**GoBike**

**Software Design**

**CSCI-P465/565 (Software Engineering I)**

**Project Team**

**Will Boland**

**Jyoti Bhushan**

**Deepika Awasthi**



# 1. Introduction

This section introduces the design approach to the software system.

## 1.1 System Description

* The purpose of this project is to provide a one stop solution for Bike travelers. Often, they face challenges in planning a tour mostly due to lack of time and unaware about new places. This application has it all (searching for accommodation, exploring places, booking various mode of travel) plus they will get a bike to explore places
* This application is also meant to motivate people to travel on bikes as it is environment friendly and provides access to explore more by visiting remote areas where vehicles are inaccessible.
* There will be tremendous benefit to customers, company and environment.

## 1.2 Design Evolution

This section is intended to document the rationale behind the selected design solution.

### 1.2.1 Design Issues

Designing of UI is inspired by Material UI and React JS. React JS helps to keep the components in modular mode and hence the maintainability would be easy

**1.2.2 Candidate Design Solutions**

We have tried to keep UI and Backend as two different services (deployed as 2 microservices on AWS beanstalk). The backend APIs will be called internally by UI and hence the rendering will take place. However, in order to make the design more robust, if this solution doesn’t meet the efficiency, we will try to modify the architecture where we will keep the service and UI on one server and integrate the 2 components in one.

### 1.2.3 Design Solution Rationale

We have chosen the former approach out of the two as we are focusing more on modularity of project and keeping the 2 stacks separate which will facilitate the ease of design and more creativity.

## 1.3 Design Approach

### 1.3.1 Methods

This application will be built as per Object Oriented Design. The project will take advantages of following design patterns:

* MVC Design Pattern: Application will have three different layers: Model layer will contain only application data information, how data is designed and will be hidden from the User. Presentation layer also known as View will mostly deal with the layout of the page viewed by User. The Controller layer which is the intermediate layer between View and Model will mostly deal with communication between UI and services.
* Façade Design Pattern: Application will utilize this design pattern to hide complexities of the application and will provide users a simple interface to interact and perform action.
* Observer Pattern: Application will utilize this pattern to observe any changes done on UI. For example, this can be applied to the search box. An event will be generated upon any changes done.

### 1.3.2 Standards

Followings standards are being followed:

* + The project has multiple interfaces which allows users to interact with the system and has followed naming conventions as per Java naming convention (Camel case) for Interfaces, package, Classes, Methods, Constants (all uppercase) etc.
  + In Git, we have maintained a Master and Develop branch where direct commits are not allowed. This will help in keeping the code base bug free. Developers will have to raise Pull requests to develop and upon proper review of code by other members, feature branches will be merged to Develop and then to Master which will be used to deploy the application in the upper environment.

### 1.3.3 Tools

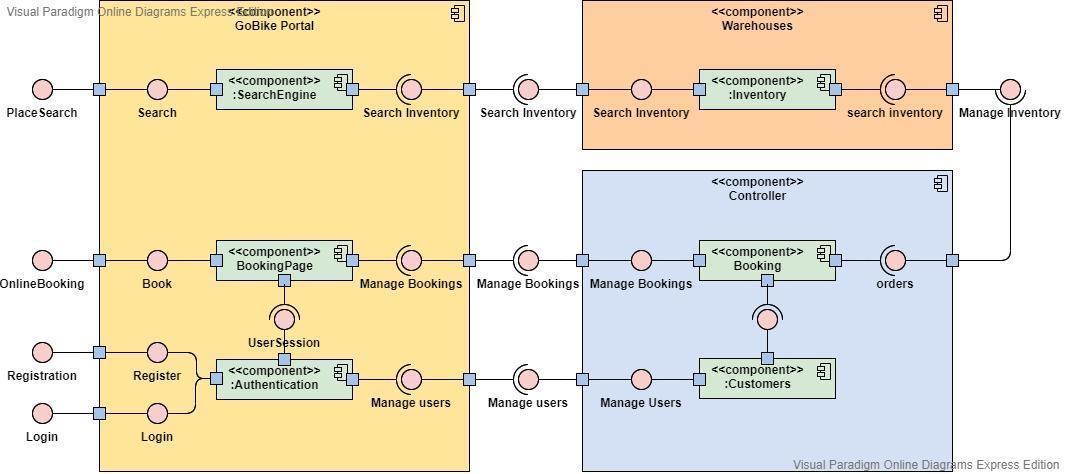
Online tool for creating design document: <https://online.visual-paradigm.com>

