# **Software Object-Oriented Design**

*Help Me! Laurier*

Vaughan Hilts (120892740)   
Brandon Smith (120201510)   
Colin Gidzinski (120663460)

***Group #1***

***Dr. Chinh Hoang***

Table of Contents

[Introduction 3](#_Toc403600060)

[Purpose 3](#_Toc403600061)

[Overview 3](#_Toc403600062)

[System Overview 3](#_Toc403600063)

[System Architecture 4](#_Toc403600064)

[Data Design 4](#_Toc403600065)

[Component Design 4](#_Toc403600066)

[Interface Design 4](#_Toc403600067)

[Further Reading / Technology Glossary 5](#_Toc403600068)

[System Overview 6](#_Toc403600069)

[Client Overview 6](#_Toc403600070)

[Server Overview 8](#_Toc403600071)

[System Architecture 10](#_Toc403600072)

[High Level Overview 10](#_Toc403600073)

[Detailed Subsystem Analysis 11](#_Toc403600074)

[Interface Design 12](#_Toc403600075)

# Introduction

## Purpose

The following document is intended to give developers a better insight as to how the software will be developed from a technical perspective. Things like architecture, system design, component design, and all technical aspects of the software will be discussed. The documentation will strive to give a complete overview and ensure every developer is on the same page during the construction of the software product, “Help Me! Laurier”.

It should be noted that because of this, the documentation is **very technical.** If you are looking for a more broad requirements or introduction to the application, please consider reading the *Requirements* or *Analysis* sections first.

## Overview

To get a general overview of the document, see the table of contents provided. For a more in-depth look at what each section will contain, check out the below descriptions:

### System Overview

The system overview will give a general description of the functionality of the application. This will include things like: client-end choices, back-end choices, technology stack descriptions, database technology choices and a general overview of them. This section is designed so that a new developer can read it and figure out what their technical skill set will need to look like to contribute to the project successfully.

While the section will not outline every library and piece of software used throughout the system, it will describe all the major components. For example, if a specific server-sided library is used for a small task such as time zone calculations, it may not be mentioned. Critical components such as **AngularJS** will be discussed as they are mandatory to understanding the bulk of the application.

A brief description of why certain things were chosen will also be discussed along with other technologies that were considered. This will allow other developers to gain insight on how previous developers thought and why they had made the choices they had.

### System Architecture

The system architecture will provide a more detailed look at all the different subsystems and components of the application. Here, you will find decompositions of how each subsystem will interact, function and provide services to the other systems in the application. This gives a high level technical overview of the different systems in the application. If you are interested in how everything works from an overview perspective, this is the place to check out first.

In this section, you will also find some rationale for the choices that were made. This allows a section to document alternatives and notes on them. If a particular architectural decision is not going well, it is important to be able to reflect.

### Data Design

In this section, we discuss the overview of all data that flows in and out of the application. You will find a listing of all the data the applications needs to manage, how it is managed, where it is stored, processed and organized. If you want to know the specifics on a specific piece of data or how it interacts with the application, check here.

### Component Design

In this section, objects will be looked at it in a more individual level. The information from the system architecture overview will be analyzed in a more granular and specific way. Check this section out for brief descriptions on programming UMLs, member function descriptions, and descriptive code analysis.

### Interface Design

In this section, the user interface will be briefly laid out in the form of mockups and brief descriptions. This will help give a more general idea of how the application should function from a user experience standpoint. This is important to the front end developers, so that they can properly capture the essence and vision of the UX team. This section will try to outline each screen briefly and label what each button should do, mapping it to use a case or action.

## Further Reading / Technology Glossary

Before reading this document, it may be useful to list a few technical terms and information on where to read more about them as they are referred to throughout the documentation.

