rijksuniversiteit groningen

Hestia for iOS


# Architecture Document

**Client:** Lars Holdijk

**iOS Team:**

Andrei Mădălin Oancă

Daan Groot

Juan José Méndez Torrero

Jur Kroeze

Marc Fleurke

Tom den Boon

Zino Holwerda

**TA:** Feiko Ritsema

**Iteration: 7**

## Introduction

This project aims at extending the existing Hestia system, which is developed to be an extendable home automation platform. The current system consists of a server and an Android client. We extend it with an iOS client. A web server/site is also currently in development by a separate team of software engineers.

In this document we describe the design- and architectural decision that were made creating the front- and back end of the iOS application.

# Architectural overview of the entire system

The Hestia system contains various parts:

- Two clients, the iOS app, which is the main focus of this document, and an Android app.
- The peripherals, which can perform home automation tasks.
- A server, which provides a general interface for communication between the client and the peripherals.
- A web server/site, which will supply the login functionality as well as function as a bridge between the server and client.

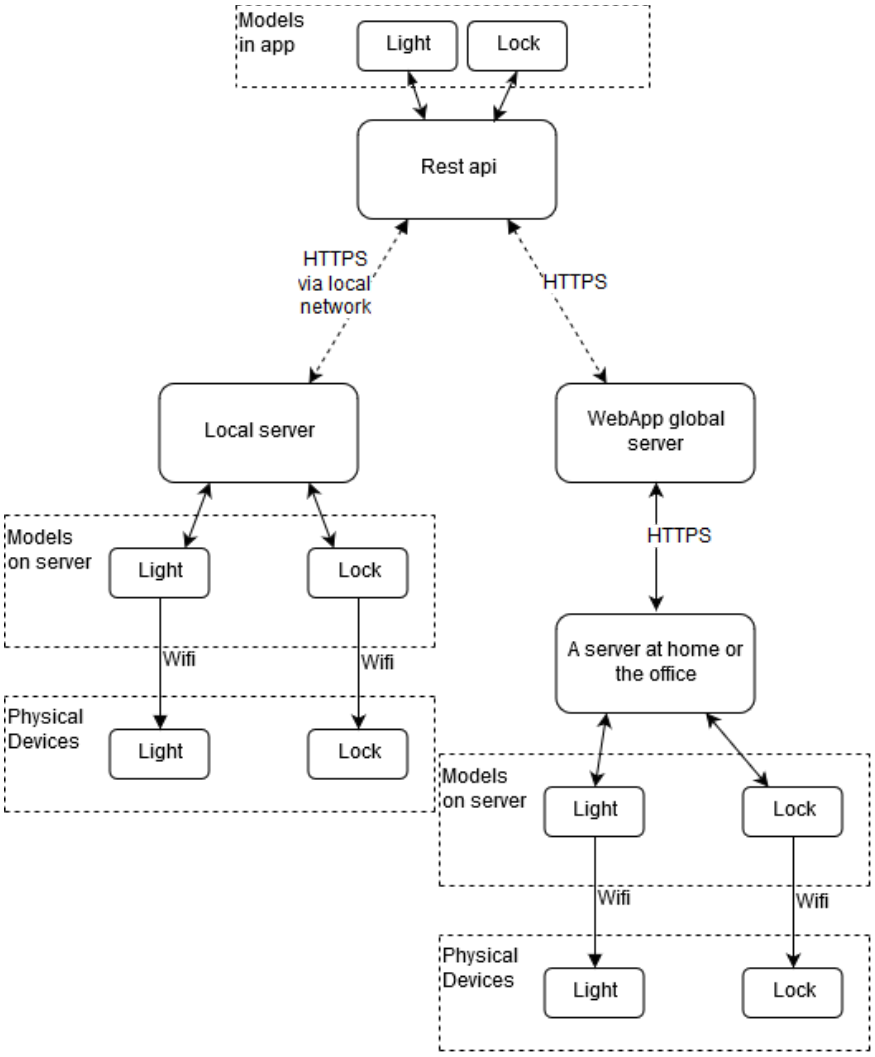



Diagram of the communication between the app, servers and devices.

## iOS client

The app is designed to be the skin that can be used to interact with devices. The functionality is nothing more than a wrapper around the computation that is performed on the server. The client communicates with the server through HTTP requests over an HTTPS connection. This connection follows the REST protocols. The messages that the server and client use to interact are JSON objects, this way the interaction remains uniform across different platforms.

With the current login system, the user can choose between logging in on a server that is on the local wifi network, or logging into a global server, so that the devices can be changed even when the user is not present. After login, the user will be shown a menu from which navigation is possible to different sections of the app. The main focus of the application lies on the device's screen. From here the user can change and add devices. These operations will use the API provided by the server and are performed on the server itself.

### Server

A quick reiteration of the functionality of the server. It functions on the local network as a REST API, which means it can be interfaced using the HTTPS methods: GET, PUT, POST and DELETE. The server keeps track of the different devices for the client to interact with.

# iOS app

## Architectural overview

The Hestia iOS app is split, very similarly to Android app, into a front- and back end. The back end section is concerned with sending and receiving JSON objects to and from the server, it also serializes and deserializes said objects. The frontend is concerned with formatting and presenting the gathered data to the user and making it editable. The back end also contains the needed code and data for interacting with the server, saving its address and other data and maintaining a list of devices with their activators.

