rijksuniversiteit groningen

Hestia for iOS

Requirements 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: 7

2018

Introduction

With the introduction and advancements of technology in more and more aspects of our lives, a new area of interest has emerged, home automation. Because of the increasing interest in this area, many platforms have been created that promised to ease the control of the hardware in a home. Many manufacturers have their own platform for their products, an approach that is not desired for the end users. Just imagine the hassle of needing to switch systems each time you want to turn on the lights in the bathroom and kitchen, just because the smart light bulbs are not created by the same manufacturer. There are problems even on platforms which promise to function with a wide variety of peripherals.[1] The problem lies in the fact that the developer needs to manually update the platform in order to work with new hardware. This is a problem which hinders the regular user from using such platforms.

The Hestia system aims to solve this problem by creating a platform which is independent from the hardware used in home automation. The main idea is that the system is extendable and the user can add new peripherals without anyone needing to change the internal structure of the platform. The user can create their own plugins for the peripherals that they possess. Using these plugins, the Hestia system will be able to control everything and the user will never be required to change systems just because of different hardware.

The most important goal of this system is to create a stable platform which is extendable and makes it easy to control every smart device in a user's home. Any such device should be able to connect to our system. The possible peripherals that the system will support are the following, but these are not the only ones and many more could be included in the future: light bulbs, smart locks, refrigerators and heating systems.

The Hestia system is currently only available on Android devices. Since the iOS operating system has a considerable share in the smartphone market, our team is asked to develop an iOS application extending the existing Hestia system.

This requirements document will now continue with a short system overview and then our team will present you the functional and non-functional requirements. The requirements are categorized by their importance in the final project. A glossary can be found at the end of the document.

System Overview

The system that we are planning to implement already consists of the following components: the server side and the Android application. With our main goal in mind, which is to create a really versatile system to work with, we will also implement an application for iOS devices.

Any peripheral that will be added to the Hestia system must have the ability to be controlled by means of the Android application, iOS application and the Web Interface (which is also currently in development). Using the iOS application, the user should be greeted with a well

[1] http://fortune.com/2015/08/19/smart-home-stupid/

polished design, that follows the Apple's Human Interface Guidelines.[2] The application should also be really easy to operate, without being too cluttered with information. With the iOS application, the user should be able to control, install and remove peripherals.

The control, installation and removal of each of the devices will be sent to and managed by the server. All these operations will be done from the iOS application. Finally, once the information is sent to the server, it will apply and save all the changes that the user has made.

Critical Functional Requirements

- The application must be able to connect directly to a local server.
- The application must be able to get all devices on the server and show them to the user.
- The application must show the activators of a device and the state of these activators.
- The application must allow the user to set the activators on the server, such that an action is performed by the peripheral.
- The application must allow the user to add devices to the server using plugins, so that the user can interact with a peripheral.
- The application must allow the user to change the name of a device on the server.
- The application must allow the user to delete a device in the application. By doing so, the device should be deleted from the server.
- In case of connection to the Webserserver, the application must enable the user to log in safely with authentication (Auth0) to make sure there is no unauthorized access to the Webserver.
- The application must allow the user to change the server in case of local login.
- The application must remember if the user chose to connect to a local server or to the Webserver the last time he started the application, such that he does not have to select this each time he starts the application.
- The application must remember the login information and the entered server information, such that the user does not have to enter it all the time again when starting the application.
- The application must show informative messages if wrong information is entered, for example when adding a server or adding devices to the server.
- The application must enable the user to choose between connecting to a local server or login to the Webserver on first launch of the application.

Important Functional Requirements

- The application must present the same functionality when connecting to a local server through the Webserver as is the case when connecting directly to the local server.
- The application must allow the user to use voice control to interact with the application.
    - Use voice control to select local or global login.
    - Change the state of activators for a device.

[2] https://developer.apple.com/ios/human-interface-guidelines/overview/themes/

- The application must enable the user to select from which servers the devices should be presented in the main devices window (in case of Webserver login).
- The application must discover automatically local servers, such that the user only has to select the local server from a list and does not have to type the IP and port.
- In case of Webserver connection, the application must let the user decide on which local server a newly added device should be saved.
- The application provides information about the developers in the settings screen. (To be able to report bugs for example.)
- The main device screen supports pull-to-refresh to retrieve the device list from the server. This will be helpful if multiple clients are connected to the same server.
- The application must be able to display the additional information pertaining to the required info fields in the add devices screen.
- The application must allow the user to reset all user defaults (server information, last chosen login type etc.).

Useful Functional Requirements

- The application must offer the user the functionality to move to the next input field using the keyboard in case of multiple input fields in one screen. The action that should be performed after all fields are filled, should also be invokable from the keyboard (bottom right key). This is the case in the server connect screen.
- The application shows the devices from different servers separately in the main devices screen (in case of connection to the Webserver).
- The application shows informative information in the header above a group of devices on the server the devices belong to.
- The user should be able to add a new local server to the Webserver.

Won’t do

- Implementing different methods of connecting to the server, besides the internet protocols. (e.g. Bluetooth)
- The application must allow the user to change the IP address and port of a device on the server. (This boils down to deleting the device and adding a new one with the same name.)
- The list of devices shown in the application should contain a little icon or image to show the type of the device.
- The application supports different activators for the plugins besides switches (boolean) and sliders (float).
- The application must show other information of a device than its name to the user (IP and port for example).
- Scheduling actions such that they can be performed at a later time by the system. (e.g. switching lights off after midnight).
- Ability to set timers on certains actions, such that they only take a set amount of time. (e.g. keeping the light on for specific amount hours.