**Apache Cordova:** Apache Cordova is a set of tools designed to allow developers to package their web apps into native, easy to use packages to be distributed on various phone and tablet devices easily. (Read more: <http://cordova.apache.org/>)

**AngularJS:** AngularJS is a client side framework that makes writing MVVW like applications easy with JavaScript. (Read more: <https://angularjs.org/>)

**Ionic Framework:** The Ionic Framework is a high performance framework based on *AngularJS* and *Cordova* to create like-native applications using the greatest web application. (Read more: <http://ionicframework.com/>)

**SASS:** SASS is an extension of the CSS standard which describes some added in functionality, such as constants, expressions, and fallbacks for newer specifications. (Read more: <http://sass-lang.com/>)

**Bower:** A client sided package manager for libraries and tools. It used throughout the application to manage dependencies and ensure all developers are up to date. (Read more: <http://bower.io/>)

**npm:** The Node Package Manager (npm) is a tool for importing libraries and module into a node.js project. It is similar to Bower but is mainly used to manage the back-end of the software, rather than the client side. (Read more: <https://www.npmjs.org/>)

**V8:** V8 is Google’s implementation of the ECMAScript JavaScript specification. It is a fast, optimized version of the specification designed with performance in mind. (Read more: <https://developers.google.com/v8/>)

**Node.js:** Node.js is a sever side implementation of JavaScript using the V8 JavaScript engine to interpret. (Read more: <http://nodejs.org/>)

**MongoDB:** MongoDB is a document-oriented database that allows storage of data via blobs and “documents”, unlike traditional relational database management systems. It is a popular choice among many *Node.js* developers. (Read more: <http://www.mongodb.org/>)

**nodemon:** A build monitoring tool for *Node.js* that assists in rapid builds. (Read more: <http://nodemon.io/>)

**Gulp:** Gulp is a streaming build tool used to simplify the process of developing JavaScript applications. Similar to *make* in **C** and other build tools, *Gulp* can manage compiling, minifying, packaging, compressing images, building native applications with ***Cordova***and more. (Read more: <http://gulpjs.com/>)

**REST API:** A REST API is a specific API that provides CRUD operations at HTTP endpoints.

**Sails.js, Express:** A structured web application server provided that makes creating REST APIs a snap. (Read more: <http://sailsjs.org>

# System Overview

*Help Me! Laurier* is an Android application that runs on the newest bleeding edge technology to make use of the latest and greatest software available, to make the best applications possible. To do this, powerful frameworks like **AngularJS and Ionic Framework** are used on the client side. Powered by powerful HTML5, they allow the rapid development and flexibility required to build our application. For brains on the server side, we have chosen things like the upcoming **V8** powered, **node.js and npm.** With these technologies, we aim to utilize all the tools available to the full potential. Below, we will describe the rationale, other choices considered and how we intend to use most of the major technologies.

## Client Overview

On the client side, our choice of AngularJS, Ionic Framework and Cordova enables us to work with some of the best technology from the best companies. Google develops, and therefore has stake, in Angular JS which our team has experience with. Thus, for the construction of this specific product we would like to make use of Angular. Angular provides data binding, routing, view management, resource management, controller management, and scaffolding to help develop. It makes the most sense for our specific application.

Ionic Framework is another layer on top of Angular, specifically targeted towards making great looking, easy to develop mobile applications built with Angular. As our team also has experience with this and is built top down to be integrated 100% with Angular, it was our logical choice. The set of controls and power provided to map UI to a data contract model in a few lines of code made it ideal for creating an application fast.

Cordova, Bower, and Gulp are used as part of the build process to ensure a smooth development process. Gulp is one of the few JavaScript build tools available, with the alternative being Grunt or Yeoman. Ionic uses Gulp, so it was our logical choice. Bower is the only reasonable option for client-side dependency management, so it is also the clear choice. Cordova is described in more detail below.

