rijksuniversiteit groningen

Hestia for iOS

Architecture Document

Client: Lars Holdijk

iOS Team:

Andrei Mădălin Oancă

Daan Groot

Geanne Barkmeijer

Juan José Méndez Torrero

Jur Kroeze

Marc Fleurke

Tom den Boon

Zino Holwerda

TA: Feiko Ritsema

Iteration: 1

Introduction

Architectural overview of the entire system

The current 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 a function as a bridge between the server and client.

Client

The app is designed to be 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 object, this way the interaction remains uniform across different platforms.

The current login system is a temporary solution, this will be changed to use Google' Firebase. The login will also not happen locally anymore as cooperation with the web server will enable logging in from everywhere using said system. 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

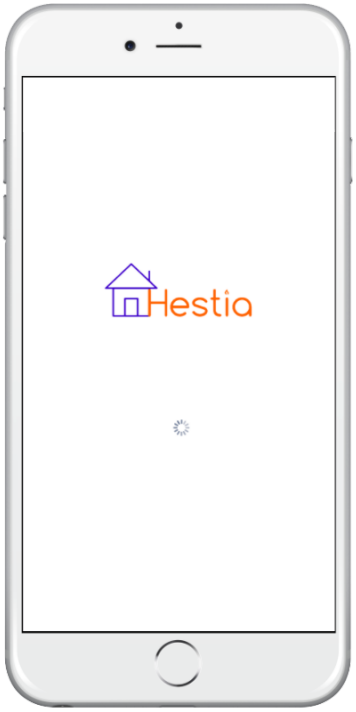
Architectural overview

The Hestia iOS app is split, very similarly to android app, into a front- (No separate yet) and back end (package backend). The back end section is concerned with sending and receiving JSON objects 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.

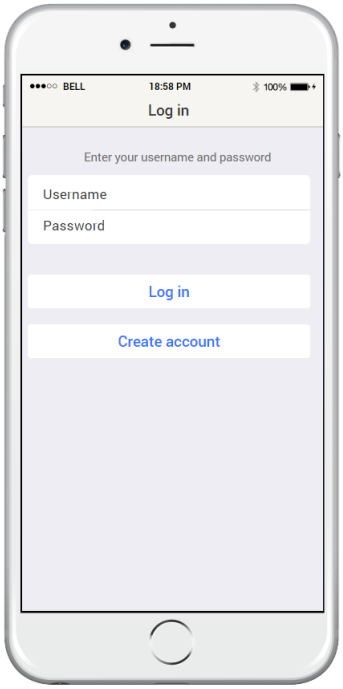
User interface design

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



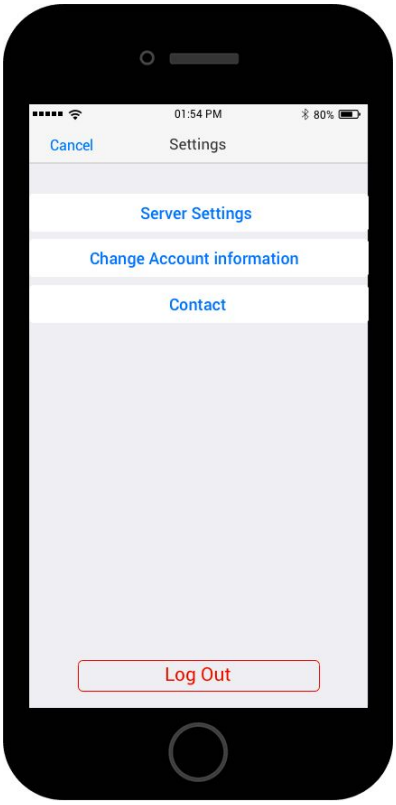
Apples build-in apps do not feature a loading screen, but third-party apps often do. We decided to keep things simple and use a white screen with the Hestia logo and a small circular loading indicator that can be found in iOS apps and the iOS operating system itself, for example on startup of the iPhone. This gives the user a visual clue that the app is starting up.



Login screen

To be able to control devices, the user should login to the app with a username and password. This screen is kept as simple as possible using a login screen template from the Justinmind prototype software, which was modified a little to accommodate a Create account button. This button in necessary for a user that has not yet an account to create an account and use the app. Further, the screen contains two input text fields and a login button.