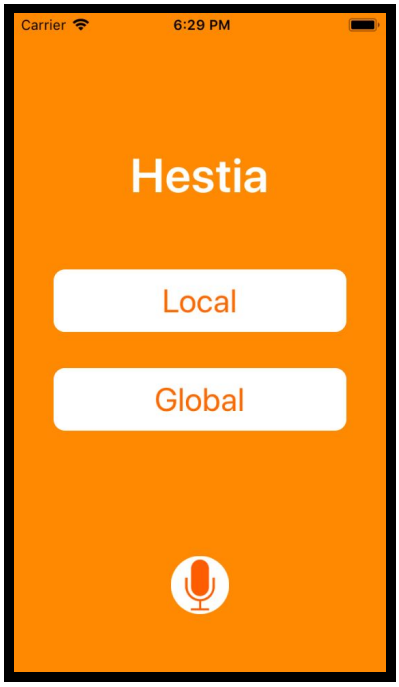
## User interface design

Following the demands of the customer, we tried to design a “native” iOS look and feel for the application. We took inspiration for the design from the following Apple developed iOS apps: Contacts, Remote, Music, Settings and Clock. Examining the way Apple designs its apps should be a good starting point to achieve the desired iOS look.


### Loading screen

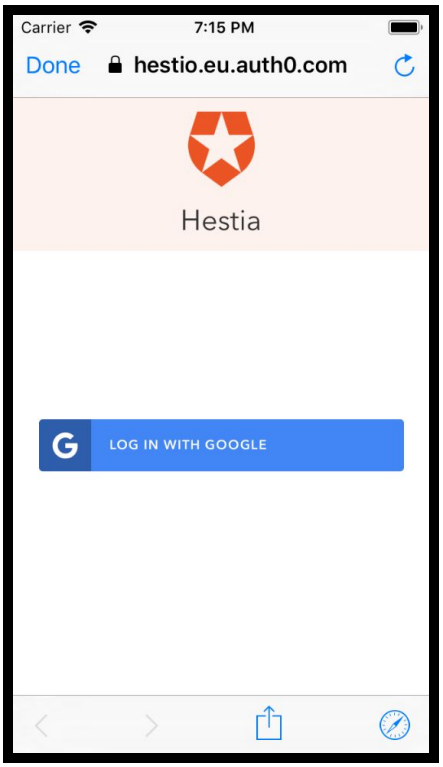
Apples built-in apps do not feature a loading screen, but third-party apps often do. We decided to keep things simple and use a white screen with the Hestia logo.

### Choose local/global screen




On first launch of the application, the user has to choose if he wants to connect to a server that is in his local network, or to the Webserver, which will give him access to his private servers. This is accommodated for by a screen with two buttons, reading local and global. Since the Hestia logo consist mainly of an orange color, we decided to make the screen orange with with buttons. The screen also feature a microphone icon that activates voice recognition. This can be used to verbally indicate the desired login type.

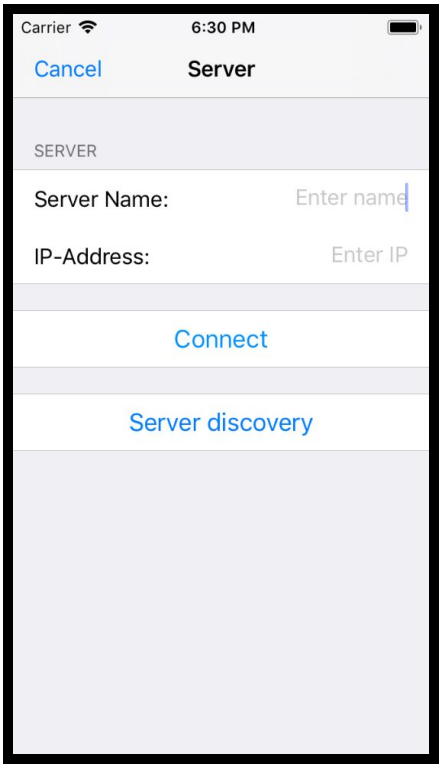
## Auth0 login screen




If the user taps the global button, the Auth0 login screen appears in a Safari browser window. This is the default way of displaying these login screens and because incorporating it differently in the app is very difficult, if not impossible, we decided to use this default option. This is the same way as for example a Google login screen appears in the iOS Google Drive app. The Hestia Web team is responsible for the layout of the actual login webpage. Currently the Hestia Web team supports logging in with a Google account. When clicking the Google button, the user is taken to the accounts.google.com-page. Here the user can sign in with his Google-account. After specifying the username and password, the user will return to the Hesta iOS app.

## Local login screen

The initial design of the app featured a local login screen, when connecting to a local server. After discussion with the customer, the screen was dropped, because the security of the local server is an issue for the owner of the local server, not the Hestia iOS app.




