**Bright Light Systems**

**System Design Document**

**v. 1.0**

*by Michael Gulenko & Dallas Winger*

**INTRODUCTION:**

Bright Light Systems is an app for Android OS and iOS that allows users to take full control over the Philips Hue light bulbs by communicating user’s requests to a Hue Bridge which on its end will adjust the lights according to request.

The main set of features consist from following:

* Color/Brightness Change
* Color/Brightness Transitions Over Time
* Theme Organizer
* Scheduling (Alarm or Timer based)
* Location Based Actions (Auto On/Off)
* Message Notification (Text, Email, Facebook, etc)
* Music Mode
* Homescreen Widget Control
* Smart Suggestions

This document will describe high level system design specifications of this features as well as components’ interaction within the system.

# GOALS AND CONCERNS

There are several factors that dictate certain criteria for the design. The first factor is a marketing strategy. Bright Light Systems will be developed and released as a commercial product. The system provides a collection of different services, where individual price will be applied per different service. Before purchasing, users will have an opportunity to try a specific service for fixed period of time, and then will have to purchase it for further use.

Another factor is the development process and the future support of the product. The first release may or may not include certain services which can be developed and added later on. Thus, the design has to be flexible and clean, allowing developers to add new features and services in future while the system itself has to function independently from available services.

# Security

The Philips Hue system has built in measures for security that this app will have to comply with to function properly. Upon pairing to a Hue Bridge, the system will require the user to prove physical access to the system by pressing a button on the Hue Bridge; much like WPS technology implemented on many of today’s wireless routers. The main security goal for this project is to not allow unintended access to a Bridge or set of lights. Additionally any information this system stores (in the Database component) considered to be unique to a user, or that could be used to allow access when unintended, should be obfuscated (in the form of a hash or encryption algorithm).

# Battery life and Performance

Being an application for a mobile platform, this app should perform as efficiently as possible. On an “average” device this app shouldn’t exceed 15% of the battery usage. This is an important metric to look at because the feature set involved will require background services to listen for events. This will be the responsibility of the Service Manager component.

# SYSTEM COMPONENTS

The Bright Light system is divided on two main subsystems: **Core** and **Modules**.

The Core subsystem contains all necessary essential components to run the application and provide a set of core features (see BLS Core Features document for more details). The Modules subsystem, on other hand, can’t run independently. It contains components that describe additional unlockable features such as geofencing, alarms, bright light analyzer, music mode, and others. Detailed description of each component is provided below.

**CORE COMPONENTS:**

* Activity Stack
* Database
* Gateway
* Message Notifier

# Activity Stack

The Activity Stack will be responsible for all of the UI and its logic. An Activity in Android is considered to be: “a single, focused thing that the user can do.” An activity consists of a Java class and XML layout. The XML layout file is responsible for controlling the UI layout to be displayed to the user, while the Java class is responsible for all of the listeners and logic behind the Activity/UI. As Bright Light Systems will have more than one UI screen, it will require multiple activities; thankfully the Activity Stack itself, is managed by Android.

# Database

The databasecomponent is responsible for storing and retrieving information about light bulb settings, themes, events, and services. The data will be stored on the device, while access to the database will be provided through SQL interpreter that will be issuing SQLite queries.

# Gateway

The gateway component will be completely responsible for Philips’ Java Multi-Platform and Android SDK. To make the desired changes to lights connected to the Hue Bridge locally, Phillips has given developers a Hue Bridge API. This is a RESTful API over HTTP. This means that every controllable parameter has its own local URL to make the request to. All Hue Bridge responses return in JSON blobs (JavaScript Object Notation) with UTF8 encoding for easy parsing. This API requires direct access to your bridge so you’ll only be able to access it when your app and bridge are on the same local network. To access the bridge from outside of the network the user must interact with the portal (a web based control panel which connects your home to the internet). For this, a web view will frame the control panel to allow for this access.

# Message Notifier

This component is responsible for centralizing the internal communication within the **entire** system. Its main job to notify other components when a certain event(s) occurred with appropriate message that contains all necessary data to appropriately process that event.

**MODULE COMPONENTS:**

* Module Loader
* Module
* Service Manager

# Module Loader

This component is responsible for loading varies modules based on licensing and for providing an interface to use their feature set from the Core.

**Module**

This component represents a specific set of features that can be loaded during runtime by the Module Loader. The Module contains a logic, backend procedures and algorithms for those features as well as appropriate GUI that will be available for the Core upon loading.

# Service Manager

The service manager will handle tasks related to services such as start, abort and lock or unlock. The service manager will be responsible for retrieving data from the Android Service(s) running in the background; such services would include Geofencing, notification listeners, etc. It will be the job of the Service Manager to control whether or not that service is in use.

**INTERNAL COMMUNICATIONS**

The diagram 1 describes communication and logical data flow between components.



Diagram 1.

Shows bidirectional external communication.

Internal request to post a message

Internal notification about posted message

Module is unlocked

Module is locked