DP1 2020-2021

System Design Document

Project Tabernas Sevilla

https://github.com/gii-is-DP1/dp1-2020-gi-01

Members:

Buiza Núñez, Juan
Cabezas Díaz, Esteban
De las Heras Cózar, Adrián
Núñez Arenas, Carlos
Rouichi, Adil
Zamudio Amaya, José Antonio

Tutor: José María García Rodríguez

GRUPO GI-01

Version 1.1

09/01/2021

Version history

Date	Version	Description of changes	Sprint
13/12/2020	1.0	• Creating the document • Looking for the information needed.	2
7/1/2020	1.1	• Added domain/design diagram. • Added Layer diagram. • Explanation of the application of the MVP pattern. • Explained 2 decisions we had to make.	3

Contents

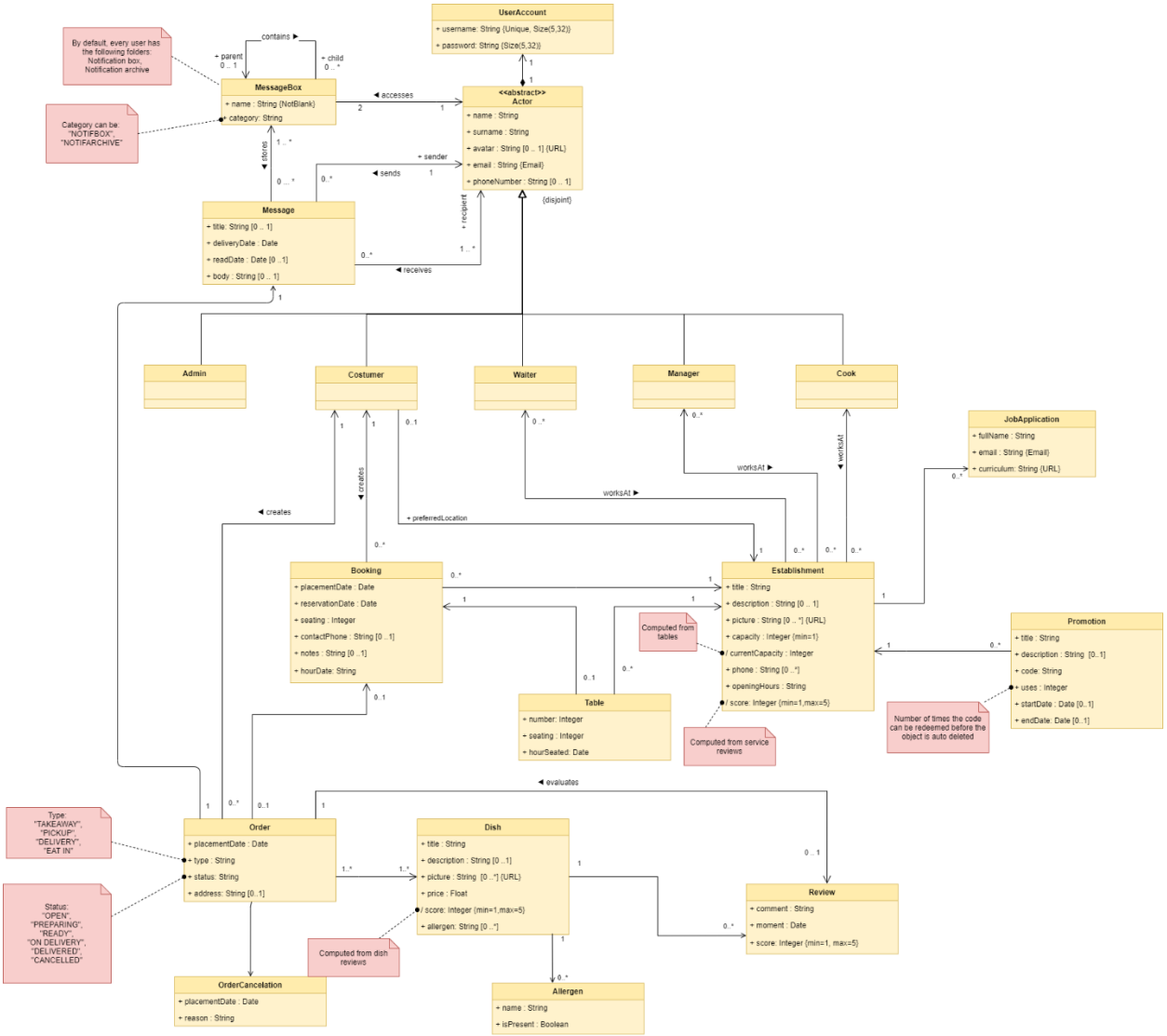
Introduction