## Server connect screen

When tapping the Local button in the choose local/global screen, the user lands in the Server connect screen. In this screen he can enter a name for his local server, for reference within the app, and the IP address of the server. The cursor is automatically placed in the first field, such that the user can start typing immediately. On the bottom right key of the keyboard appears the word ‘Next’ for the first field and ‘Go’ for the last one, to be able to navigate quickly. Also a clear

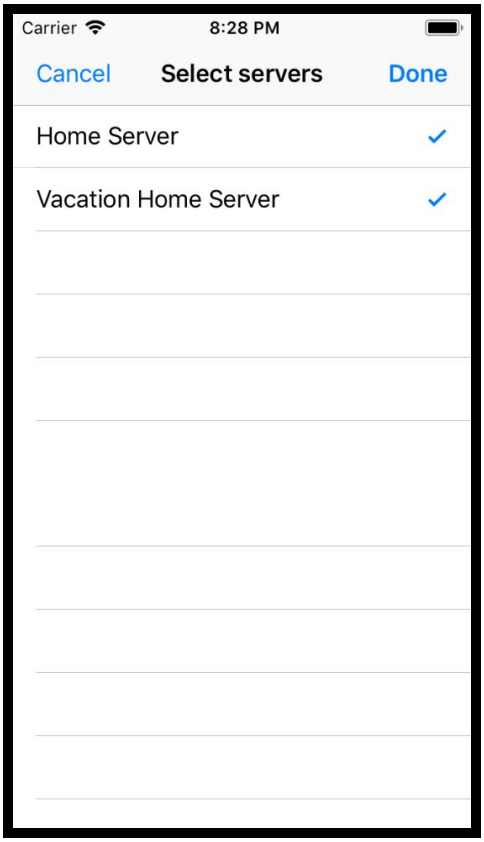
button appears in the fields when the user is typing, such that he can clear his input in one tap. If the user taps the connect button or the ‘Go’ key on the last field, it is checked that the input represents a real server. If this is true the user is lead to the Devices main screen.

### Considered alternatives

In previous versions of the app, the Server connect screen also contained an input field for the port of the server. We decided to remove this, as, by design of the server, the port is always 8000. Including this input field in this screen only causes inconvenience for the user, as he has to type more information.

## Server Select screen

After tapping the Global button in the Local/global screen and successfully completing the login, the user is presented with a list of (local) servers that are on the Web server. He can then select the servers from which he wants to see the devices on the Devices main screen. The selected servers are marked with a check mark. He can tap done to go the Devices main screen or cancel to go back to the Local/global screen.



## Settings screen

As many applications in use, the Hestia application includes a Settings menu. Is is accessible by tapping the Settings button in the Devices main screen. The contents of the Settings screen depend on the operating mode: Local or Global. In case of Local mode the this menu contains 4 buttons:

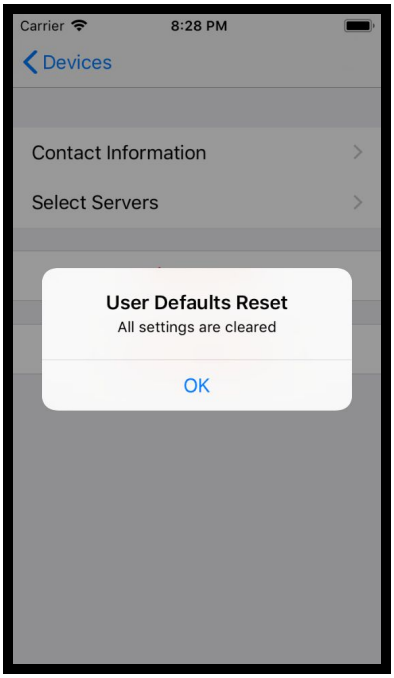
- Server
- Contact Information
- Disconnect
- Reset Defaults

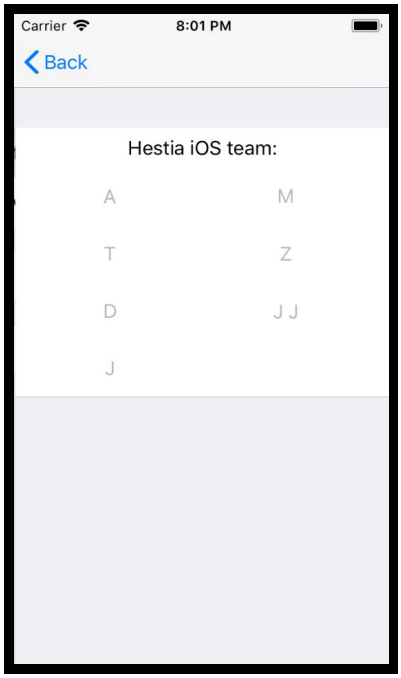
In the Server, the user will be able to change the server by typing the name and IP address of the new server that he wants to connect to.

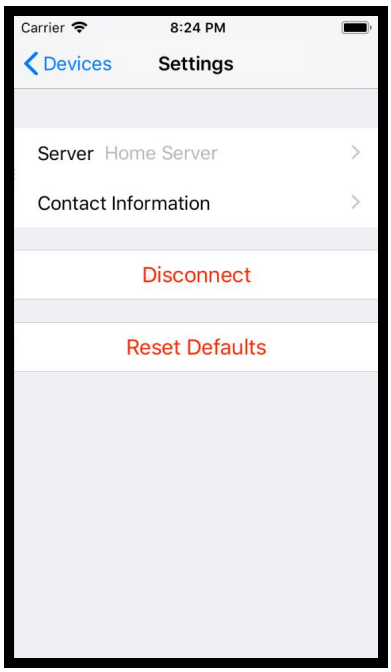
In the Contacts information, there will be information about our development team.

