

<Name of Software Application>

# **CS 465 Project Software Design Document**

Version 1.0

## Table of Contents

[**CS 465 Project Software Design Document** 1](#_Toc36198462)

[Table of Contents 2](#_Toc36198463)

[Document Revision History 2](#_Toc36198464)

[Instructions 2](#_Toc36198465)

[Executive Summary 3](#_Toc36198466)

[Design Constraints 3](#_Toc36198467)

[System Architecture View 4](#_Toc36198468)

[Component Diagram 4](#_Toc36198469)

[Sequence Diagram 5](#_Toc36198470)

[Class Diagram 5](#_Toc36198471)

[API Endpoints 5](#_Toc36198472)

[The User Interface 6](#_Toc36198473)

## [Document Revision History](#_heading=h.lnxbz9)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 2.0 | 31 January | Sergio Mateos | Documented information about the functionality of Travlr Gateaway |

## Instructions

Web applications are complicated because of the complexity of the technology to support thousands of connections in a short time frame. The team is currently working on building a full web application for the client, Travlr Getaways. Maintaining the functionality and quality of the web application is very important. The Software Design Document (SDD) will contain technical specifications and descriptions of the Travlr Gateways web application.

## [Executive Summary](#_heading=h.35nkun2)

The Travlr Gateway is a web application that is divided into index, user, and travel information. The users are available to see the picture of the destination, rooms, and meals. The users are available to access their information on the web application to make a reservation.

The Travlr Gateways web application would be developed by using the MEAN (MongoDB, Express, Angular, and Node) stack, MEAN is a JavaScript framework for developing web applications. The Travel Gateaways web applications require flexibility, scalability, and extensibility, that’s why the MEAN stack is the best option as the framework. On MEAN stack Node written on JavaScript would be the framework, MongoDB would be the database, Angular would be the client side and front end, and Express the webserver. Customers would be easy access to any part of the web application to maximize the convince of the customers. Since the Tralvr Gateways’ web application will use MongoDB, the data would be multidimensional which increases the performance.

The customer-facing side will allow them to search for travel, rooms, food, and everything that customers will need to have an amazing vacation. The Single-Page Application (SPA) reduce the amount of loading time, decrease the difficulty of development, and is more responsive, but it’s not well suited for large-scale application since the scalability is minimal.

## [Design Constraints](#_heading=h.1ksv4uv)

The design constraints for developing the Travlr Getaway web application using MEAN are beneficial for the stability of the web application and will facilitate the change and updates. The high amount of traffic that the web application will have been easy to handle by using the MEAN stack. The web application requires compatibility with old and new web browsers and devices. The web application must implement proper security procedures, designed properly developed for the customer’s convenience, and quality size for different devices. Also, budget and timeline are crucial for a well-developed project. The failure of any of the last design constraints could lead to the failure of the project development.

## [System Architecture View](#_heading=h.44sinio)

### Component Diagram



The Travlr Getaway web application will implicate three major components: Database, Client, and Server. The components would have their specific function and purposes:

* Client: The client components would start from the client session which along with the web browser will connect to the traveler portfolio. The images stored in the graphic library would be displayed in the web application.
* Database: The information would be stored on MongoDB, containing the information from the users and the web application.
* Server: Information will need to be validated before access to the site which would be authenticated in the authentication server, after the authentication is successfully passed it will allow the, to use the server session. The server session will connect to the Travel Database and Mongoose ODM which would export the information.

The connection would be in a loop and repeat, the customer will access content via web application where a web application will display, and the customer will be able to interact. The Client would connect to the Database where the data and information would be stored and be able to export to the Server. The data would be analyzed and passed to display for the customer.

### Timeline Description automatically generatedSequence Diagram

Client Side:

1. Route:
   1. -> The actor (user) requests to the system and its transfer to controllers by redirecting to views.
2. Controllers:
   1. -> Interact with the actor and send it to the HTTP website.
3. HTTP Client:
   1. -> Load and authenticate the page.
   2. <- Display the views on the website.
4. Browser:
   1. -> Request controller functions and send them to controllers.
   2. <- Pass the result and respond appropriately.

Server-Side

1. Controller:
   1. -> Call the service to
2. Browser:
   1. -> Request the information stored in the database
   2. <- The information it sends back from the client browser.
3. MongoDB:
   1. <- Retrieve all the information which would be filtered to send it back to the actor side.

## Diagram, schematic Description automatically generatedClass Diagram

The class diagram is the process of the execution of the web application. The actor (user) who is looking for a trip would access the web application by an HTTP. The process is easy for the user side, but the backend is where the application is functioning. Based on what the actor is seeking to do the website will implement the right functionality determined by the user selection and communicate back to the actor. The Object-Orientate system describes the top part are the class name. The second space represents the attributes, which would be described as variables. The last spaces are the methods. The arrows represent the relationship between classes. The TripInfo is inherited from the CruiseInfo, FlightInfo, and HotelInfo. CruiseInfo, FlightInfo, and HotelInfo. Would inherit their respective classes depending on the actor selection. All the classes would provide information to the TripInfo and transfer the information to the Travel\_Agent which would pass it to the MemeberAccount.

## [API](#_heading=h.2jxsxqh) Endpoints

| **Method** | **Purpose** | **URL** | **Notes** |
| --- | --- | --- | --- |
| **GET** | <Retrieve list of things> | </api/things> | <Returns all active things> |
| **GET** | <Retrieve single thing> | </api/things/:thingId> | <Returns single thing instance, identified by the thing ID passed on the request URL> |
| **POST** | <Create list of things> | </api/things> | <Create a new list of things> |
| **POST** | <Create single thing> | </api/things/:thingId> | <Create a single thing by the ID passed on the request URL> |
| **PUT** | <Update list of things> | </api/things> | <Update & replace a list of things. It will update all the records and overwrite the data> |
| **PUT** | <Update single thing> | </api/things/:thingsId> | <Update & replace single thing identified by ID pass on URL. It will update all the records and overwrite the data> |
| **PATCH** | <Update/modify list of things> | </api/things> | <Update the full list of things and modify the list of things> |
| **PATCH** | <Update/modify single things> | </api/things/:thingId> | <Update a single thing with the ID URL pass and modify the list of things> |
| **DELETE** | <Delete the list of things> | </api/things> | <Delete all the list of things> |
| **DELETE** | <Delete a single thing> | </api/things/:thingId> | <Delete a single thing passed on the URL with the ID pass> |

## The User Interface