Technical Design Document

**Site Design Proposal**

**Team 2**

Shopping List Project

# Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Description** | **Version History** |
| 2016/05/15 | Jared Wooten | Initial Creation | 1.0 |
| 2016/11/29 | Team 2 | Modified version | 2.2 |
|  |  |  |  |

# Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| SOW | Statement of Work |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# References

1. SOW
2. Project Plan.xls (Google Docs)
3. Estimation.xls (Google Docs)

# Table of Contents

[1 Version History](#_gjdgxs)

[2 Acronyms](#_30j0zll)

[3 References](#_1fob9te)

[4 Table of Contents](#_2et92p0)

[5 Introduction](#_tyjcwt)

[6 Scope](#_3dy6vkm)

[7 Technical Architecture](#_1t3h5sf)

[7.1 Use Cases](#_4d34og8)

[8 Technical Design](#_2s8eyo1)

[8.1 Design Constraints](#_17dp8vu)

[8.2 Data Model](#_3rdcrjn)

[8.3 User Interface](#_26in1rg)

[9 Platform Specific Technical Information](#_lnxbz9)

# Introduction

The purpose of this document is to expand upon the SOW [1] and define the technical architecture and high level design required to fulfill project scope. Anything not included in this document is considered out of scope and may incur additional time and costs.

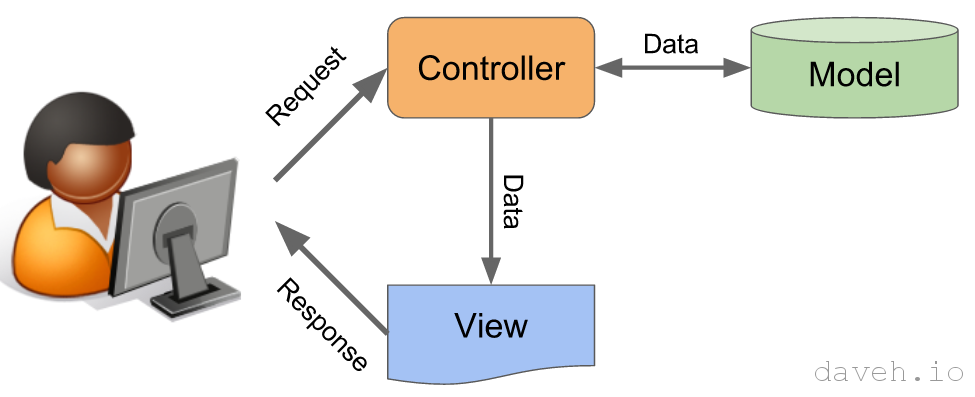
Eleven Fifty Academy (EFA) chose Team 2 because of its history of successful implementations of complex business solutions. The project will create a shopping list application for multiple users with a beautiful design and flawless functionality. Within two weeks, Team 2 will create this application with the ability for multiple users to create and edit lists and list items in a web interface with features specified in the SOW.

# Scope

This document is structured as follows:

* Technical Architecture
* Technical Design
* Platform Specific Technical Information

# Technical Architecture



**MVC:** We will be using the model view controller design pattern, which is designed to separate the user view’s view (site interface) from the model’s business/domain logic (domain logic is a more general term, but it encompasses what we normally refer to as business logic).

**Model:** The model represents the underlying, logical structure of data in the application and the high-level classes associated with it. In other words, the model handles the business and/or domain logic that is built with the Java code we have written and also includes the data we use within our program. A domain model is generally implemented as an object model (class, bean, POJO) within a layer that uses a lower-level layer for persistence and "publishes" an API to a higher-level layer to gain access to the data and behavior of the model.

The model will include domain objects, which typically consist of the classes we build, including Java beans and POJOs.

The model will typically include the databases and schema we build to interact with the interface. This model does not interact directly with the user interface, and is actually completely ignorant of it.

**Database:** This component is the database where backend data will be stored. The application will be developed initially using an H2 database for testing purposes, and then migrated to the chosen external database provider.

**Services**: This component is the assembly that encapsulates the business logic of the app. Domain services are typically written to “do stuff” with the data. This will include the services file(s) we build, such as EmailService.java.

**View:** This is the user interfaces that can be seen by the user. Typically, they are HTML files, and can even be forms that can be updated by the user. The updates do not occur in the view, rather changes are handled by the controllers we build.

**Controller:** A Controller represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

**UI:** Encompassed by ‘view’ in the MVC model, this component is the main UI of the app as defined by the MVC web application articulated above. his typically handled by html files seen by the user. See section 8.3 User on Interface.

**TLDR:** MVC is a way to separate the domain or business code from the presentation code. The Model does all the real business work. The views provide the look and feel, and the controller maps one to the other.

## Use Cases

The application will be written in Java using the Spring Boot MVC framework and Thymeleaf template engine and deployed to Heroku or a similar hosting service. The data will be stored in ClearDB database. Unit testing will be performed manually. The front end will implement HTML, CSS, and possibly JavaScript.

### Login Page

Users should be able to log in with a username (email address) and password (encrypted).

### Shopping List CRUD Pages

Users should be able to create, read, update, delete, sort, and check off lists and list items.

# Technical Design

## Design Constraints

The application will be compatible with Chrome browsers. The interface will be responsive so that users on mobile devices have a good experience.