The system in development fulfils three fundamental functions for Tabernas Sevilla's business model – provide information about the menu and establishments, order food (for take away, delivery, pickup or eat in) and reserve a table. The system will consist in a web application. A fully interactable menu will be developed from which users can select which dishes from Tabernas Sevilla's extensive menu they wish to order and which of the 4 locations throughout Seville to collect or eat at.

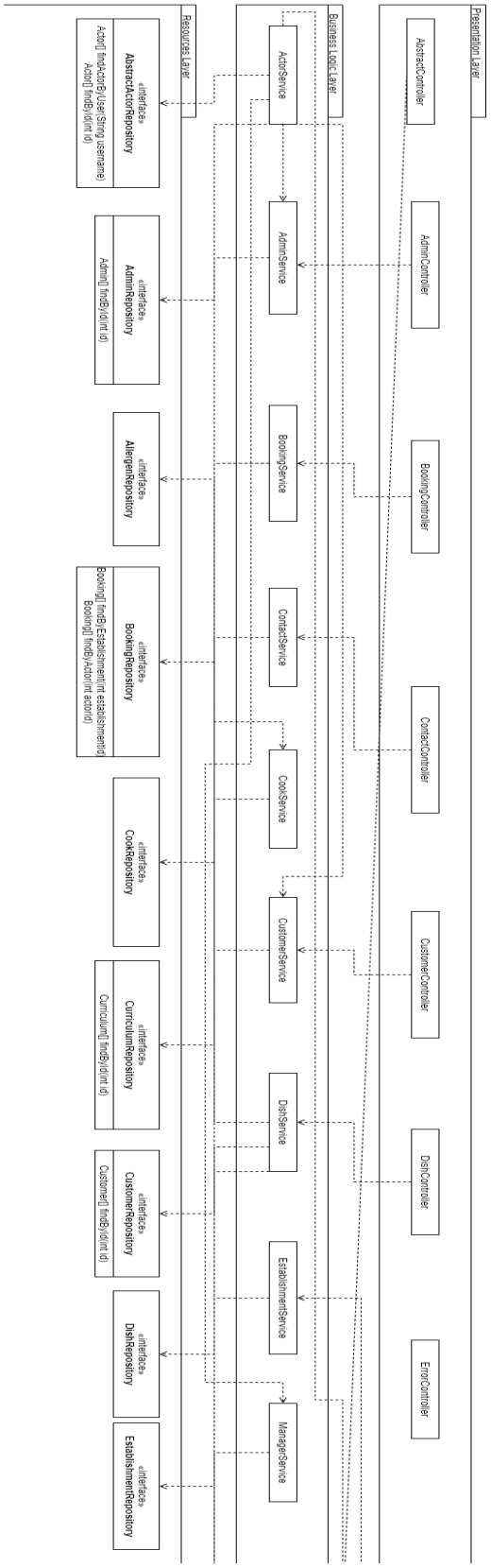
In addition, it should implement the functionality to manage the bar's capacity; being able to make table reservations with time selection, and if desired, also pre-ordering dishes. This later point helps to solve one of the basic problems of the business, the management of the waiting list, which is currently done by hand by the waiters.