Settings screen



As any application in use, the Hestia application includes a Settings menu. This menu contains 4 buttons:

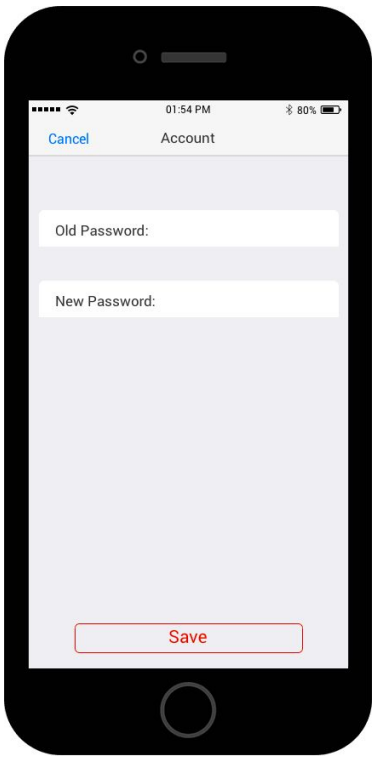
- Server Settings
- Account Settings
- Contacts
- Log out

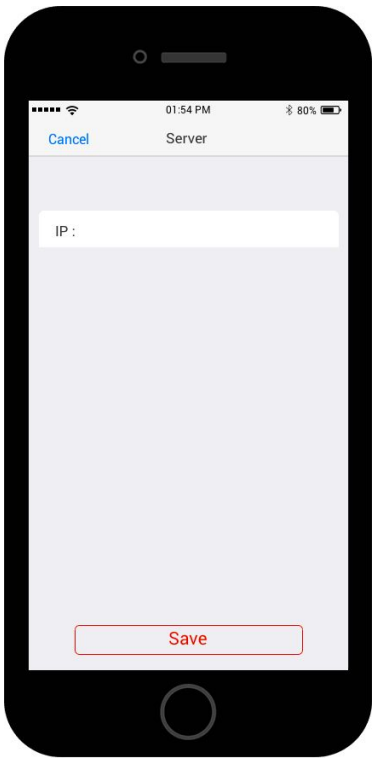
In the Server Settings, the user will be able to change the server by adding the IP address of the new server that he wants to connect to.

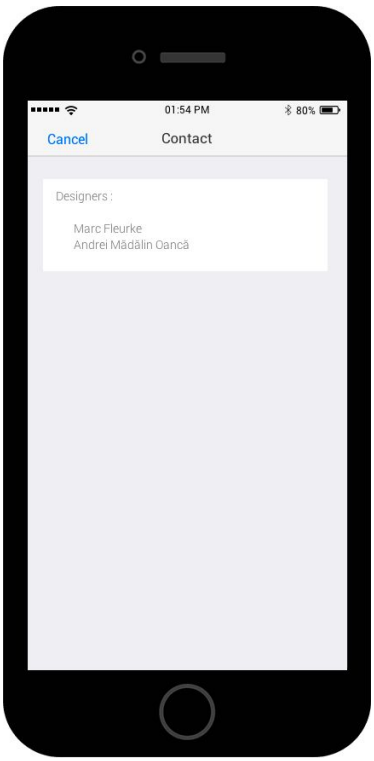
In the Account Settings, the user will be able to change his account's password by introducing the old one and the new one.

In the Contacts, there will be information about how to get contacts information of our development team.

Log out just terminates the session for the current user and goes back to the Log In screen.