Disconnect leads the user back to the choose local/global screen where he can choose global.

Reset Defaults resets all user defaults, such as the IP address of the server and whether the last operating mode was local or global.



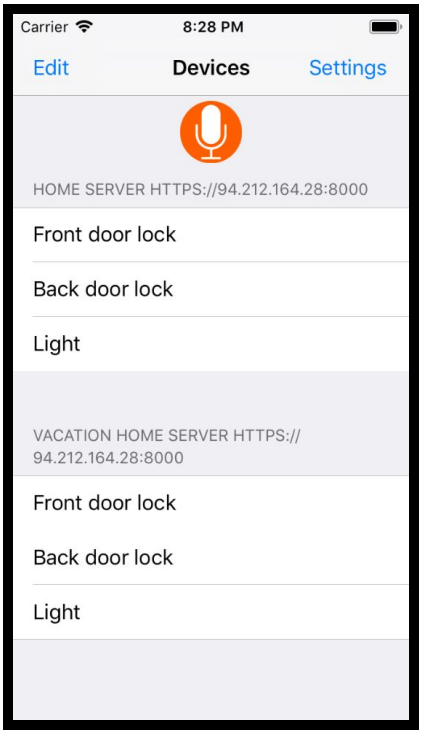
In case of Global mode there are also four options:

- Contact Information
- Select servers
- Disconnect
- Reset Defaults

Only the Select servers option is really different from the Local mode. The Select servers button open again, embedded in the Settings menu, the Select server screen. The user can now select again the servers of which he wants to see the devices.

## Devices screens

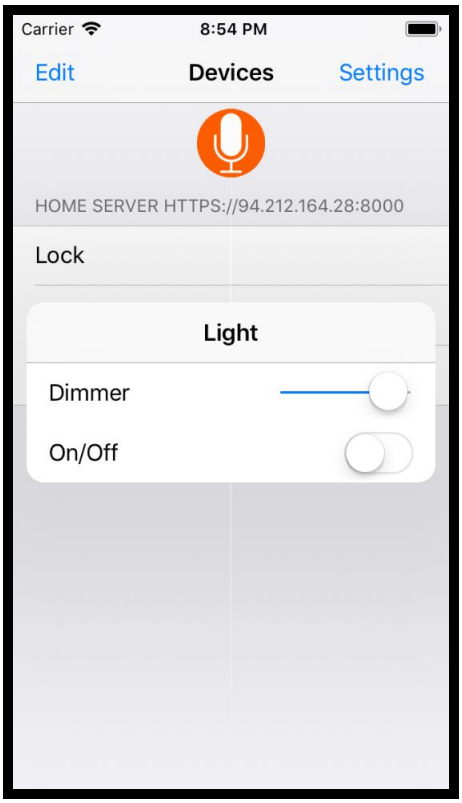
### Devices main screen




The devices screen has a central role in the app. The user arrives at this screen after logging in (global) or connecting to a server (local). The contents of the screen are similar for both cases. In case of local mode, the screen holds just one section with devices and in case of global mode, it has multiple sections: one for each server. If the user taps on a device cell, a pop up appears with the activators of the device. This can be the on/off button in the style that is abundant in iOS, for example in Settings or a slider in the same style. For example, if there are multiple “floating” activators for a device, multiple sliders will appear. If tapped on the next to the pop up window, they will disappear. So, initially the screen will be free from any activators, which gives a clean overview of the devices.

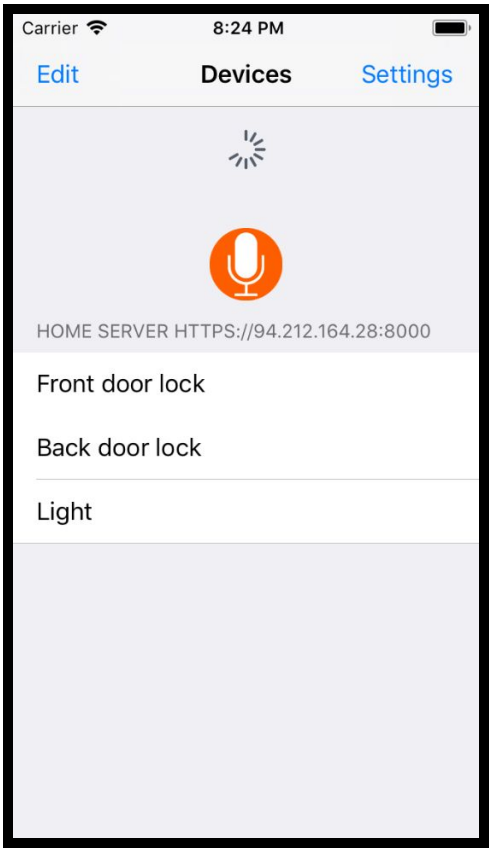
### Considered alternatives

First we considered a different approach for showing sliders. The idea was as follows. A boolean activators would be shown next to the device name. If a device is switched on, the slider(s) appear(s). If switched off, the slider(s) disappear(s). This keeps the screen also free from elements that are not functional and has the extra benefit that it gives the user a clear




suggestion on turning on the devices, that the device supports finer grained control. After discussion with the customer, we found that he did not like the sliders to be present if a device is turned on. This is also not in line with the flexible nature of the plugins: one does not know beforehand that a plugin has a boolean activator. Hence, that is why we decided to implement the alternative described in the section above.