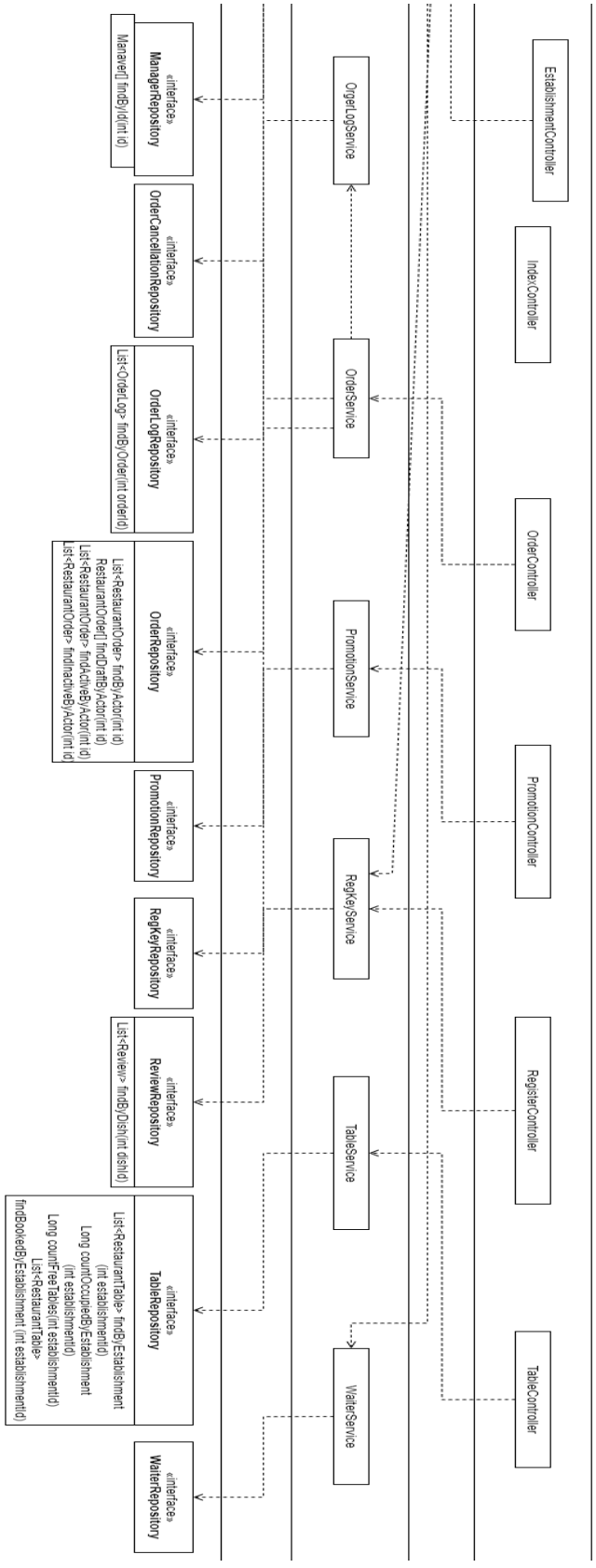
UML Diagram:

Domain/Design Diagram



Layer Diagram




EstablishmentController
IndexController
OrderController
PromotionController
RegisterController
TableController
OrgetLogService
OrderService
PromotionService
RegKeyService
TableService
WaiterService
«interface» ManageRepository
Manaver(): findBy(id(int id)
«interface» OrderCancellationRepository
«interface» OrderLogRepository
List<OrderLog> findBy(Order(int orderId)
«interface» OrderRepository
List<RestaurantOrder> findBy(Actor(int id)
RestaurantOrder[] findDraftBy(Actor(int id)
List<RestaurantOrder> findActiveBy(Actor(int id)
List<RestaurantOrder> findInactiveBy(Actor(int id)
«interface» PromotionRepository
«interface» RegKeyRepository
«interface» ReviewRepository
List<Review> findBy(Dish(int dishId)
«interface» TableRepository
List<RestaurantTable> findBy(Establishment
(int establishmentId)
Long countOccupiedBy(Establishment
(int establishmentId)
Long countFreeTables(int establishmentId)
List<RestaurantTable>
findBookedBy(Establishment (int establishmentId)
«interface» WaiterRepository

Design and architectural patterns applied.

Pattern: Model–view–controller

Type: Design

Application Context

Model–view–controller (usually known as MVC) is a software design pattern commonly used for developing user interfaces that divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways that information is presented to and accepted from the user.

Classes or packages created.

The components of the MVC are:

- Model: the central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic, and rules of the application.
- View: any representation of information such as a chart, diagram, or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- Controller: accepts input and converts it to commands for the model or view.