# 2. System Architecture

## 2.1 System Design

System is designed using MVC architecture along with other design patterns which involves data abstraction, modularization and security. System is designed as per component basis which means there is segregation of responsibilities.It has mostly four layers:

* UI Layer
  + This is the User Interface layer responsible for the main interaction between the users and our project. It consists of all of the layout and visual aspects of the program.
* Controller
  + This layer consists of a series of components that allow for different features. For example, the search controller allows for searching and interfacing with different packages in our product.
* Service
  + This layer mostly contains business logic. It validates incoming requests, process requirements, gets required information from the database and returns only required information to an external API which then talks to UI or other exposed APIs.
* DAO
  + This is a Data Access Layer responsible for interacting with databases and hiding information which is not required for external resources.



## 2.2 External Interfaces

The application will interact with following external Interfaces:

* Payment Gateway
  + This will allow for easy payment and processing of credit card transactions from our platform. This also will incorporate failed transactions.
* Hotel APIs
  + We will interface with hotel APIs in order to provide up-to-date information for our users. This will include current hotel accommodations, pricing, and other information relevant to the user experience.
  + For this, we would be utilizing external APIs from ‘Amadeus’. Amadeus provides all required APIs for searching accommodation including hotels and homestays and booking.
* Travel APIs
  + Travel APIs will allow us to work with airline booking management systems to provide real time information on airline pricing and times.
  + For this, we would be utilizing external APIs from ‘Amadeus’. Amadeus provides all required APIs for searching airports, flights and booking.

# 3. Component Design

The Application comprises multiple components: Search, Accommodation Booking, Travel Booking, Itinerary, Payment.

**3.1 Search Places**

This component will allow users to search for their locations to go biking or for our package deals. It will provide an interface for users to be navigated to other portions of the site.

**Responsible Development Team Member:**

Will Boland

**Component User Interface**

Search box, autocomplete, dropdown, clickable

**Component Objects**

Controller: SearchController-> Allows for easy navigation and search of packages and places.

Service:

SearchService : Handles looking up searches for caching.

**Component Error handling**

We will display an error message with common fixes and solutions.

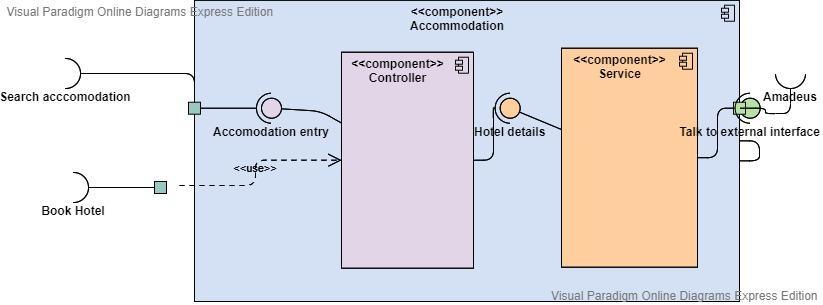
**3.2 Accommodation Search and Booking**

This component will allow users to book accomodation. It will provide an interface to users to select flight and dates as per their plan which upon confirmation will be booked.

**Responsible Development Team Member:**

Jyoti Bhushan

**Component Diagram**

****

**Component User Interface**

External Interface: The application will be using an external interface from ‘Amadeus’ for getting all accomodation related API.

We will be having our own internal interface which will talk to external APIs. UI will interact will our internal API for uniform flow.

