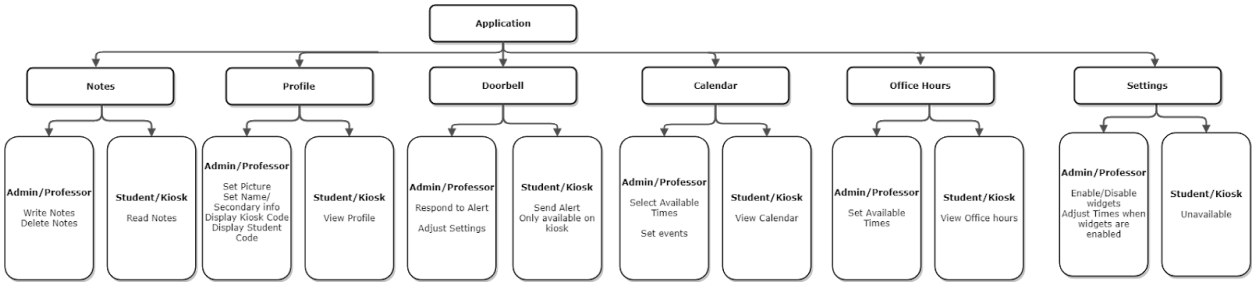

Figure 3.2: Functional Decomposition

3.4 Physical products

3.4.1 Overview

In this section we will list and defend the products we used for our final design.

This table lists the three purchases, and a url to Iowa State's preferred distributor for the products

Product	**Price**	**CDW URL**
Amazon Kindle Fire 7	$64.99	https://www.cdw.com/product/Amazon-Kindle-Fire-7-tablet-8-GB-7/4839833
Tablet Cradle	$36.99	https://www.cdw.com/product/Peerless-Universal-Tablet-Cradle-PTM200-wall-mount/2728537
Wall Mount	$15.99	https://www.cdw.com/product/Tripp-Lite-Display-TV-LCD-Wall-Monitor-Mount-Tilt-13in-27in-EXCLUSIVE-PRICE/3348649

3.4.2 Amazon Kindle Fire 7

This tablet was selected because it was the most cost effective option that still met our requirements. Since it runs the Android operating system, it was very easy to port our React-Native app onto it. Further, it is small, and very inexpensive for a tablet.
The cost is so low in part because there are advertisements that run on the lockscreen, but since our tablet never locks this shouldn't be an issue.

3.4.3 Peerless Universal Tablet Cradle PTM200

This cradle was selected because it is tamper-proof. Considering how easy it is to enter the buildings on campus at any time of day, it was critical that the tablet could not be easily stolen. In order to remove the tablet from this cradle, one must first remove the wall-mount from the wall entirely, and then disassemble the entire system. Without a toolbox and a significant amount of time, this simply cannot be done.

3.4.4 Tripp Lite Display TV LCD Monitor Wall Mount

There were two requirements for our choice in Wall Mount: being hard to remove, and interfacing with the Peerless Universal Tablet Cradle that was selected. As both this wall mount and the cradle use a 75mm x 75mm mounting interface, that requirement was met. Secondly, this wall mount is designed for full size televisions, not small tablets. It supports up to 88 pounds, so it will be very difficult to just rip out of the wall. In order to remove the mount, one requires a long, awkwardly shaped wrench, making the device effectively theft-proof.

3.5 Standards

Our code writing process followed agile development. This is a common practice that many companies use for developing code. It allows for rapid development, but more importantly it holds the other developers to

the same standards as their peers. These standards include both ethical programming practices and coding standards. We looked at the IEEE code of ethics to ensure that our practices were ethical. All of our standards for writing code were found by searching the web or personal preference. We do not believe that our standard would be deemed unethical by any party. Standards are applicable in our project because we want every team member to look at our code and be able to easily understand it. Having standards on how the code should be formatted allows for someone who didn't write the code to understand it more quickly.

1. Node
    a. Lines not too long that you can not read them easily.
    b. Two Spaces for indentations, looks better in web browsers and GitHub.
    c. Use named functions
2. React Native
    a. Consistent code organization
    b. Multi-line JSX, put each element being returned on a separate line
    c. Conditional JSX, declare empty variable at top, only populate it if the condition is met. Will render either the populated variable or nothing at all.
    d. 3 or more attributes on a component, display them on multiple lines
    e. Responsive interface design
    f. Multi-platform solutions

4 Related Products/Literature

4.1 Previous work/literature

While we have not worked on any similar projects, we looked at different examples of similar projects. For example, grocery stores use kiosks to provide a self checkout service, which speeds up checkout for most people. We have also experienced kiosks being used at career fairs for filling out forms while waiting in line to speak to representatives. Lastly, we have experience using kiosks for check in queues at places like the doctor's office. All of these examples provide different types of use cases that we can implement into our project. The downside of these examples is that their specific feature set is pretty limited. Small feature sets lead to smaller user groups. We wanted to make sure our project has a large enough feature set that different people could find many reasons to use the system. Having a feature like video chat will make this kiosk act as an actual way to communicate with the owner of the kiosk and not just a place to enter your information.