In addition to dividing the application into these components, the model–view–controller design defines the interactions between them.

- The model is responsible for managing the data of the application. It receives user input from the controller.
- The view means presentation of the model in a particular format.
- The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it, and then passes the input to the model.

Advantages achieved by applying the pattern.

1. Faster Web Application Development Process: MVC offers support for rapid and parallel development. So, developing web applications using the MVC model it is possible that one developer work on the view while the another can work on the controller. This helps for easy implementation of the business logic of the web application. It surely benefits developers for completing the web application three times quicker compares with the applications that are developed using other development patterns.
2. MVC web application supports asynchronous technique: .Net developers can integrate MVC architecture with JavaScript Framework. It means that MVC applications can be made to work even with PDF files, site that runs only on the specific browsers, and for the desktop widgets. MVC also allows using the asynchronous technique, which allows web developers to build faster loading web apps.
3. Offers multiple views: In the MVC architecture, it is possible to create multiple views for a model. Today, there is a great demand for accessing new ways to access your application and for that MVC development is certainly a great solution. Furthermore, in this method, code duplication is certainly less as it can separate data and business logic from the display.

4. Ideal for developing large size web application: It works well for developing web applications which need to be supported by large teams of developers and for Web designers who wants greater control over the application behavior.
5. MVC model returns the data without the need of formatting: It is helpful for the developers because the same components can be used and called for use with any interface. For example, any types of data can be formatted using HTML, but with MVC framework you can also format using the Macromedia Flash or Dream viewer.
6. A modification never affects the entire model: It is obvious that you make minor changes in a web application such as like changing colors, screen layouts, fonts and adding an extra support for mobile phones or tablets. Furthermore, adding a new type of views is very easy in MVC pattern as Model part does not depend on the views part. So, any changes in the Model will never affect the entire architecture. Thus, today there are many enterprises which opting for the development of web applications based on MVC architecture to take the above-given advantages. Today you need to find certified MVC .Net developer which satisfies your web application development requirement.

Design decisions

Decision 1

Description of the problem:

We considered using Thymeleaf, after Juan proposed to use it as he had used it before. Thymeleaf is a modern server-side Java template engine for both web and standalone environments.

Solution alternatives evaluated:

Alternative 1st: Continue to use JSP and JSTL for the views as in the Pet Clinic project.

Advantages:

- We can take the Pet Clinic project as an example.
- No new learning required.
- We can use EV's technical videos to help.

Inconvenient:

- Sometimes too much code is repeated.
- Does not add value to the project.

Alternative 1.b: Implement the views with Thymeleaf.

Advantages:

- It provides full integration with Spring Framework.
- Facilitates the use of fragments to reuse code.
- It brings more value to the project.
- We can aim for a better mark.
- To expand our knowledge.

Inconvenience:

- We do not have many examples to take as a reference.
- It requires an extra effort on our part.

Justification for the solution adopted.

Although it means extra work for us, we believe that it is worth selecting the alternative 1.b, because Thymeleaf is a technology whose popularity is increasing in recent years and brings value to both our project and our training as developers.

Decision 2

Description of the problem:

If there are some dishes created in the database with the script data.sql, when doing create or update operation, it returns an error because JPA does not auto-increase the value and for that reason there were several dishes with the same ID.

Solution alternatives evaluated:

Alternative 1.a: Include the data in the BD initialization script itself (data.sql).

Advantages:

- Simple, requires nothing more than writing the SQL that generates the data.

Inconvenient:

- Slows down all work with the system for development.
- We must look for the real data.

Alternative 1.b: Use the PopulatorDatabase class, a way to enter objects via Java with the methods we have created instead of entering them directly by SQL script.

Advantages:

- Few lines of code in which the methods already created are used.
- Frees up JPA's workload.
- Makes handling entities easier with controllers.
- Makes integration easier since it is Java-based.

Inconvenience:

- It is laborious to include all the data in this way.

Justification for the solution adopted.

For most entities we will use alternative 1.a, as there is no conflict. But for some entities such as Dish and Review, we will use alternative 1.b since multiple errors occur when doing the initial data loading with the SQL script.