In the top right of the devices main screen, one can find a settings button. This leads the user to the Settings screen as described above. The top left of the screen contains the edit button. On tap the screen turns to edit devices mode. The placement of the edit button is not consistent among iOS apps: sometimes it is on the right, sometimes on the left. We decided that placing the edit button on the left and Settings button on the right made the most sense, because the Settings screen slides in from right to left and once in the Settings screen, one can go back in a reverse transition. So, placing it in the top right gave the most intuitive look.



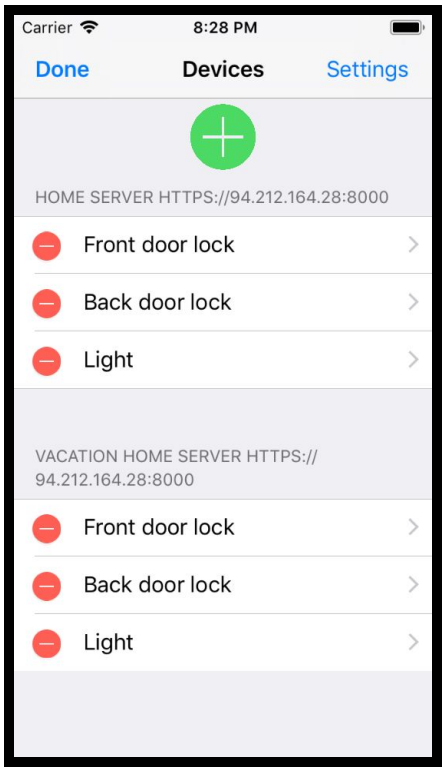

On top of the Devices list, one can find the Microphone icon, which activates voice control. The colors are inverted compared to the icon in the Local/global screen, because this color combination looks better on a bright colored background.

### Considered alternatives

We considered adding a “+” button in the top navigation bar in the devices main screen to add devices, but rejected this idea in favor of the way we add devices as explained in the next section. The main reason for this choice is that adding the “+” button instead of the

settings button, would force us to find another way of accessing the settings screen. This would lead to a less natural solution than the one established now.

Edit devices screen




On tap of the edit button, the main devices screen evolves in the edit devices screen. Before each listed device appears the standard iOS delete icon.

Besides the red delete icon, a right arrow “>” appears to the right of the device name, indicating that tapping it takes the user to a new screen (the (individual) device edit screen).

The Microphone icon changes to a green “+” icon, which leads the user to the Add devices screen.

Considered alternatives

In previous versions of the app, the insertion icon was placed at the bottom of the devices list, as a new “list item”. A green add icon and the text “New device” indicated that the button should be tapped if the user wants to add a new device. However, we removed this and placed the insertion icon on top of the devices list. This has the advantage that it is directly visible in case of many devices on the server, which could not be the case with the icon on the last row, which could be off screen.

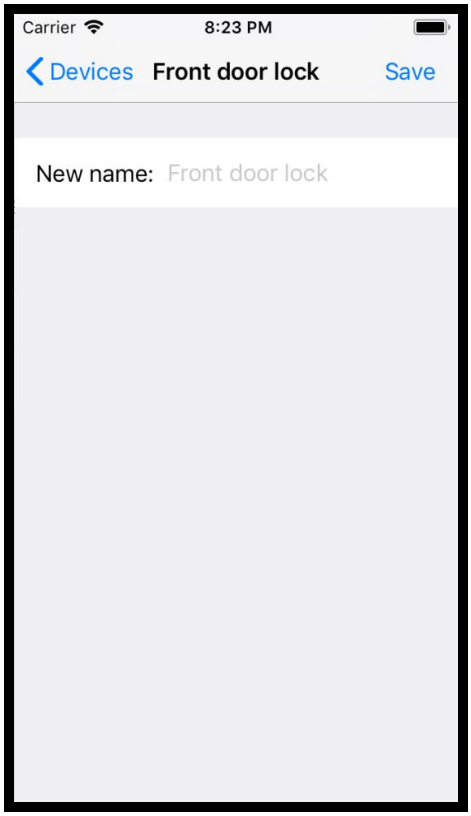
The way that the edit button responses will seem natural to iOS users. A similar response can be found in the Clock and Music app.

If the user taps the red deletion icon, a confirmation button slides in at the right side of the cell. If the taps this the device is deleted from the list.

Considered alternatives

We had the idea to implement the deletion confirmation as a pop-up window, but this turned out to be non-standard and very difficult to implement.