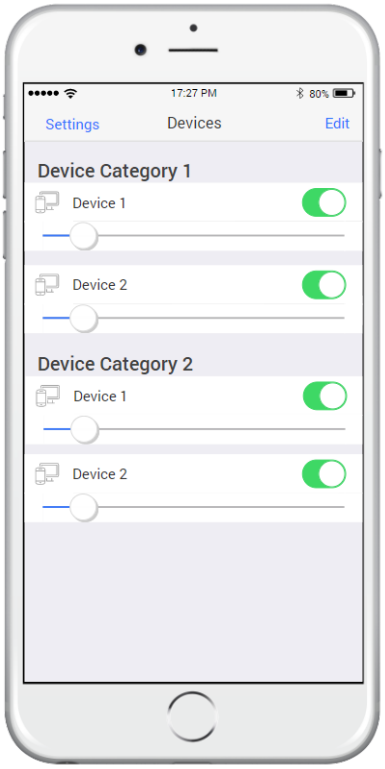


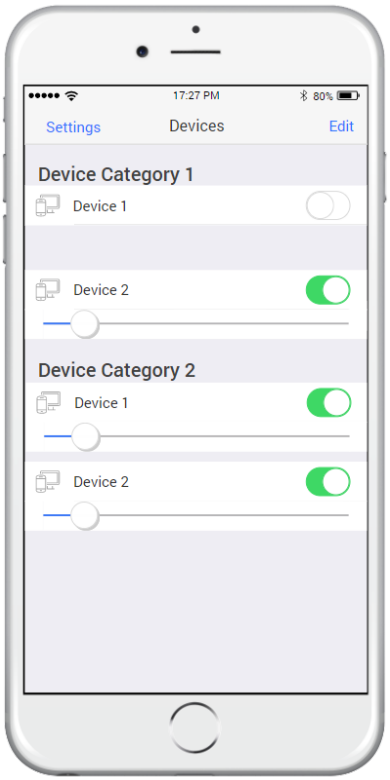




Devices screens

Devices main screen





The devices screen has a central role in the app. The user arrives at this screen after logging in. The screen contains a list with devices, ordered by category. The device name is preceded with a small icon to provide a visual hint to the user regarding the sort of device. Following the device name is an on/off button in the style that is abundant in iOS, for example in Settings. Some devices support a “floating” activator, for example, a dimmable light. For these kind of devices, a slider appears when the device name is tapped. If there are multiple “floating” activators for a device, multiple sliders will appear. If tapped again, they will disappear. So, initially the screen will be free from any sliders, which gives a clean overview of the devices.

Considered alternatives

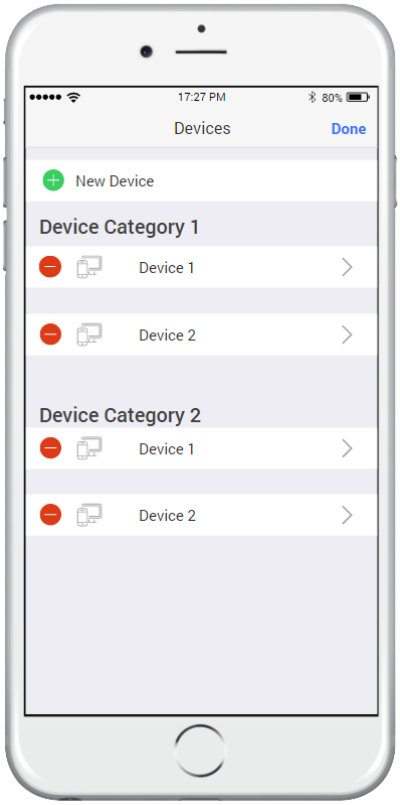
First we considered a different approach for showing sliders. The idea was as follows. If a device is switched on, the slider(s) appear(s). If switched off, the slider(s) disappear(s). This keeps the screen also free from elements that are not functional and has the extra benefit that it gives the user 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. Hence, that is why we decided to implement the alternative described in the section above.

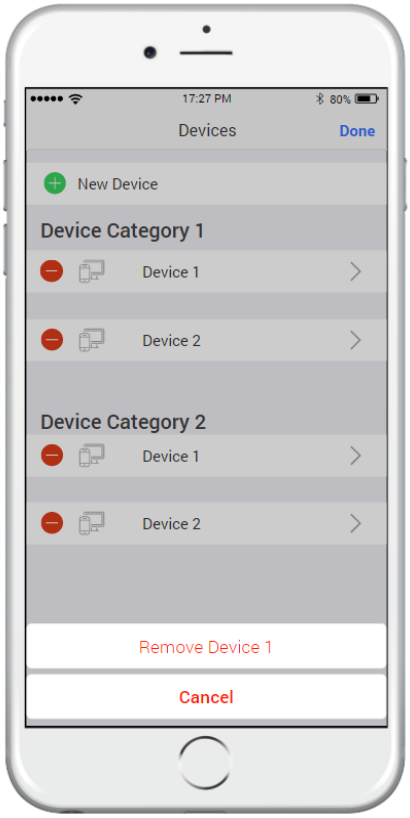
In the top left of the devices main screen, one can find a settings button. This leads the user to the settings screen. In the top right of the screen contains the edit button. On tap the screen turns to edit devices mode. The placement of the edit button is not consistent among iOS apps: sometimes it on the right, sometimes on the left. We decided that the right would make more sense, as our application does not have a “+” button in the devices main screen, that is found in many apps the have the edit button placed in the left corner. Moreover, it is more easy to reach for right handed people.

