# **Software Requirements Specification**

*Help Me! Laurier*

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

# Table of Contents

[Table of Contents 2](#_Toc402442522)

[Introduction 4](#_Toc402442523)

[Purpose 4](#_Toc402442524)

[Scope 4](#_Toc402442525)

[Terms, Acronyms and Glossary 5](#_Toc402442526)

[Overall Description 5](#_Toc402442527)

[Perspective 5](#_Toc402442528)

[Constraints 6](#_Toc402442529)

[Specific Functional Requirements 6](#_Toc402442530)

[Response Time 6](#_Toc402442531)

[Anonymity 6](#_Toc402442532)

[Use Cases of the Application 7](#_Toc402442533)

[Scenario: Asking a question 7](#_Toc402442534)

[Scenario: Answer a question 8](#_Toc402442535)

[Scenario: Check a question 8](#_Toc402442536)

[Scenario: Login and setup 9](#_Toc402442537)

[Scenario: Check notifications 9](#_Toc402442538)

[Scenario: Distribute a question 10](#_Toc402442539)

[Scenario: Push an answer 10](#_Toc402442540)

[Scenario: Update profile 11](#_Toc402442541)

[Scenario: Reporting a question 11](#_Toc402442542)

[Scenario: Connection Down 12](#_Toc402442543)

[Use Case Diagram 13](#_Toc402442544)

[Team Organization and Time Management 14](#_Toc402442545)

[Development Organization 14](#_Toc402442546)

[Timeline Management 15](#_Toc402442547)

[Skill Matrix 16](#_Toc402442548)

# Introduction

## Purpose

The purpose of this document is to give detailed requirements for our application “Help Me! Laurier” to give a broad overall understanding of how our application will work before creating the application. It will show how the application will function as a whole with an overarching overview. This document will be used to evaluate the feasibility and allow outside clients and customers to audio the desirability of our application before proceeding to build it.

## Scope

“Help Me! Laurier” is an internet connected application that runs on Android smartphones that connects students with knowledgeable peers to get them the answers they need. This include academics, financial information, extracurricular activities, and student recommendations (restaurants, entertainment, etc.). Students have the option of being anonymous to contribute their questions and knowledge while retaining the right to protect their privacy. “Help Me! Laurier” deploys a distributed approach for getting the answers students need, quickly.

## Terms, Acronyms and Glossary

**User, Client:** A user interacting with our mobile application contributing questions and knowledge to the bank of data.

**Administrator:** A backend super user that manages the bank of questions, monitors for spam and collects metrics about our system.

**Front-End:** A set of views and interface that a user will interact with, in this case the mobile application.

**WiFi:** A wireless signal protocol used to connect to the internet

**Push Notification Server:** A service provided by Google to send information to a user’s device as it happens, similar to text message notifications, e-mail notifications, or notifications about your social media.

**Push Oriented:** A term used to describe data which is sent to users when required rather than on demand

**Pull Oriented:** A term used to describe data which is sent to a user when they request it.

**OAuth:** A process used to authenticate with open providers such as Google. Used in the application to login using student credentials.

**ApplicationServer:** This term is sometimes used to refer to the server component of the application, especially in use cases.

# Overall Description

## Perspective

“Help Me! Laurier” is a networked application, so the system will consist of a mobile application and a backend server that the mobile application will connect to. The actual front-end application is where the users will spend all of their time – the server simply exists to facilitate communication between the students. Users will use this front-end to view the questions of other students, receive notifications about topics they may be interested in and submit questions for other students.

The only thing required by the mobile application is a strong wireless data signal (or “WiFi”) in which the application can use to periodically use to communicate with the server to retrieve question information and submit question content. Occasionally, users will receive updates from the push notification server regarding recent application activity. This functionality is all contained within the single application, so there is no need for the user to do anything outside of it.

The server side application is responsible for storing and categorizing all the content in a performant way. The server will be an intelligent program capable of crunching information, creating related question links, and sending out notifications. It is responsible for delegating all content to the appropriate channels.