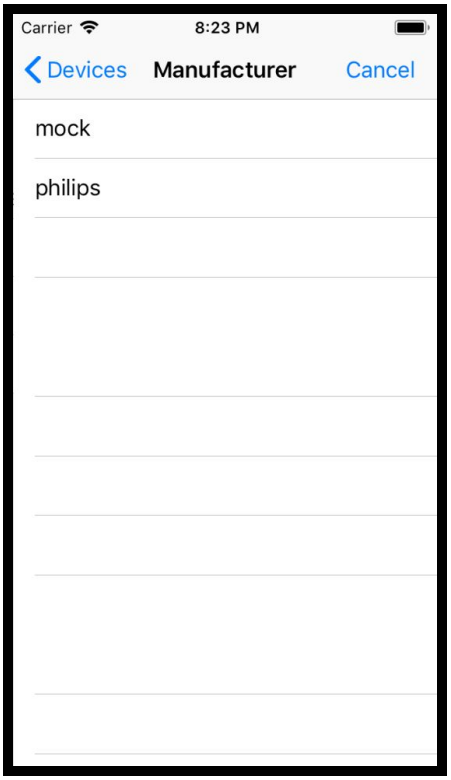
Device edit screen




This screen is entered when the user taps on the right arrow next to a device name in the Edit devices screen. Here, the user can change the name of the device. The user can confirm changes by pressing save in the top right corner, which leads the user back to the Devices main screen.

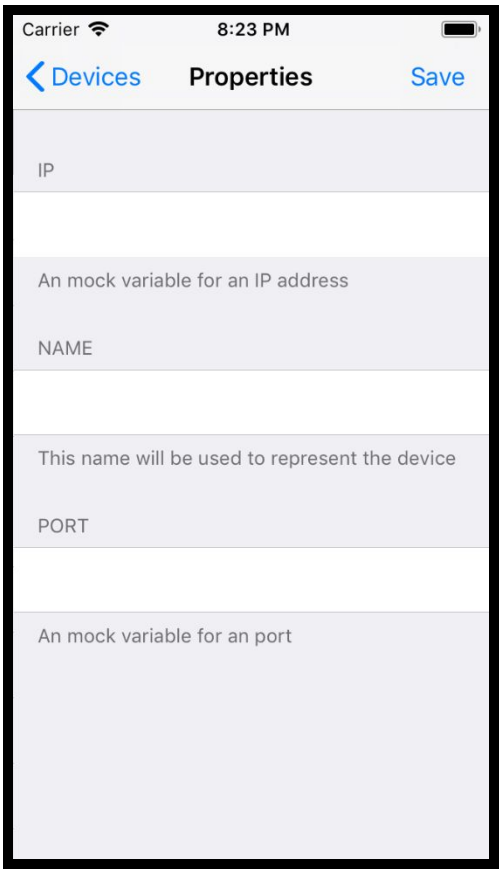
Add device screens

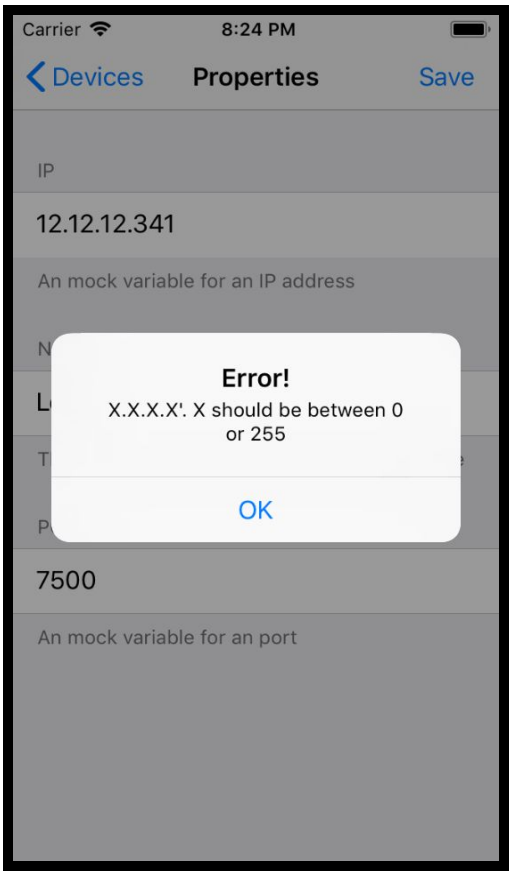
These screens are entered on tapping the green “+” icon in the Edit devices screen. The process of adding a device leads the user through a few different screen. First the user has to choose the Manufacturer from the list that is shown. If he taps a manufacturer, he goes to the next screen where he can select a device. Upon selecting a device, a new screen appears, where the user has to provide the properties for the device to be added. If the text input fields are filled in, where the user can enter for example a name for the device and the IP address at which it can be found by the server, he can tap the save button. If all goes well, the device is added to the device list and the user goes back to the main devices screen.




In both the Choose manufacturer and the Choose devices screen, a cancel button is shown in the top right corner, which leads the user directly back to the Edit devices screen.

In case of Global mode, the user is first presented with an additional screen that shows the list of visible servers. He has to tap the server that he wants the device to be added to.


Carrier
8:23 PM
Devices
Properties
Save
IP
An mock variable for an IP address
NAME
This name will be used to represent the device
PORT
An mock variable for an port


Carrier
8:24 PM
Devices
Properties
Save
IP
12.12.12.341
An mock variable for an IP address
Error!
X.X.X.X'. X should be between 0 or 255
OK
7500
An mock variable for an port

## Front end architecture

We used a combination of the Storyboard designer and coding in C# using Xamarin.iOS to implement the front end.

