# **Software development requirement & planning document for *Wedding APP Micro Service***

**1.1 The Wedding App micro service were build using these technologies:**

* Intelij IDE (you can also run the application using other IDEs such as Eclipse, if they support all the requirements).
* Spring boot version 2.5.3
* JDK 1.8
* Maven

**1.2 The following dependencies added to maven which can be found in the pom.xml dependencies’ tag.**

* H2 database
* Spring web
* JPA repository
* Junit 5

**1.3 UI technologies used:**

* HTML 5
* CSS 3
* Bootstrap 4

# **Statement of purpose**

Wedding App MS is an online wedding planner, it enables the user to explore between available wedding packages, add extra items to their desirable package, choose the drink package which suits them best and finally gets an estimated price based on their selection. This MS also enables the business owners to add, update and delete their services/packages through the admin part.

Wedding App MS is designed for wedding venues or related businesses. It can easily be used by users who are planning to have a wedding. It simplifies the planning procedures for both businesses and the customers. Users can combine the different wedding styles closer to their budget at the comfort of their home, instead of having hours of conversations with the wedding planners on-site.

# **Features included in the APP**

The Wedding app MS consists of several Html pages. The main page is for navigation purposes. It welcomes the users, and users can navigate through the different pages with the navigation bar provided. Also, there is a button provided that will redirect to the admin side. The admin side is for administration purposes, businesses can use this button to access the data management pages.

## **3.1 Customer side**

“Our packages” link provided in the nav bar, will redirect the user to the list of the packages. Wedding packages, Add-ons, and drink packages available can be seen on this page. Users can choose each package by the drop-down list provided below the lists. If the user does not choose any item, the first option will be chosen by default. And finally, at the bottom of the page, the number of guests should be entered. This number is required and cannot be negative. Then by entering submit user will be redirected to the result page.

In the resalt page the total price will be printed for the user along with a thank you message. User can continue to navigate through the site by the navigation bar to go back to the different pages or exit the application.

## **3.2 Admin side**

By pressing the blue admin button provided in the top left of the page, you can be navigated to the admin data management system, where they can play around with data stored in the database.

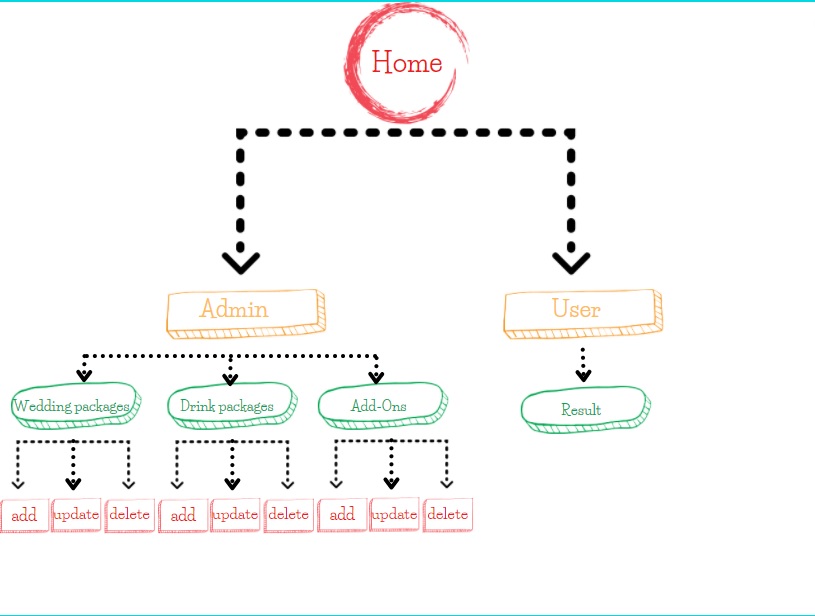
There are different pages available in this part which can be accessed through the navigation bar provided on each page. Note that this navigation bar is different from the one available on the user side.

The “Wedding packages” link redirects to the list of the available packages. There are buttons to add, update or delete the packages.

The “Drink Packages” and “Add-Ons” links also will redirect to their relative pages and admin will be able to manipulate the data related to these parts.

There are several additional links and buttons available, which are for designing purposes and do not have actual functionality. These links are the Log In button located in the top-right of the pages, about us and, contact us that can be seen in the nav bar.