On the server side, we realize our product is very content-heavy and requires a strong back-end as we are very push oriented versus pull (although, this is supported, too). This means our back end will utilize a strong server capable of storing a lot of text data in an efficient manner. Data will be indexed, timestamped and stored for easy archiving and storage. The mobile application will not interact with this database, only though the server.

## Constraints

**Lack of network connection must be handled.** The application relies on a network connection, so the application must be able to operate offline and recover when reconnected at an arbitrary time.

**Intelligence about pushing.** The selling point of the application is that it can intelligently display information to people who are interested and get the users in front of the right people. To do this without causing annoyance, the application must be smart and accurate. The application must be able to detect inactivity and find the right people for the job without fail.

## Specific Functional Requirements

### Response Time

The application must be able to send out notifications quickly and process fast. Long running algorithms to decide where content will be passed around are not acceptable. When a question is submitted, it must be determined who can potentially answer within **one minute.**

### Anonymity

This does not affect the user experience but it must be noted that while anonymity is kept where possible, users that could be identified as a threat to the system must be able to be identified. *This means the server should keep track of who posted and answered what, in case of court order or authority.*

# Use Cases of the Application

## Scenario: Asking a question

**Description:** The user has thought of a question they want to ask the student peer advisors and wants to use the application to ask a question. They have loaded the application and have arrived at the main menu screen.

**Actor(s)**: CuriousUser, ApplicationServer

**Flow:**

1. Tap the button on the screen that indicates “ask a question”
2. User is prompted for a question type consisting of: academic, school related and social categories.
3. Depending on the selection, the user will be presented with more sub types
   1. Selecting “Academic” presents a list of courses and the option for a user to filter and select one.
   2. Selecting “School Related” presents a list of possible subcategories such as “Financial” or “Organizations”
   3. Selecting “Social” presents a list of school clubs and other recommendations.
4. Once the student has entered a sub category it prompts them to ask a question. While typing, they will be presented with some potential similar questions.
   1. If they select one of these similar questions, the use case will be aborted and use case **“Check an answer”** will be initiated.
   2. Otherwise, the user will continue to type their question and flow will continue.
5. After the user has entered their question, they may select “Anonymous” which will hide their identity from the question before selecting **submit**.
   1. Even if a user selects anonymous, user information will be recorded for safety and legal reasons.
6. At this point, the user will be notified that their question has been submitted successfully. They will be returned to the main menu.
7. The **ApplicationServer** actor will be notified of this submission and the use case “**Distribute a question”** will be activated.

## Scenario: Answer a question

**Description:** A user is currently viewing a question in the application and has decided to answer it. They will be answering within the application.

**Actor(s):** User, ApplicationServer

**Flow:**

1. If a student chooses to respond to a question, they may select “Answer”
2. The user will be presented with a textbook and some minor editing utilities, like the ability to attach a picture.
3. After entering their response, the user will optionally select “Answer Anonymously” and then touch “Submit”
   1. Even if a user selects anonymous, user information will be recorded for safety and legal reasons.
4. At this point, the user will be notified that their answer has been submitted successfully and they will be returned to the question.
5. The **ApplicationServer** actor will be notified of this submission and then the use case “**Push an answer”** will be activated.

## Scenario: Check a question

**Description:** A user has initiated an action to view a question – either from a notification of some other part of the application that has invoked this use case.

**Actor(s):** User, ApplicationServer

**Flow:**

1. The user is presented with the question title, description and a list of answered, in the order they have arrived by default; then sorted by satisfaction
   1. Each question has two buttons: “**Satisfactory”** or “**Unsatisfactory”**
      1. If the student presses satisfactory a push is sent to database to move the answer up.
      2. If the student presses unsatisfactory a push is sent to database to move the answer down.
2. The user may be prompted with other options for the question, such as “report” or other administrative utilities.

## Scenario: Login and setup

**Description:** A user has just downloaded and installed the application and wants to get started. This use case is invoked the first time the user launches the application and has not yet completed the entire use case flow.