### Auth0 login

To incorporate the Auth0 login screen, we used the information from the Auth0 website.[1] It uses async methods to wait for the user input.

### Devices screens

The devices main screen shows a list of devices. A natural choice to implement this in Xamarin.iOS is a (UI)TableView. The TableView is dynamically filled with UITableViewCells, which contain the device name. In edit mode, a disclosure indicator is shown as Accessory to indicate that one can move to the next screen in a settings menu. The TableViewController (in which the TableView lives) is tied to a NavigationViewController to be able to show the Edit/Done button in the top right corner. Which button is shown depends on the mode of the TableView. If the TableView is in Editing mode, the button shows Done, and Edit otherwise. The Microphone icon and the "+" icon are conveniently place in the TableHeaderView above the table. The actual behavior of the TableView is defined in a class that is subclassed from the UITableViewSource class. This class provides a number of method overrides that determine, for example, how a row behaves when selected.

The screens used in the add device process are also TableViews with dynamic content, but simpler than the devices main screen.

The local Server connect screen is also a TableView, but one with static cells, because the number of cells is known in advance. In this way, the screen is much easier to design using the Storyboard designer.

[1] https://auth0.com/docs/quickstart/native/xamarin

The screen in which the user can edit the name of a device, is not a UITableView, but a combination of simple UIViews for the label and the input text field. This is because the amount of content of this screen is fixed.

### Xamarin.UITests

We chose this form of UI Testing due to 2 main factors. The main factor was that we already made a working app in xamarin, and thus the implementation of Xamarin.UITests is going to be easier than something like Calabash. Secondly it is really easy to set-up on an already existing project. Now if UI Testing is implemented correctly, we can now spot bugs with ease. For example, after changing code in the project. We do not need to check if the app is working correctly manually, but instead we can just run the tests.

### Exception handling

If something goes wrong when interacting with the server, a ServerinteractionException is thrown by the back end code. We catch those exceptions in the front end code and present a UIAlertController with an informative warning message. That is the typical warning message display in iOS. The user can then press “OK” and we prevent that the application performs unwanted actions or crashes.

### Input field checking

To prevent the user from specifying wrong information for a device, we apply Regex to check the name and (possible) IP address in the Add devices screen.

### Activity Indicator

To give the user a visual clue that the app is refreshing the devices from the server, we used a UIActivityIndicatorView, that is the familiar spinning gear from iOS.

## Back end architecture

The backend consists of the import network handler and server interactors which do most of the lifting for connecting to the servers. The network handler and two server interactors are three separate classes.

The NetworkHandler class is used for low level http requests. It has methods for GET, POST, PUT and DELETE requests. For handling the http requests we use a third party rest client package named RestSharp. Another option for doing http requests is with the build in HttpClient class. We have chosen RestSharp over HttpClient, because HttpClient requires the user to work with asynchronous tasks. RestSharp doesn't and therefore it's easier to use. For sending and retrieving data we use JSON, since the Hestia server only accepts JSON. We use the Newtonsoft framework to parse JSON, because it's an easy to use JSON framework and it comes with deserialization and serialization functionality.

Next there are the HestiaServerInteractor and HestiaWebServerInteractor classes which are used for communicating with servers. The HestiaServerInteractor class contains all necessary methods for interacting with a hestia server, e.g. getting a list of devices. The HestiaWebServerInteractor has as purpose to interact with the server created by the Hestia-Web team (from now on we will refer to this server as hestia web server). It can for example give a list with hestia servers or interact with a hestia server through the hestia web server. Both classes depend on the NetworkHandler, because they use its methods for executing http requests.

After we and the web team decided to use Auth0 for authenticating users we had to add support for it to our back end code. When a user successfully authenticates using Auth0, Auth0 will return an access token. This access token is used in the NetworkHandler to authenticate to the web teams server.

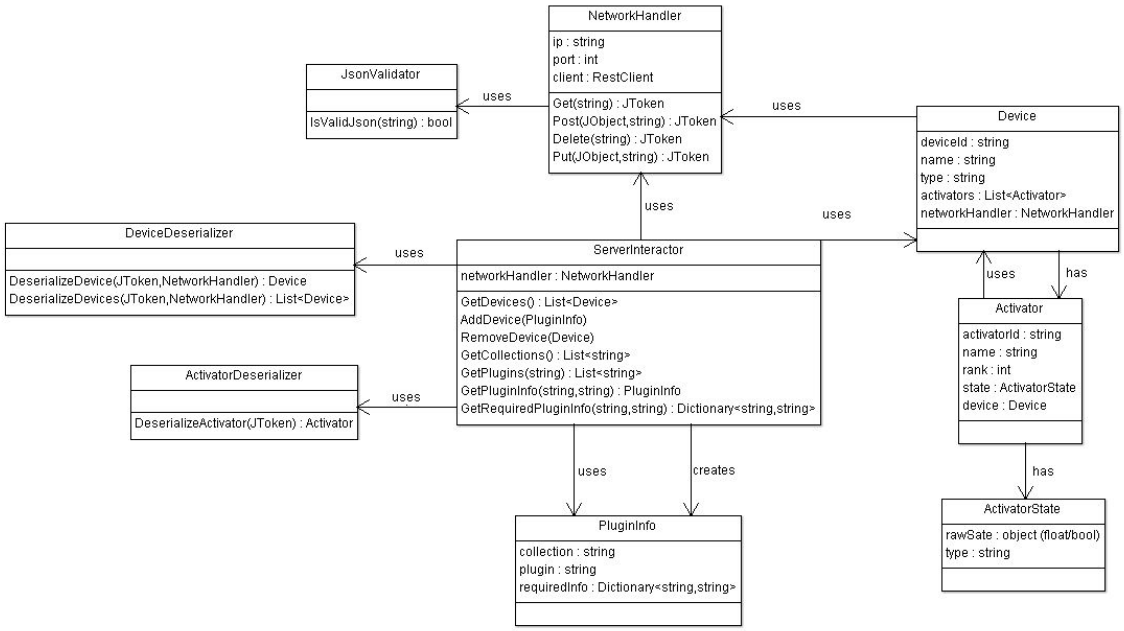
Besides the network handler and server interactors we have four packages: authentication, exceptions, models, utils and speech_recognition.