- Location based control of peripherals. (e.g. lights go out if the person leaves the house).
- The application lets the user search for a device by name.
- The application supports TouchID to login.

Critical Non-Functional Requirements

- The traditional iOS look and feel should be best maintained while also keeping similarities to the Android and web application.
    - Navigation buttons are consistently placed in the most suitable corner of the screen.
    - The application should have appropriate transitions between the screen, which are native to an iOS application.
- The user will be able to use the iOS application intuitively. Even if the user used to be an Android's user, the user will understand how the iOS application works.
- The builds are tested using continuous integration testing (appcenter).
- The application has an recognizable icon (image) which the user should tap to open the application.
- The application shows an appropriate launch screen (logo) when starting up the application.

Important Non-Functional Requirements

- The application does not have too many wasted space. This holds especially for the main devices screen.
    - The application should present the activators as pop up in the main devices screen. It also should the device name in the pop up screen.
- The back end code is unit tested.
- User Interface testing is performed on the front end.
- Clean code.
- Concise, visual yet clear documentation.
- The user information will be completely secure.

Useful Non-Functional Requirements

- Secure transmission of data between the application and server. (e.g. through encryption.)
- The back end unit testing has high code coverage.

Customer Meeting Log

When	What
February 23, 2018	Customer wants voice control capability of the application Customer wants a nice iOS look and feel design of the application. Functionality is not important first, but installing peripherals is primary
March 19, 2018	Customer did not like that all sliders are visible when a device is turned on. This would clutter the screen if a device has more than one slider. So we should change the design that the sliders are not always present when a device is switched on. He also did not like the categorization in device categories. Categorization by room could be an option, adding an icon to a device that indicate the type is sufficient.
April 19, 2018	Customer wants that the user can choose to connect to a local server or use Firebase. In the Firebase case the user should be able to select the server(s) he wants to connect to. The devices can then be shown bundled. He didn't like the layout much or he had to get used to it. He found that there was a lot of unused space. He suggested to represent the devices in boxes, just like Windows phone. In addition to changing the name of a device, editing port and IP can be added. The back end should be unit tested. We should do UI testing.
April 25, 2018 (First deployment on customer device, quick feedback on first look)	The customer wanted that also on the tap of the entire cell for adding a new device this function should occur, now this was only on tap of the green insertion icon. The customers wanted the credentials and server information to be remembered by the app.
May 3, 2018	The customer wanted an option to refresh the list of devices in the main window. The customer wanted the information about the required info field be shown in the add devices screen. The customer still found that there was too much wasted space in the application screens. The customer did not want any login anymore for the local case. The customer wanted that the application goes direct to the devices main screen if there was already information about a previous login.
May 9, 2018 (Short meeting with	The first activator should be shown together with the device name. The other activators should be shown in a popup window in the

customer to discuss the “wasted space” issue)	devices main screen.
May 17, 2018	Priority for next sprint should be voice recognition, either in the Siri way or just in-app. Siri is preferred, but probably more difficult. Finish local server discovery Finish UI Testing Better exception handling Fix remaining bugs (Scheduled action front end + define API calls)

Change Log

Who	When	Section	What
M. Fleurke	February 23 2018	Document	Created document + headings.
A. M. Oanca	February 24 2018	Document	Added extra headings. Finished paragraph Introduction. and System Overview. Modified design of the cover.
Z. Holwerda	February 25, 2018	Document	Merged and updated requirements. Adjustments of style all across. Added headings. Updated the introduction & overview.
D. Groot	February 25, 2018	Document	Improved formulation and fixed some grammatical errors.
M. Fleurke	February 26, 2018	Document	Added location based control requirement. Started glossary. Small grammar fixes.
A. M. Oanca, M. Fleurke, Z. Holwerda	February 26, 2018	Document	All issues resolved, requirements merged even more. Fixed consistency.
A. M. Oanca	February 27, 2018	Document	Some retouches for Introduction. Updated the logo.
M. Fleurke	February 27, 208	Introduction	Fixed/added the footnotes.
J.J. Méndez Torrero	March 13, 2018	Document	Updated the document with Feiko's notes.
D. Groot	March 13, 2018	Document	Minor text fixes.
M. Fleurke	April 28, 2018	Document	Major revision of the requirements following from the customer's feedback and the development process. Extended glossary.
M. Fleurke	May 10, 2018	Document	Updated and added requirements following the meetings with the customer on May 5 and 9.
M. Fleurke	May 29, 2018	Document	Updated priority of requirements

			and some additions and deletions. Added customer meeting May 19.

Glossary[3]

Peripheral
The real world object that is controlled by the application (via the server), for example a light bulb or a smart lock.

Plugin
The class containing all the methods and Activators to communicate with the peripherals.

Device
An instance of an Plugin that is added to the system and can be controlled through the Client.

Action
An action that is performed by the peripheral, for instance a light turning on or turning off.

Activator
Represents the action of the peripheral in the form of a state that can be changed. The action of a light turning on would for instance be represented by the activator going from the state False to True.

Client
The application that sends request to the Server. This can be the Android or iOS app, or the Web application.

Server
The local server on which information of peripherals is stored. The application interacts with peripherals through the server.

Webserver
The server that is implemented by the Hestia Web team. It functions as a bridge between the local servers and the iOS client. The iOS app should login to the Webserver and send requests via the Webserver to the local servers.

[3] From the 2017 Hestia architecture document, with modifications.