You can see a simple layout for the app in the following diagram:



## **3.3 feature of APP**

# **Implementation plan**

1. I start this MS by creating an empty maven project with the spring initializer
2. The required dependencies added, which are listed below:

* Spring Web: Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
* Spring Data JPA: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
* H2 Database: Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser-based console application.
* Thyme leaf: modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.
* Junit 5 for performing the developer-side testing

1. Git hub repository created and initialized along with the first commit
2. The planning and designing phase started to gain a better understanding of the project.
3. Models, classes, and packages were implemented which are as follow:

### **4.1 Models and classes**

* Wedding packages: This class is a model class for wedding packages. Which stores the package id, hour (the length of the function), Style of the wedding, and the price of each package.
* Drink packages: The Drink packages class is another model class to store each drink package; this class includes the drink package's id(auto-generated), price, name, and detail.
* Add-Ons: The class add-on also is a model class That is used to store the add-on id which is auto-generated, name, price, and detail.

The required getter and setter were generated for each class. In addition, the classes were mapped to the database with @Entity and @Table annotations. Id is also auto-generated with @Id and @GeneratedValue (strategy = GenerationType.IDENTITY) annotations.

### **4.2 Packages**

* WeddingAppSpringboot.controller: The controller controls the data flow into the model object and updates the view whenever data changes.
* WeddingAppSpringboot.model: Which includes the model classes mentioned above
* WeddingAppSpringboot.repository: to simplify the development of the database, Spring Data JPA was used. For each entity, a repository interface was defined in the application. A repository contains methods for performing CRUD operations.
* WeddingAppSpringboot.service: which contains all the methods required for adding, updating, deleting, and manipulating data which is later called in the controller.

1. With the @Autowired annotation the repository layer injected to the service layer.
2. Methods were created in the service layer to add the functionality to Wedding App microservice
3. Methods used for reading the data from data base:

* getAllWeddingPackages(); returns a list of Wedding packages
* getAllDrinkPackages(); returns a list of Drink packages
* getAllAddOns(); returns a list of Add-ons

1. Methods for finding the data based on their ids:

* getWeddingPackageById(long id); returns a weddingPackage object
* getDrinkPackageById(long id); returns a drinkPackage object
* getAddOnsById(long id); returns a addOn object

1. Methods for updating the data in the database:

* updateWeddingPackage(WeddingPackage weddingPackage);
* updateDrinkPackage(DrinkPackage drinkPackage);
* updateAddOn(AddOn addOn);

1. Methods for deleting the data in the database:

* deleteWeddingPackage(Long id);
* deleteDrinkPackage(Long id);
* deleteAddOn(Long id);

1. method used to calculate the final price:

* CalculateTotalPri calculateTotalPrice(double wpPrice, double dpPrice, double aoPrice, int guests);

1. Regular commits were made in the git hub repository to maintain the version control
2. The h2 data base were connected and customized in the application.properties
3. A data folder created to store and keep the data, since H2 database is a in-memory data base and does not keep the data
4. The required table for each model were maid with the relative fields to store the model data
5. The data can be added, updated, deleted or found by repository layer which extends the JPA repository
6. By the use of Autowiring the service layer in the controller layer, the methods in controller were created with proper annotation such as @GetMapping, @PostMapping, and @RequestMapping.
7. Html pages were created in the src/main/resources/ templates folder which are listed below:

* index.html
* user-page.html
* result.html
* admin\_weddingPackages.html
* admin\_addOns.html
* admin\_drinkPackages.html
* new\_weddingPackages.html
* new\_addOns.html
* new\_drinkPackages.html
* update\_weddingPackages.html
* update\_addOns.html
* update\_drinkPackages.html

1. Using Thymeleaf which is an open-source java library, and is used as a java template engine, connected the controller to the Html pages, to get and post a response or redirect to different pages.
2. Finally using, CSS 3, Bootstrap 4 create a simple yet responsive design for the Wedding app MS
3. Tests were implemented to make sure that the methods are functioning properly and without any error.

The below diagram shows the application flow of my Spring MVC web application with Thymeleaf:



### **4.3 Flow off Spring MVC**

# **Junit testing**

To make sure that the Wedding app MS is running without any problem, adequate tests were implemented on the service layer and all the methods in the service layer were tested successfully. And can be found in src/test folder.

In addition, other forms of error handling were performed in Html forms and inputs such as:

- The value of the text boxes can’t be empty

- The text boxes which are designed for numbers, only accept numbers

- The negative numbers cannot be submitted to the database

# **References**

* Spring Initializr ’ <https://start.spring.io/>’
* Spring Boot CRUD Web Application with Thymeleaf, Spring MVC, Spring Data JPA, Hibernate, MySQL ‘[Spring Boot CRUD Web Application with Thymeleaf, Spring MVC, Spring Data JPA, Hibernate, MySQL (javaguides.net)](https://www.javaguides.net/2020/05/spring-boot-crud-web-application-with-thymeleaf.html)’