Software Engineering
Software Requirements Specification (SRS) Document
Pick And Go

09/26/2023

Version 1

By: Marcus Thompson

Duncan Norment

Kenneth Alvarado

1.Introduction

1.1 Purpose

The goal for the Pick and Go application is to give tourists the opportunity to share and choose packages for new cities that they plan on traveling to for the first time. This will allow them to discover different popular attractions that they may be interested in, and make choosing travel destinations easier. New visitors will be able to create and share their own new discoveries as well and share them to others through their own personal packages.

1.2 Document Conventions

The purpose of Pick and Go application will allow people to be able to make their own package of all the attractions and places they like to go to in a particular city. New users will be able to choose a package by searching a city of interest and choosing between ones that have already been created. They will also be able to rate these packages on how their experience went. They will also be able to create their own personalized packages for others to select and review.

1.3 Definitions, Acronyms, and Abbreviations

Java: A programming language that will be used to handle manage packages from users.

HTML Hyper Text Markup Language: this code will structure the interactive web based application users will work with.

CSS Cascading Style Sheets: changes the look of the webpage and how it will be displayed for the users.

Spring Web: An open-sourced Java based Framework that will be used to build our web application by using Spring MVC. This is one of our dependencies for our web application.

NetBeans: An Integrated Development Environment that we will use to code our program in Java.

MySQL: Open-source database management system that will handle the City information.

MVC Model-View-Controller: the architectural pattern for our application.

Thymeleaf: A modern server-side Java template engine for both web and standalone environments. This is one of our dependencies for our web application.

1.4 Intended Audience

This document is intended for the developers, testers, and project manager.

1.5 Project Scope

The goal of this user friendly application is to help tourists plan their next dream vacation. Customers will be able to discover different locations that they might not have heard of before but will

remember forever. This will help for publicity for different locations by bringing in new customers. It will also make customers more likely to revisit after their first experience.

The benefits of the project to a business include:

· Increases publicity which will introduce more customers for the business and for them reference the business to other people.

· Takes stress off of tourists on deciding on travel plan or requiring to hire a travel agent.

· Can be used to create connections and networking with other people with the use of the rating and comment sections.

1.6 Technology Challenges

1.7 References

2: General Description

2.1 Product Perspective

Pick and Go was created to make traveling to new cities easier for tourists. The idea came from innovators that have a strong desire to travel themselves.

2.2 Product Features

The features for this product include the ability for tourists to create a personal account to create their own personal package for travel, explore other packages created by other tourists, review them, and comment. Developers will also have the ability to create, edit these packages for any tourist. For Administrators, they will have control over adding and deleting those accounts. There is also a customer support messenger for any tourist to communicate with an administrator for any change requests for their account.

2.3 User Class and Characteristics

Our web based application is straight forward and user friendly. The only computer knowledge required is being able to navigate a simple webpage, and search for cities they might be interested in. The application allows for tourists to focus more on enjoying a trip rather than spending time trying to plan it.

2.4 Operating Environment

This application is designed to work on the web and will be able to work on multiple devices such as computers, tablets, and mobile devices.

2.5 Constraints

2.6 Assumptions and Dependencies

This application is dependent on Spring web and Thymeleaf to create and run the web based application using the MVC architecture. The back end for the application will be developed using Java in Netbeans. MySQL will be used to quickly move and arrange data for specific packages and cities the application will use.

3. Functional Requirements

3.1: Primary

- ❖ Admin
  - -FR3.1.1: The admin can add a new user to the system, including specifying their role.
  - -FR3.1.2: The admin can update the information of an existing user, including their role.
  - -FR3.1.3: The admin can delete a user account from the system.
  - -FR3.1.4: The admin can log in to the system to access administrative functions.

-FR3.1.5: The admin has access to a customer service messenger to communicate with users regarding account-related issues.

- ❖ Developer
  -FR3.1.6: The developer can log in to the system using their credentials.
  -FR3.1.7: The developer can create a new package for a specific city, specifying attractions and places to visit.
  -FR3.1.8: The developer can set packages to be either private (for personal use) or public (shared with other users).
  -FR3.1.9: The developer can customize the entire package, affecting all users who select it.
  -FR3.1.10: The developer can leave comments on packages they have created.
- ❖ User
  -FR3.1.11: The user can log in to the system using their credentials.
  -FR3.1.12: The user can rate packages they have used.
  -FR3.1.13: The user can leave comments on packages they have used.
  -FR3.1.14: The user can customize a package for their own use, without affecting the package as a whole.
  -FR3.1.15: The user can store packages they have customized for later use.
  -FR3.1.16: The user has access to a customer service messenger to communicate with administrators regarding account-related issues.

3.2: Secondary Functional Requirements

- ❖ FR3.2.1: Implement password protection mechanisms to secure user accounts and information, ensuring that only authorized users can access specific functions and data.
- ❖ FR3.2.2: Implement an authorization scheme that restricts users from altering or viewing packages and information belonging to other users, ensuring data privacy and security.

4. Technical Requirements

4.1: Operating System and Capability

- ❖ The application should be compatible with any operating system that is able to view and interact with traditional web pages. However, we all are on Windows/Microsoft computers.

4.2: Interface Requirements

4.2.1: User Interfaces:

- User Login Screen
  - Layout: A simple login form with fields for username and password.
  - Buttons/Functions: "Login" button to submit login credentials.
  - Messages: Display messages for login success or failure.