**Actor(s):** User, ApplicationServer

**Flow:**

1. The user will be presented with a “Welcome Screen” that will display some brief information about the application and how to get started.
2. After dismissing the above welcome, the user will be prompted for a **Google Account Login.**
3. The user will select their ”MyLaurier” e-mail and authenticate using Google OAuth
   1. If the user selects and authenticates with a non-Laurier e-mail, they will be asked to repeat this step.
   2. Similarly, invalid credentials will result in repeating this step until correct.
4. Upon successful authentication, the user will be moved to the main menu and the use case terminates.

## Scenario: Check notifications

**Description:** A user wishes to view their notifications to see what has changed since they last used the application.

**Actor(s):** User

**Flow:**

1. From the main menu, the user will select “Notifications” to be brought into the Notifications window.
2. From here, the user will be given a list of notifications they can view. They’re divided into several groups:
   1. “Answers for you” is a heading which will contain notifications regarding things you may have asked
   2. “Questions for you” heading containing notifications about questions that the user should be able to answer with their experience
3. A user may select either type of notification to be brought to the linked page – in either case selecting a notification will invoke the use case “**Check a question**” for the appropriate question page.

## Scenario: Distribute a question

**Description:** The server has received a question from a source and needs to distribute it to users who may be able to answer it.

**Actor(s):** ApplicationServer

**Flow:**

1. The server has received the question and is ready to distribute it.
2. If it has not yet recorded the question, it will connect to the database and record it.
3. A subset of users will be selected that are most likely able to answer this question based on a number of factors, including but not limited to: experience, answer history, satisfaction, and initial setup.
   1. If a subset cannot be found, the use case terminates immediately.
4. The selected subset will receive a push notification which when clicked will invoke the use case **“Check Notifications”**
5. A timer will be set for a later interval to re-execute this use case for the same question If an answer has not been received in the given interval. A different subset will be scheduled for this time to avoid double push notifications.

## Scenario: Push an answer

**Description:** The server has received an answer from a source and needs to distribute it the user who has asked.

**Actor(s):** ApplicationServer

**Flow:**

1. The server has received the answer and is ready to distribute it.
2. The question in which the answer belongs to will be looked up and then the answer attached and recorded in the database.
3. The user who asked the initial question will then be sent a push notification, alerting them their question has a new answer.

## Scenario: Update profile

**Description:** The user wishes to get more accurate answer candidates and has decided to update their profile to improve the accuracy of the application.

**Actor(s):** User**,** ApplicationServer

**Flow:**

1. The user will be presented with some options and various headings. Such things could be:
   1. **Academic Information:** This contains information regarding courses, your expertise level and comfort level, and enrolment status
   2. **Organization Affiliation:** This contains questions about your organizational affiliations
   3. **Extracurricular Affiliations**
2. The user will select one of these and then proceed to answer questions about their various associations. Exact layouts can be found in preceding documents.
3. When a user is satisfied with their changes, they can click “Submit” to have them persisted to the server and reflected in all stages of the application immediately.

## Scenario: Reporting a question

**Description:** The user wishes to report a question for being outdated, old, or inappropriate (spam) or abuse.

**Actor(s):** User**,** ApplicationServer

**Flow:**

1. The user will click the “Report” bottom from the “Check a question” screen and see options for reporting the question and/or various answers.
2. When the user selects this option, they will be prompted with reasons as to why this question needs to be reported:
   1. Dealing with abusive material
   2. Spam
   3. Outdated information
3. After the user identifies the reason, they click submit and the administrators will be notified to take action accordingly.
4. The user returns to the question view.