The exceptions package contains the ServerInteractionException class. The class inherits from the Exception class and is thrown when an error occurs during communication with a hestia or hestia web server.

Next up we have the models package. This package contains among other things classes that hold information sent or received by the client, namely Activator, ActivatorState, Device and PluginInfo. There is also the class HestiaServer located inside this package. This is a model for hestia servers received form the web team server. Information from the servers is received in JSON format, hence deserialization is required. Plugin info can be automatically deserialized with NewtonSoft, but the other models need some extra help. Therefore, in the models package we have the deserialzers package. This package contains classes that deserialize JSON to a C# object.

The final back end package is the Utils package. It has a JsonValidator class which can be used to verify if a string is a JSON string. There is also the PingServer class, this class allows the user to check if, given an ip and port, a hestia server exists.

The following diagram shows the interactions between all back end classes.



## Voice control

We have also implemented commands using the voice recognition API by apple, this allowed us to make the app mostly usable by voice. There is the possibility of turning on and off your home appliances such that you can be lazy and don't even have to scroll through that long list of smart devices you have in your house.

## Continuous integration

**App center**. We choose this for the incredible difference that was expected over travisCI. With app center from we averaged build times of approximately 2 minutes as opposed to twenty minutes we would have gotten from using

## Team Organisation

When we started working we took from our meeting with teaching assistant and customer that it might be wise to work in a Scrum like fashion. Besides this we divided our group up into two departments: frontend and backend. This way we could efficiently work on our separate parts. However we were not meeting on a daily basis, we made it weekly basis and separate meetings during the week in which we worked together.

To keep track of our tasks we used Trello to pass them to specific people and help planning through the sprints.

## Technology stack

Testing:

- Xamarin.UITests
- Xamarin.UnitTests

Backend

- Newton.JSON for deserializing.
- Resx files and indexing for resources
- RestSharp for interacting with the server.
- Apple voice recognition API.
- Auth0 for connectivity to the web team.

## Change Log

| Who | When | Section | What |
|---|---|---|---|
| M. Fleurke | February 23, 2018 | Document | Created document + headings |
| Z. Holwerda | March 1, 2018 | Document | Modified the front page to match other document. |
| M. Fleurke | March 11, 2018 | User Interface design | Section about device screens |
| M. Fleurke | March 12, 2018 | User interface design | Adding more on device screens + screenshots |
| A.M Oanca | March 12, 2018 | User interface Design | Added Settings description + screenshots |
| Z. Holwerda | March 13, 2018 | Document | Changed version style, added a small general overview. |
| M. Fleurke | March 25, 2018 | Front end architecture | Added architecture of devices screen |
| Z. Holwerda | March 27, 2018 | Back end | Added overview of the app. |
| J. Kroeze | April 17, 2018 | Back end | Added description of deserializers |
| A.M. Oanca | April 17, 2018 | Front End | Changed Screenshot according to the application. Changed description for some of the subparagraphs. |
| M. Fleurke | April 17, 2018 | Introduction | Wrote short introduction + small fixes and additions in other sections |
| G. Barkmeijer | April 17, 2018 | Front end | Updated the section of the login screen. |
| Z. Holwerda | April 17, 2018 | Team organisation | Changed the style of the document slightly. |

| | | | |
|---|---|---|---|
| | | and document. | Small team organisation. |
| D. Groot | April 17, 2018 | Back end | Added info about the network handler and server interactor. |
| D. Groot | May 1, 2018 | Back end | Added a class diagram for back end. |
| M. Fleurke | May 13, 2018 | Front end | Updated information on front end screens |
| M. Fleurke | May 29, 2018 | Front end | Updated information on front end user interface and architecture and updated screenshots. |
| T. den Boon | May 29, 2018 | Front end | Updated UI Testing. |
| J. Kroeze | Mag 29, 2018 | Back end | Added server interaction diagram, update general introduction |
| Z. Holwerda | May 29, 2018 | Document | Technology stack, styling |
| D. Groot | May 29, 2018 | Back end | Updated back end architecture section |