Considered alternatives

We considered adding a “+” button 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. Tapping on the delete icon issues a pop-up on the bottom of the screen, presenting the user two choices: Remove device and cancel. This is a way of asking confirmation of the user that is applied often in iOS apps.

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).

On top of the devices list, a new “list item” appears. A green add icon and the text “New device” indicate that the button should be tapped if the user wants to add a new device. Indeed, on tap, the add device screen appears.

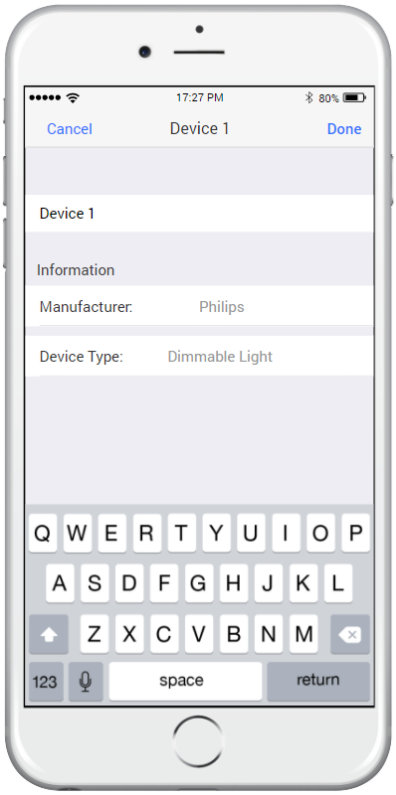
The way that the edit button responses will seem natural to iOS users. A similar response can be found in the Clock and Music app.

Considered alternatives

Instead of the pop-up that appears on tapping the delete icon, we also considered showing a delete confirmation button on the right of the device name, as is done in for example the Clock app. This seems to give users a less severe warning than the pop-up. Since removing a device is not something that can be undone very quickly, it seemed a good idea to give a more severe warning.

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. The screen is by default in edit-mode, such that the name of the device can be changed. Also other static information about the device is shown here. The user can confirm changes by pressing done in the top right corner, which lead the user back to the devices main screen, or undo changes by pressing cancel in the left top corner.

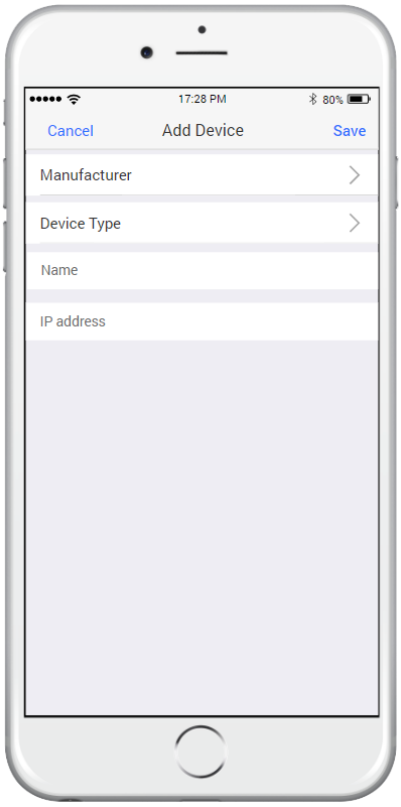


Considered alternatives

The done button could also lead the user back to the edit devices screen. But since the user usually wants to change only the name of one device at a time, it seemed more logical to go back to the devices main screen.