## Data Model

The data model will use a code-first strategy with data migrations enabled to facilitate ease of deployment and incremental database development. Code first strategy means our development is both bottom-up and implementation-first. We first write the Java classes and the data POJOs that represent objects and data types used by the web services. We then use these to generate the web service’s interface and associated schema types.

### Identity Management

Role-based Spring Security with encrypted passwords.

### User

|  |  |  |
| --- | --- | --- |
| Id (key) | Int | Unique Id |
| Email | String | User’s email address/username |
| Password | String | User’s password (encrypted) |
| fullName | String | User’s first and last name |
| Active | Boolean | Whether user is active or inactive |
| CreatedUtc | DateTimeOffset | Utc created date |
| ModifiedUtc | DateTimeOffset | Utc modified date |

### UserRole

|  |  |  |
| --- | --- | --- |
| Id (key) | Int | Unique Id |
| UserId (foreign key) | Int | Relates to a user |
| Role | String | ADMIN or USER |
| CreatedUtc | DateTimeOffset | Utc created date |
| ModifiedUtc | DateTimeOffset | Utc modified date |

### ShoppingList

|  |  |  |
| --- | --- | --- |
| Id (key) | Int | Unique Id |
| UserId (foreign key) | Int | Relates to a User |
| Name | String | Name of shopping list |
| Color | String | Color to display as (hex value) |
| ShoppingListGroupId (foreign key) | Int | Relates to a shopping list group |
| CreatedUtc | DateTimeOffset | Utc created date |
| ModifiedUtc | DateTimeOffset | Utc modified date |

### ShoppingListGroup

|  |  |  |
| --- | --- | --- |
| Id (key) | Int | Unique Id |
| Name | String | Name of list group |
| CreatedUtc | DateTimeOffset | Utc created date |
| ModifiedUtc | DateTimeOffset | Utc modified date |

### ShoppingListItem

|  |  |  |
| --- | --- | --- |
| Id (key) | Int | Unique Id |
| ShoppingListId (foreign key) | Int | Relates to a ShoppingList |
| Name | String | Description of item to shop for |
| ShoppingListItemPriorityId | Int | One of several enumerated values (relates to ShoppingListItemPriority) |
| IsChecked | Bool | Checked items are purchased/complete |
| ShoppingListItemNoteId (foreign key) | Int | Relates to ShoppingListItemNote |
| CreatedUtc | DateTimeOffset | Utc created date |
| ModifiedUtc | DateTimeOffset | Utc modified date |

### ShoppingListItemPriority

|  |  |  |
| --- | --- | --- |
| Id (key) | Int | Unique Id |
| Name | String | Description of the priority (“It can wait”, “Need it soon”, and “Grab it now!”) |
| CreatedUtc | DateTimeOffset | Utc created date |
| ModifiedUtc | DateTimeOffset | Utc modified date |

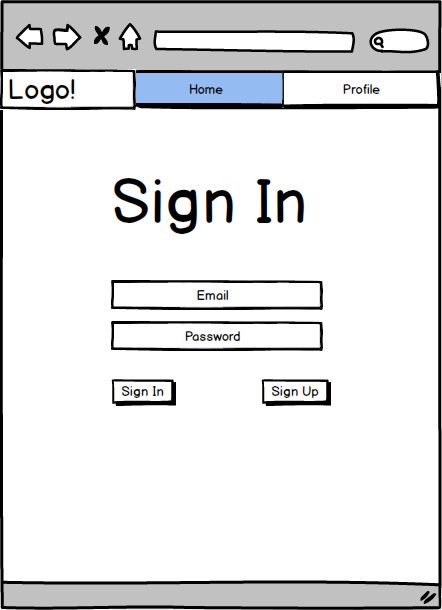
### ShoppingListItemNote

|  |  |  |
| --- | --- | --- |
| Id (key) | Int | Unique Id |
| Body | String |  |
| CreatedUtc | DateTimeOffset | Utc created date |
| ModifiedUtc | DateTimeOffset | Utc modified date |

## User Interface

Mockups of the main pages of the app.

### Shopping List – Login page



### Shopping List – Signup page



### Shopping List – User edit page

### 

### 

### 

### Shopping List – Lists page



### 

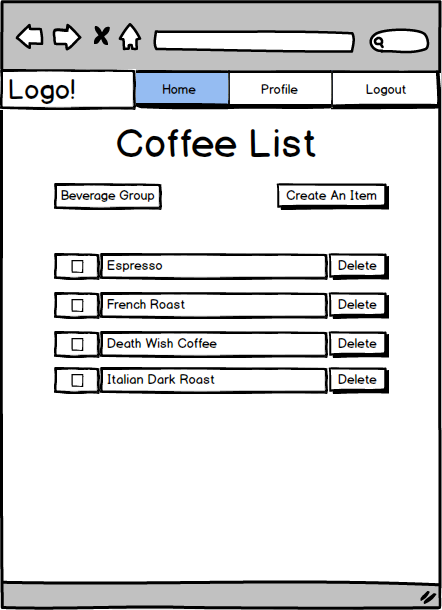
### 

### Shopping List - Add List page



### Shopping List – List page

* Add list edit button
* Display note under each item
* Add edit button to each item



### Shopping List – Add List Item page