Email, Discord, and Skype are all applications that have features that are similar to the features we plan on implementing for our application. They also are finished products that the client would not be able to make additions to. Our project was made to be easily extendable, meaning if the client wants something more added in the future this can be done. This wouldn't be possible by using another company's software.

Our finished application is platform agnostic. This means that both android and ios are able to run our application, allowing more freedom for the client. Many other applications that have similar features do not have this flexibility.

This application was designed and implemented by Iowa State University students for Iowa State University Students and Faculty. We believe we have a better grasp of their wants and needs than an outside source.

There is also the possibility of an extension of this project be a future senior design project to add new features.

5 Testing Process/Results

5.1 Acceptance Tests

1. Verify that information can be remotely updated via phone
2. Verify that the professor's home page is displayed and is configurable
3. Verify that notes can be left by the professor and can be read by students at the kiosk
4. Verify that when the kiosk is in do not disturb mode, it cannot be accessed by students
5. Verify that the calendar functionality is configurable
6. Verify that when the doorbell is pressed at the kiosk, the professor's phone receives an alert
7. Verify that the professor can respond to a doorbell alert and the response is viewable from the kiosk
8. Verify that the device is difficult to remove from the enclosure
9. Verify that the app cannot be used for malicious purposes
10. Verify that the kiosk can be removed by authorized personnel
11. Verify that the UI is responsive
12. Verify that a large number of professors can create accounts

5.2 Test Plan

5.2.1 Functional Testing

During this project we utilized Jest as an automated testing framework. We used Jest to test our React Native functions and components using deterministic functional tests as well as snapshot testing. Using Jest, we tested to make sure that all of the app data and the app state are being updated correctly when certain functionality is used. For example, when the professor updates information on the app, we can verify that the information is correctly displaying on his/her remote version of the application as well as on the kiosk version. A set of functional tests as well as snapshot tests was created for each feature that the app contains.

5.2.2 Field Testing

Some field testing of our application was necessary to gage customer satisfaction. The enclosure for the device was tested for security and feasibility. In addition, field testing was needed to gage the look and feel of the application. Throughout the project, each member of the project tested the look and feel of the parts that they coded. Once all of the functionality was implemented, our UI design lead took over and made sure that the product was attractive according to his industry experience designing UI.

5.3 Results

The results of testing was that while useful to create unit tests and snapshots for some of the functionality, it was difficult to test end to end use cases due to the complexity of react-native conflicting with the intended simplicity of Jest. Jest was created to run unit tests, so checking things like adding new entries to the database is not what Jest was intended to do. As a consequence, the field testing was much more effective at verifying these types of uses. As far as snapshot testing goes, it did not go quite as expected. We could never get the snapshot of our app to work correctly and the calendar component relies on the current

date, so the dates will never be the same from snapshot to snapshot. This also led to field testing being more useful.

6 Appendix I - Operation Manual

6.1 About

This appendix functions as an Operations Manual, starting with a simple guide on how the software is used, and then going into how to set-up the software on both emulators and an Amazon Kindle.

6.2 Application Installation

For the sake of this section, we will assume you are using an Amazon Kindle Fire 7. If you would like to purchase this tablet, the link can be found in section 3.5.1.
Firstly, plug the device into your machine.
Secondly, run the following commands. Note, you will need npm and git to be installed on your machine.

```
git clone https://git.ece.iastate.edu/sd/sdmay18-28.git

npm install

mkdir android/app/src/main/assets

react-native bundle --platform android --dev false --entry-file index.js
--bundle-output android/app/src/main/assets/index.android.bundle --assets-dest
android/app/src/main/res

react-native run-android
```

Note, if you are building for an iPhone, you must instead run the commands below.

```
git clone https://git.ece.iastate.edu/sd/sdmay18-28.git

npm install

mkdir android/app/src/main/assets

mkdir ios/app/src/main/assets

react-native bundle --platform ios--dev false --entry-file index.js
--bundle-output android/app/src/main/assets/index.ios.bundle --assets-dest
ios/app/src/main/res

react-native ios
```

6.3 Usage

This application utilizes 3 separate modes in order to encapsulate functionality: Admin, Kiosk, and Student. Any of the modes can be accessed from the Login page. The Admin mode requires a username and password, while the student and kiosk modes only require a code provided during the admin account registration.