Add device screen

The screen is entered on tapping New device in the edit devices screen. The user has to provide four properties for device to be added. First, he should tap the Manufacturer button, which leads the user to the Manufacturer list screen as described below.



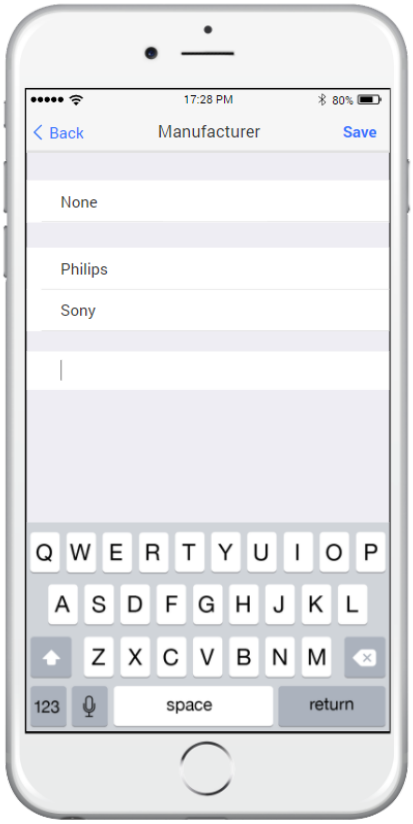
The second button is the Device Type button, which behaves similarly to the Manufacturer button.

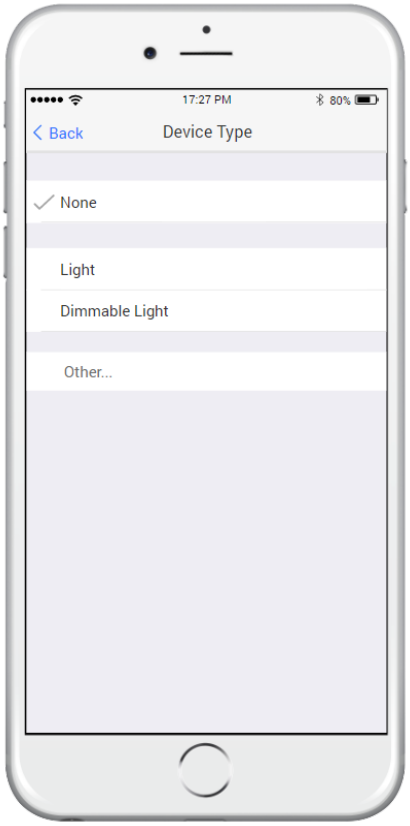
Then two text input fields follow, where the user can enter a name for the device and the IP address at which it can be found by the server.

The additions can be undone by tapping cancel, or saved by pressing save.

Manufacturer list screen

In this screen, the user can select the manufacturer of the device or select none if it is unknown. The manufacturers that are known on the server are presented in a list. The user can also enter a manufacturer manually. Tapping a manufacturer automatically takes the user back to the add device screen.





Device type list screen

This screen is like the Manufacturer list screen, but now shows device types.

Front end architecture

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

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 and on/off switch if present. This switch in placed in the

AccessoryView. This is the place on the right of a cell, where normally for example a right arrow is shown 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 'New Device' is also a UITableViewCell, which is shown if the TableView is in Editing mode. The actual behavior of the TableView is defined in a file 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.

Back end architecture

The backend consists of the import network handler and server interactor which do most of the lifting for connecting to the server. Besides this we have three packages: exceptions, models and utils. Utils is rather self explanatory it contains some essential utilities which have various appliances. Furthermore we have the exceptions class, currently we have but one exception; the Com exception, is thrown when the network handler fails on connection, however not yet implemented. Lastly we have the models package, here are the three important classes which define how a device is read and wrote in a JSON file. We use these to communicate to the server which stores them. For each section necessary for describing a device: Device, Activator and RequiredInfo we will have a serializer to turn them into JSON object for sending.

Our network handler uses two things id and port. This will return the network handler object which can be passed to the Server interactor such that it will connect to the server that is described in the handler. This splits functionality over multiple classes such that it remains clean code. The interactor also can get devices, plugins and required info. The devices can also be changed and deleted from here such that it will also be changed accordingly on the server.

Technology Stack

Team Organisation

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.