**Component Objects**

Model Object: Accomodation-> This model object represents the actual table. It contains following attributes:

* + User Id
  + Booking Id
  + Amount
  + BookingDate
  + Document <E confirmation page>
  + CheckInDate
  + CheckOutDate
  + Place

DTO Object :

* + AccomodationRequest: This DTO object will contain all the required information related accommodation booking.
  + AccomodationResponse: It will contain all information required to display on UI.

Controller: AccomodationController-> This will provide all APIs related to travel:

book(), retrieveAccomodation(), retrieveAll().

Service:

AccomodationService : This class is responsible for handling all the business logic required by this component. It contains methods like book, retrieveDetails.

**Component Error handling**

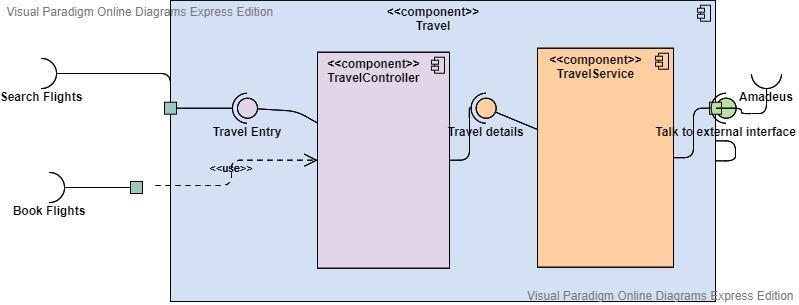
If the room is not available during the time

**3.3 Travel Search and Booking**

This component will allow users to book flights. It will provide an interface to users to select flight and dates as per their plan which upon confirmation will be booked.

**Responsible Development Team Member:** Jyoti Bhushan

**Component Diagram:**

****

**Component User Interface**

External Interface: The application will be using an external interface from ‘Amadeus’ for getting all hotels related API.

We will be having our own internal interface which will talk to external APIs. UI will interact with our internal API for uniform flow.

**Component Objects**

Model Object: Travel -> This model object represents the actual table. It contains following attributes:

* + User Id
  + Booking Id
  + Amount
  + BookingDate
  + Document <E Ticket in this case>
  + TravelDate
  + Source
  + Destination

DTO Object :

* + SearchAirportResponse : This contains all the information related to airport (iataCode, terminal, name)
  + SearchFlightRequest: This DTO object contains all the required information for searching flight.
  + SearchFlightResponse: It contains all flight details.
  + SearchFlightAmadeusResponse : This object contains all the information which we get from amadeus

Controller: TravelController-> This will provide all APIs related to travel:

searchAirport(), searchFlights()

Service:

* + TravelService : This class is responsible for handling all the business logic required by this component. It is responsible for interacting with Amadeus for details and transforming responses to render on UI.

**Component Error handling**

Travel booking shouldn’t be done if the dates selected are past dated.

**3.4 Itinerary**

This component is responsible for displaying the entire itinerary for Users. Itinerary will be built as per the user preference and booking. This component is also responsible for a detailed plan of redefined packages. It will include following information:

* + Start and end date of the trip
  + Total duration of the trip
  + Day wise plan
  + Total cost of the trip
  + Accomodation details
  + Travel details
  + Schedule for every activity planned for that trip

**Responsible Development Team Member:** Deepika Awasthi

**Component Diagram**

**Component User Interface**

A separate Tab for users will be provided which lists a detailed plan of the trip. It will provide easy access to day to day plans starting from breakfast to dinner.

It will have visual information in the form of a map displaying the source and destination of the tour.

**Component Objects**

Model Object:

Itinerary-> This model object represents the actual table. It contains following attributes:

* + name
  + numberOfDays
  + startDate
  + endDate
  + description

UserItinerary -> This model object contains all information related to flights, trip, accommodations and places. It has following attributes:

* + User
  + Itinerary
  + List<Flight>
  + List<Accommodation>
  + List<Place>

DTO Object :

* + ItineraryDetails: This DTO object will contain all the required information related itinerary.