6.3.1 Admin Usage

From the Login page, first time users should choose to "Sign up". This will navigate the user to the Register page, where a username and password will be used to create an account. From there, the next page will ask the user to provide a name for their kiosk counterpart. Following that, the user will arrive on the settings page, where he or she can configure the accessibility of the features. This includes turning the features off completely, or only allowing access to the features for a period of time. After configuring settings, the user will be asked to customize their profile page with a picture, a name, and additional information. This page will also display the kiosk and student codes that will be used to access the correct information from the other modes.

The admin mode of the application focuses on updating information to display in the kiosk and student modes of the app. There are eight buttons to select from the home page, and some will require some information in order to be useful. First, the calendar widget provides users with a view of upcoming events. Use the "Add Events" button to open a modal. From the modal, the events name, start time, and end time can be provided and saved into the calendar. Events can also be deleted from the list by pressing the red "X" button. In order to add events to other days, open the calendar view and select the correct date. Then, complete the Event form as stated above. The Office Hours widget functions similarly to the Calendar widget. Office hours can be added from a modal, where the course, day, and times are edited by the user. Then they are displayed in a list and can be deleted by using the red "X" button. The Doorbell widget is used to send a push notification from the kiosk to the admin's phone. This will require admins to do minor set up in order to connect the devices. The instructions are displayed in the admin view, but the user must input the address of the receiving device to the sending device. To do this, open Admin view on the tablet being used for the kiosk mode and navigate to the Doorbell page. In the" Sending address" field, input the "Device Address" provided on the Admin's device, not the tablet itself. Once the correct device address has been provided, notifications can be tested by selecting a message from the picker and pressing the "Send Message" button. The Video Chat widget does not work. It is best to leave it disabled from the settings widget. The Notes widget allows the admin to add messages. To do this, navigate to the Notes widget, press the "Add Note" button, and input the message in the modal that pops up. Pressing the green "Add Note" button will add the note. The profile widget provides the admin with a page to input personal information to display on the kiosk. The settings widget allows the admin to disable widgets on the home screen, toggle "Do not disturb" mode, and set times for the widgets to be used. Each widget has a switch to turn them on and off, as well as times that can be edited in order to determine when the widget can be used. Lastly, the logout button allows the user to log out of the app.

6.3.2 Kiosk Usage

In order to access the kiosk mode, a user only needs to input a valid kiosk code and press the kiosk button. From there, the user can access and available widgets from the home page. The Calendar widget will display events. Events for other days can be viewed by opening the calendar view and selected the day. The Office Hours widget will display the admin's office hours. The Doorbell widget will allow the user to send a message as a push notification to the admin. The Video Chat widget does not function correctly, and should not be available in the kiosk mode. The Notes widget will display any notes left by the admin. The Profile

widget displays the admins profile, which will include an image of the admin, the admin's name, and will display the admin's office hours. If the kiosk mode is idle left on the homepage for two minutes, it will display the profile. To exit the idle page, click on the page near the image. The tablet should not log out of the kiosk mode, so there is no logout button.

6.3.3 Student Usage

From the Login page, a student should enter in the student code given to them by their professor into the corresponding text-box under the key login section of the page. The student will then press the "Student Login" in button, this will take the user to the home page. From the home page a student user has the same access as a kiosk user, but without being able to access the Doorbell nor Video Chat widgets even if enabled by the professor.

6.4 Common issues with Tablet setup

6.4.1 Computer cannot find device

Developer mode may not have been activated on the device. On a Kindle, navigate to settings. From here, select about, then look for either "Build Number" or "Serial Number" depending on the OS. Tap this selection 7 times to add a new option to settings, Developer Options. Open this menu, and there will be an option to active that will allow your computer to access the device and build unregistered apps to it.

6.4.2 Unable to load script from assets index.android.bundle

This can may be caused by out of date software, or through caching issues. This section will list commands to be run that should fix these issues.
First, update relevant software.

```
Npm install
Npm update
React-native upgrade
```

Assuming this didn't work, there is likely a caching/building issue. Try running these commands.

```
npm cache clean

mkdir android/app/src/main/assets

react-native bundle --platform android --dev false --entry-file index.js
--bundle-output android/app/src/main/assets/index.android.bundle --assets-dest
android/app/src/main/res

react-native run-android
```

7 Appendix II - Versions

7.1 About

This appendix will discuss changes that were made to the design, including what was planned on and why it was changed.