Other options have been considered before settling. However, all of them have pitfalls that were answered by Ionic and Angular. These are listed below with a brief description of why they were not used:

* Pure JavaScript
  + Development with no frameworks is cumbersome and eats up time for no reason. In the software industry today, when writing simple CRUD applications like ours – it is only logical to leverage all existing support ecosystems.
* jQuery Mobile
  + jQuery Mobile is great and provides an excellent set of controls and intuitive building blocks to great powerful applications. However, jQuery encourages mixing in view and model code with the selector pattern it has imposed for many years. **Ionic** has all the great things about jQuery Mobile – wrapped in an Angular friendly wrapper.
* Bootstrap
  + This is a great CSS library we considered but it simply has no bindings available for quick prototyping and our team has little experience with it. The learning curve would exceed the scope of the project.
* Sencha Touch
  + Sencha is incredibly popular, similarly to Cordova. All team members are familiar with Cordova, however. Cordova is hence the better choice as it meets all the requirements for this project.
* Kendo UI
  + Kendo has restrictive licensing which would make it difficult to open-source our project, a potential fate for this project in the future. In order to future proof ourselves, this option had to be ruled out.
* Grunt
  + An excellent package manager that was considered – but Ionic integrates perfectly with Gulp, so it makes sense to use what is compatible with our toolset rather than work against it.

Other technologies may have also been considered but were decided to not be of high enough relevance to be list above.

## Server Overview

When picking technology for our server stack, we decided there a few main requirements:

* Every developer had to have knowledge in the language it was going to be written in
* It had to have the ability to provide a REST API easily
* It had to run on UNIX systems (this ruled out C# / .NET)
* Having a low entry barrier for new developers was a must

With these requirements in mind, an analysis was performed and the JavaScript powered Node.js was selected for a variety of reasons. First and foremost, it uses JavaScript. As our client end is written in JavaScript, every developer will need to know this. This will make every developer able to work on both code bases with some degree of skill. Node.js runs on almost every platform and has frameworks like *Express* to provide REST APIs. Best of all, Node.js is a rising popular platform: support is everywhere and is only growing.

For storage, we had a few options. **MongoDB** is by far the most popular choice when developing with Node.js which is ultimately why it was chosen for this project. The document oriented hierarchy allows us to store massive amounts of question and answer information without a traditional relational database. However, these were considered as well. MySQL and traditional databases are unfortunately second class citizens in the context of the Node.js community. For this reason, they were quickly discarded.

Express and Sails.js are being used to give a structured feeling throughout the entire application lifecycle and prevents writing a lot of boilerplate code. They are industry hardened and have been used by big name companies like Facebook. The alternative was writing all this boilerplate from scratch in JavaScript, which is a major problem as it violates one of the major rationale we used for selecting libraries previously.

**Node.js** and the powerful **npm** were selected, but we also considered a few other possibilities based on their merits. Ultimately, we considered but rejected:

* **C# & ASP.NET**
  + C# is a powerful language that produces a great REST API with little effort, great ORMs (such as **Entity Framework**) and a wide tool service. Unfortunately, they are not multi-platform and our team is not well versed with it.
* **Ruby on Rails**
  + Nobody on the team has any experience with Ruby, even though the technology looked great and similar products such as Disqus has been developed under it.
* **Java / Tomcat**
  + This is usually reserved for enterprise companies and has a lot of boilerplate required to get started. The barrier of entry is too high here to consider using it for our current development team and expected team.
* **MySQL**
  + For reasons outlined above in detail, MySQL did not fit well within the paradigm we were aiming to use. For this reason, it was culled immediately without much second thought.

With everything we selected, we made sure it added significant value to our project. Each framework, tool, and library has been selected with extreme care as the architecture will be designed around them. Care will be taken to abstract the details around them, however.