## Scenario: Connection Down

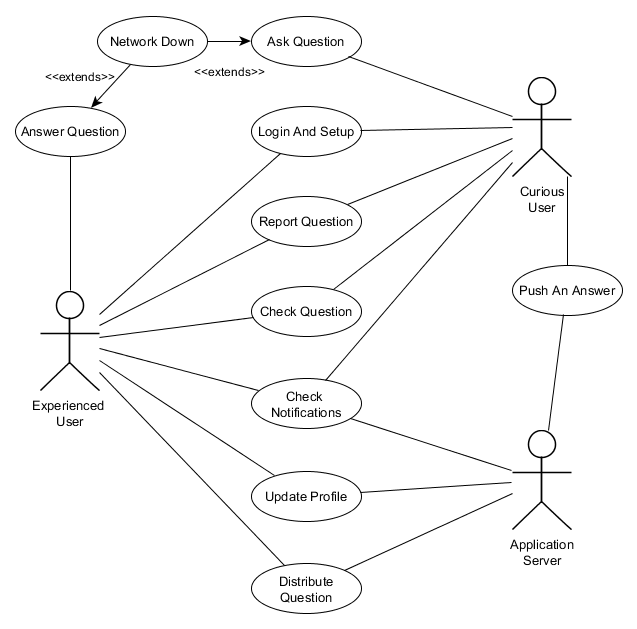
**Description:** A user is using the application when the network connection is lost or the **ApplicationServer** could not be reached. The user is then presented with a notification informing them of the connection status and that there attempt has failed

**Actor(s):** User, ApplicationServer

**Flow:**

1. The student attempts to retrieve information or notifications from the **ApplicationServer**.
2. The user is not connected to the network or cannot reach the server due to a connectivity issue.
3. The user will be presented with a toast/notification informing them that a connection could not be made and an error code describing the situation in more detail.

## Use Case Diagram



# Team Organization and Time Management

## Development Organization

The proposed method for developing the software is a model that is similar to the popular *Waterfall Method.* We will be developing requirements, specifications, design and similar documents which will be signed off by a separate party, our research professor. However, due to the scope of the product the formal SQA team will be our *professor.* After each step, we will have the documents verified before proceeding to the next step to ensure our software is as stable and well-engineered as it can be. This will involve revision before proceeding to the next step, after each step.

Each step will be as a group to maximize the talent of each member as each of us is well rounded individuals with exposure to a lot of different subsets of technical skills. Maximizing them through a potential “phase leadership” system will be difficult, so the team will be using bi-weekly (read: twice a week) meetings to organize information and sync up accordingly. By doing this, the product will be well known by each developer while maximizing our productivity output.

## Timeline Management

All developers will meet bi-weekly, on Tuesdays and Thursdays throughout the project from 1PM – 4PM. It should be noted additional individual work will sometimes be completed to optimize the quality and clarity of documentation, especially during the implementation phases.

|  |  |
| --- | --- |
| Tuesday | Thursday |
| October 28th: Begin development of the requirements as a group. Create concrete use cases, skill matrix, create timeline. | October 30th: Polished requirements and agree on any ambiguous conditions and prepare final revision of requirements documentation. |
| November 4th: Meet about preliminary analysis and begin working on. Finish during time this time. It may extend past the normal hours. | **No meeting** |
| November 11th: Meet about design and begin fleshing out CRCs. | **November 13th**: Polish design report; prepare to have a final good idea of the architecture of the application. |
| November 18th: Work on completed report and finish putting it together. Complete some work on implementation. | **November 20th:** Ensure completed report is ready. Complete more work on implementation. |
| No meeting | **November 27th**: Begin wrapping up the implementation of the project. Perform final testing, ensure stability of software |

I, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ agree to the above timeline.

Signature

I, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ agree to the above timeline.

Signature

I, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ agree to the above timeline.

Signature

## Skill Matrix

The following matrix breaks down the skill levels and interest of our developers, to help organize our team better.

|  |  |  |  |
| --- | --- | --- | --- |
| Task & Participants | Vaughan Hilts | Brandon Smith | Colin Gidzinksi |
| User Experience Design / UI (including some graphics design) | * Interested * Secondary Experience | * Interested |  |
| Database Schema Design & Support | * Interested * Secondary Experience |  | * Secondary Experience |
| Configuration Management |  |  |  |
| Client Side Frameworks & Implementations | * Primary Experience | * Interested | * Interested * Primary Experience |
| Server Side Frameworks & Implementations | * Interested * Secondary Experience |  | * Interested * Primary Experience |
| Security, Authentication and Auditing |  |  | * Interested * Secondary Experience |
| Software Testing | * Interested * Primary Experience (formal training) | * Interested |  |
| Documentation |  | * Interested |  |
| Communications (reporting) | * Interested |  |  |
| Software Architecture Design | * Interested |  |  |