7.2 Versions Changed due to project knowledge

We scrapped our plan on creating a web application that would be accessible from any device and would use Google Chrome's kiosk mode. This was decided against due to the lack of knowledge on how we would connect two people over video chat on a web browser. It would be more difficult to get use of a tablet or phone's camera from a browser than an application running straight on the device. There was also the issue of the kiosk owner having difficulties updating getting onto a webpage on their phone and trying to update things. This would be more difficult for the user than on a app. We decided on these changes after discussing more thoroughly with our client on how he wanted to use our product.

7.3 Versions that failed

We originally planned to use the canonical Iowa State Single-Sign-On (SSO), Shibboleth. This would permit professors to sign into the system securely through pre-existing sign in information. Hypothetically, this would also be able to automatically populate the professor's calendar with classes they have listed. (It was also considered that we could permit students to sign into this system in order to access grade information and leave confidential messages to the professor. This subset of the Shibboleth idea was scrapped because the features were already implemented through Canvas and Email.) In the end, Shibboleth was taking more time than it was worth, so we instead are using the sign-in system through our own database.

8 Appendix III - Other Considerations

8.1 About

This appendix discusses several facets of our team's experience working with the many challenges of this project.

8.2 Lessons Learned

Throughout this process we as a team learned a lot with regards to project design, planning, and implementation. The first thing we all realized in the beginning of this project that selecting React Native and Redux as our framework is going to lead to some extra unexpected challenges. The first of these being that it is at times confusing framework to get started with and even though it starts to work really nicely as the project expands, during the beginning stages, it is very verbose and challenging to work with. The documentation from Facebook is fairly good overall but its is a pretty new framework so there are still issues and websites like Stack Overflow don't have all the answers like it would for a more established framework. This challenge of just getting React Native to do what we wanted meant our original timeline for work did not really cut it and we feel like we should've started really coding a bit earlier in Semester 1.

We feel like we did a fairly good job of gathering requirements and deciding on features that we feel were actually important but there were a few instances when we got in to the nitty gritty of implementation where we decided we needed to change requirements. For example, our calendar widget was originally going to be a scheduler but we realized that there wouldn't really be an efficient process for this without it basically being an email anyways so we decided to scrap that portion leading to some extra work because it we had already started to implement it.

Video Chat was also, in general, a failure on our part to do the leg work in the beginning. Because of React Native is pretty new there was only one library for video chat and in our initial testing it did seem to work but once we got a little deeper in to the project it no longer worked at all and we basically had no other option to turn to because it was too late to change our overall framework. If we had gone a little farther

with our video chat testing earlier on we might've changed to a different framework and might've been able to implement a working video chat widget. For a major portion of the project we had an issue with getting the code to run on a physical device. We didn't realize that we were having this issue until later on in the project due to everyone using emulators exclusively. Eventually this issue was addressed. It seems like this might have been why Video Chat wasn't working. Since we figured this out so late we didn't have enough time to reboot the Video Chat effort.

9 Appendix IV - Code

9.1 About

This section will offer examples of code to clarify on concepts discussed elsewhere in this document.

9.2 On adding widgets

As previously discussed, the kiosk has been designed in such a way that it can be easily extended in the future, should some other senior design group care to improve upon it. The best example of this is the ease of introducing new widgets, which cover nearly all of the functionality of the app. Should a group want to, as a random example, introduce video chat, it should be simple to integrate with what already exists.

As all of our code is organized in this fashion, they should be able to copy the formatting that has been done to add this functionality. While an understanding of react-native is needed, integrating the widget once it has been constructed could not be more straightforward. Below are three code snips that show how our Doorbell Widget is integrated to the system.

From AppNavigator.js. which allows the page to be imported to and render on HomePage.js :

```
export const AppNavigator = StackNavigator({...

DoorbellPage: {

     screen: DoorbellPage,

   },...

});
```

Please note that the two ellipses are stand-ins for similarly structured code that introduces other widgets.

Further, here is a snip from widgetReducer.js, which bundles all widgets to be easily accessed

```
const widgets = combineReducers({...

 DoorbellWidget,...

});
```

```
export default widgets;
```

It bears repeating that the two ellipses are stand-ins for similarly structured code that introduces other widgets in the exact same structure.

The last of the code that is required to add a widget to the app is located in HomePage.js, which renders the button for the widget to be accessed. The code for rendering the Doorbell button follows

```
<Row style={styles.Column}>

  <View>

    <Button disabled={this.ShouldBeDisabled(this.props.DoorBell)}
style={styles.RedCircleView} onPress={this.props.DoorbellPage} >

    <Image style={styles.ButtonImage} source={require('../assets/alarm.png')}/>
</Button>

    <Text style={styles.ButtonText}> Doorbell </Text>

  </View>

</Row>
```

This covers the changes that will need to be made should a developer care to add constructed widgets to the application.