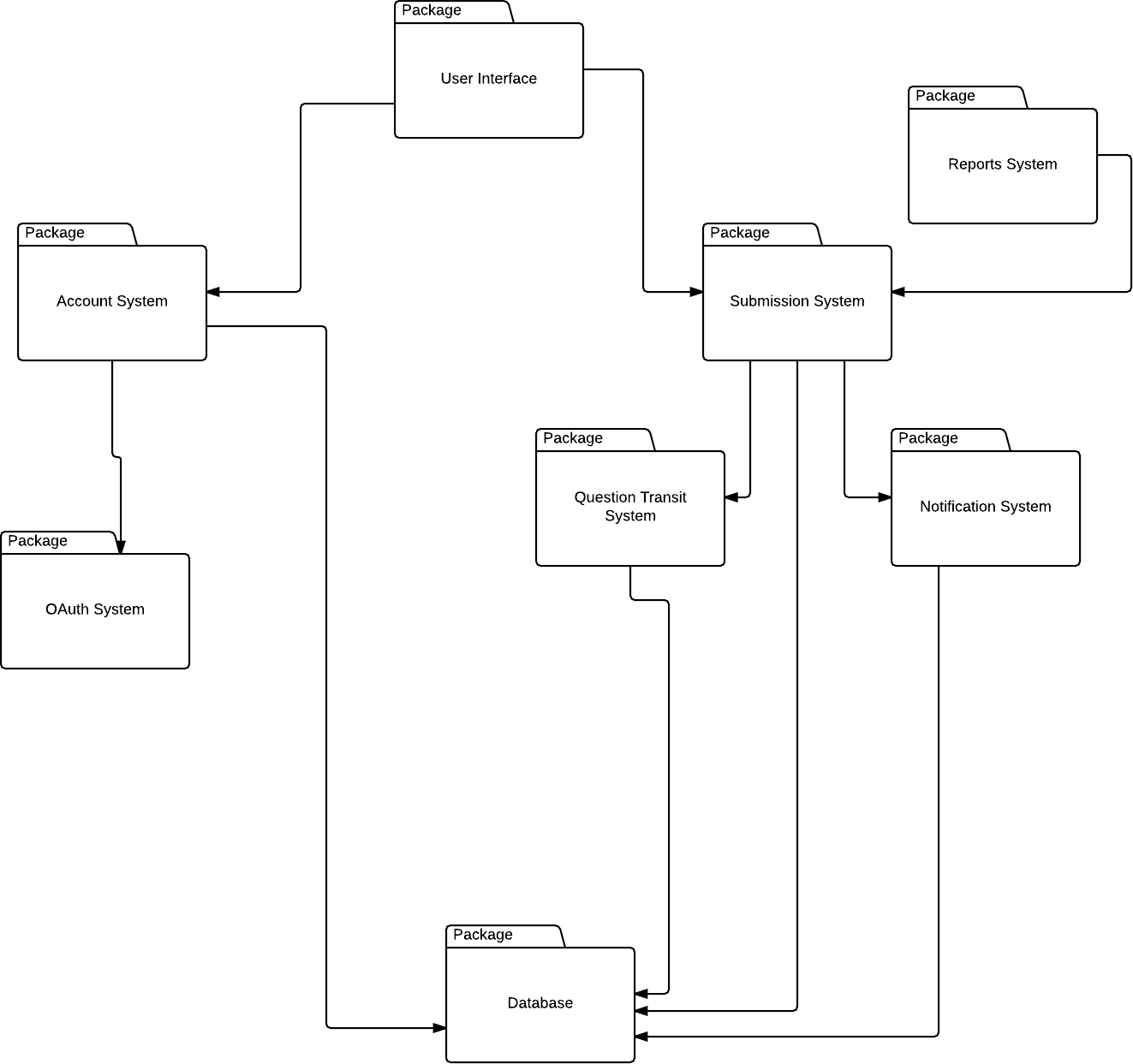
# System Architecture

## High Level Overview

Below, we will describe the system architecture by listing the relationships between the major subsystems involved in the application. We decompose them into a few major systems:

1. User Interface
2. Account System
3. OAuth System
4. Submission System
5. Question Transit System
6. Notification System
7. Reports System
8. Database System (I/O)

These major subsystems make up the bulk of the work for the application. To get a better look at how they interact, please refer to the below system architecture diagram that illustrates dependencies:



The diagram should stand by itself, but a brief description is given below for completeness sake.

The eight major subsystems in use are represented by the different packages above. At the root of everything is the **User Interface.** This interacts with all the subsystems indirectly as the user will need to interact before any of the other systems can react. Most of the systems that do actual work will have a relationship with the database. The **Submission System** will contain core services regarding questions and answers, while the **Account System** will describe authentication related tasks. Each subsystem below is separated out services that are lent to these two major systems. Their details are important but will be described in much more detail in the next few sections.

## Detailed Subsystem Analysis

# Interface Design