Controller: ItineraryController-> This will provide all APIs related to itinerary. Following are APIs:

* createItinerary
* GetAllItinerayByUser
* GetItineraryByUserAndName

Service:

* + ItineraryService : This class is responsible for handling all the business logic required by this component. It contains methods like create, get, update.
  + Using GoogleMap APIs for displaying destination on Itinerary page

**Component Interfaces (external/Internal)**

GoogleAPI

**Component Error handling**

**3.5 Payment**

This component is responsible for payment for the trip being planned by the User. It will provide flexibility to pay for accommodation, travel, meal at once which is pretty straightforward.

**Responsible Development Team Member:** Deepika Awasthi

**Component Diagram**

**Component User Interface**

Users will have a payment page where all the required information can be reviewed. Upon confirmation, an event will be triggered to the application internal interface which will then talk to the external interface for payment.

**Component Objects**

Model Object: Payment -> This model object represents actual table. It contains following attributes:

* + User Id
  + Trip Id
  + Payment amount
  + Transaction Date
  + Transaction Id
  + Booking Id

DTO Object :

* + PaymentRequest: This DTO will be used for requesting payment. Attributes : UserId, TripId, Amount, TransactionDate.
  + PaymentResponse: This will contain transactionId, Booking Id, and appropriate message in case of both success and failure
  + Payment Details: This DTO object will contain all the required payment information.

Controller: PaymentController-> This will provide all APIs related to payment: payment, getPaymentDetails.

Service:

* + PaymentService : This class is responsible for handling all the business logic required by this component. It contains methods like makePayment, retrievePaymentDetails.

**Component Interfaces (external/Internal)**

Internal Interfaces:

* + Payment controller which will have an API which will be accessed by UI to call internal services for payment. This controller will have APIs like getPaymentDetails, payment etc
  + Payment service deals with the implementation of the API provided in the controller.

External Interfaces: Our internal service will interact with payment gateway to do the actual payment

**Component Error handling**

For transaction failure appropriate error handling will be performed. The application is capable of handling any sort of error making use of try catch and throws exception handling. If some error occurs then the whole transaction will be rolled back without affecting the user.

**3.6 Rating and Review**

This component is responsible for working with the rating system of all different places to go. Rating system will:

* + Allow 1-5 star rating and review system for trips.
  + Allow rating and reviews posted about their experience
  + Allow package review for plans.

**Responsible Development Team Member:** Will Boland

**Component Diagram**

**Component User Interface**

Will have a simple drop down menu system next to any component system needed.

**Component Objects**

Model Object: Rating -> This model object represents the actual table. It contains following attributes:

* + User Id
  + Rating Id
  + Trip Id (can be null)
  + Date
  + Rating
  + Notes (can be null)

DTO Object :

* + Rating Details: All required information for the rating.

Controller: RatingController-> Allows for easy rating access and responses.

Service:

* + RatingService : This class is responsible for handling all the business logic required by this component. It contains methods like addRating, retrieveRatings.

**Component Interfaces (external/Internal)**

Internal Interfaces:

* + Rating controller which will have an API which will be accessed by UI to call internal services for payment.
  + Rating service deals with the implementation of the API provided in the controller.

External Interfaces: NONE.

**Component Error handling**

If a user can not have a rating submitted, we will provide information on why it failed and how to fix it.

**Revision History**



|  |  |  |
| --- | --- | --- |
| **Revision** | **Date** | **Change Description** |
| Software Design Document - V1.0 | 10/04/2020 | Final Draft GoBike |
| Software Design Document - V2.0 | 10/18/2020 | Added external interface information which the system is going to utilize for Travel and Accommodation  Added component diagram for accommodation and Travel |
| Software Design Document - V3.0 | 11/01/2020 | Updated Service for Travel and accommodation |
| Software Design Document - V4.0 | 11/15/2020 | Updated Itinerary Details for model , service and controller |

## Last Modified: 10/18/2016