- User Dashboard Screen
    - Layout: A dashboard displaying available cities and packages.
    - Buttons/Functions: Buttons to view and customize packages, rate packages, access customer service messenger.
    - Messages: Notifications for new messages or updates

- Package Selection Screen:
    - Layout: List of available packages for a selected city.
    - Buttons/Functions: Select a package, view package details, rate and comment on packages.
    - Messages: Package details and user comments.
- Package Customization Screen:
    - Layout: Customize the selected package by adding or removing attractions/places.
    - Buttons/Functions: Add/remove attractions, save customized package, return to package selection.
    - Messages: Notifications for successful customization.
- Customer Service Messenger Screen:
    - Layout: Chat interface for communicating with administrators.
    - Buttons/Functions: Send messages, view message history.
    - Messages: User and administrator messages displayed in chat format.
- Admin Login Screen:
    - Layout: A login form for admin access.
    - Buttons/Functions: "Login" button to submit login credentials.
    - Messages: Display messages for login success or failure.
- Admin Dashboard Screen:
    - Layout: Admin dashboard showing user management options.
    - Buttons/Functions: Manage user accounts, access customer service messenger.
    - Messages: Notifications for new messages or updates.
- User Management Screen (Admin):
    - Layout: List of user accounts with options for editing or deleting.
    - Buttons/Functions: Edit user details, delete user accounts.
    - Messages: Notifications for user account management actions.

4.2.2: Hardware Interfaces

The "Pick and Go" web application is designed to be compatible with various hardware devices, including:

- Smartphones
- Tablets
- Desktop Computers
- Laptops

Users should have access to the internet, a web browser, and the ability to interact with web pages on any of these devices to use the application effectively.

4.2.3: Communications Interfaces

The application will require the following communication interfaces:

- Internet Access: Users must have an active internet connection to access the web application.
- Database Connection: The application will connect to a MySQL database, which may be hosted locally or on a server.
- API Integration: It will connect to external services like the World Time API using HTTP to retrieve current date and time information.

4.2.4: Software Interfaces

The "Pick and Go" application will utilize the following software components:

- Frontend Framework: React will be used for building the user interface (UI) of the application, providing a responsive and interactive web experience.
- Backend Framework: Spring Boot with ThymeLeaf will be used to develop the backend of the application, handling server-side logic and rendering dynamic web pages.
- Database Management System: MySQL will serve as the database management system to store user accounts, package data, and other relevant information.
- Java Persistence API (JPA): JPA will be used to interact with the MySQL database from the Java backend.
- Communication Protocol: HTTP will be used for communication between the frontend and backend components, as well as for connecting to external APIs like the World Time API.

5. Non-Functional Requirements

5.1. Performance Requirements:

-The system will consume less than 50 MB of memory.

.	-A regular user will be able to just take their time and look through possible options and apply in less than 1 minute.

-Novice tour guide users should be able to complete the process of searching and creating a package within 10 minutes of starting the task.

-Expert tour guide users should be able to perform the same task in under 5 minutes.

5.2. Safety Requirements:

-Our program will store passwords in a safe manner by hashing them.

5.3. Security Requirements:

- People that decide to sign up to be a tour guide will only have access to message others and build their “tour” package. Only they will be able to customize their package and decide to make it private/public or to even delete it. As for a regular user they will only get a view option to see packages and message others, only thing that will be displayed for others are usernames, goes for tour guides too.

5.4. Software Quality Attributes:

5.4.1. Availability:

-The system should be available whenever it is hosted and data backups should be performed daily.

5.4.2. Correctness:

-The system must be able to show users different packages if there is one available for their desired destination, and be able to message the tour guide.

5.4.3. Maintainability:

-The codebase should be well-documented and follow coding standards to facilitate maintenance.

5.4.4. Reusability:

-Modules for user database access should be designed for potential reuse in future projects, as well as the code for certain features.

5.4.5. Portability:

-The system should be compatible with major web browsers and mobile devices.

5.5. Process Requirements:

5.5.1. Development Process Used:

-Waterfall

5.5.2. Time Constraints:.

-About 3 months to work on.

5.5.3. Cost and Delivery Date:

-No budget and the presentation is on 12-05-2023.

5.6. Other Requirements:

5.8. Use-Case Model Descriptions

5.8.1. Actor: Bob Jones (Tour Guide) - Responsible Team Member: (Duncan Norment)

Use-Case Name: Create Tour Package
Description: Allows the tour guide to be able to make a package of things he will offer, such as places to go.

Use-Case Name: Accept/Decline Clients
Description: Allows the tour guide to be able to accept a client or decline them.

5.8.2. Actor: John Adkins (Admin) - Responsible Team Member: (Kenneth Alvarado)
Use-Case Name: Manage User Accounts
Description: Allows the admin to create, update, or delete user accounts.

Use-Case Name: Customer Service
Description: Allows the admin message with users to help with any issues.

5.8.3. Actor: Jerald Smith (User) - Responsible Team Member: (Marcus Thompson)
Use-Case Name: View Packages
Description: Allows users to be able to search to attempt to find their desired package.
Use-Case Name: Apply for package
Description: Allows users to be able to apply for their desired package.

5.9. Use-Case Model Scenarios
5.9.1. Actor: Jerald Smith (Marcus Thompson)
- Use-Case Name: View Packages
● Initial Assumption: Jerald logs in.
● Normal: Jerald searches for their desired destination and looks through possible available tour packages.
● What Can Go Wrong: Jerald enters the destination name wrong and network loss.
● Other Activities: Being able to view the ratings/comments from other people for the tour packages.
● System State on Completion: Generates a list of available tour packages for the destination.
- Use-Case Name: Apply for package
● Initial Assumption: Jerald is logged in and has found a tour package he likes.
● Normal: Jerald will now be able to apply for it and wait for a response from the tour guide.
● What Can Go Wrong: Network loss
● Other Activities: Jerald will have to wait to be accepted and then will be added to a list of other customers that will be a part of the tour.
● System State on Completion: A new application will be sent to the tour guide.

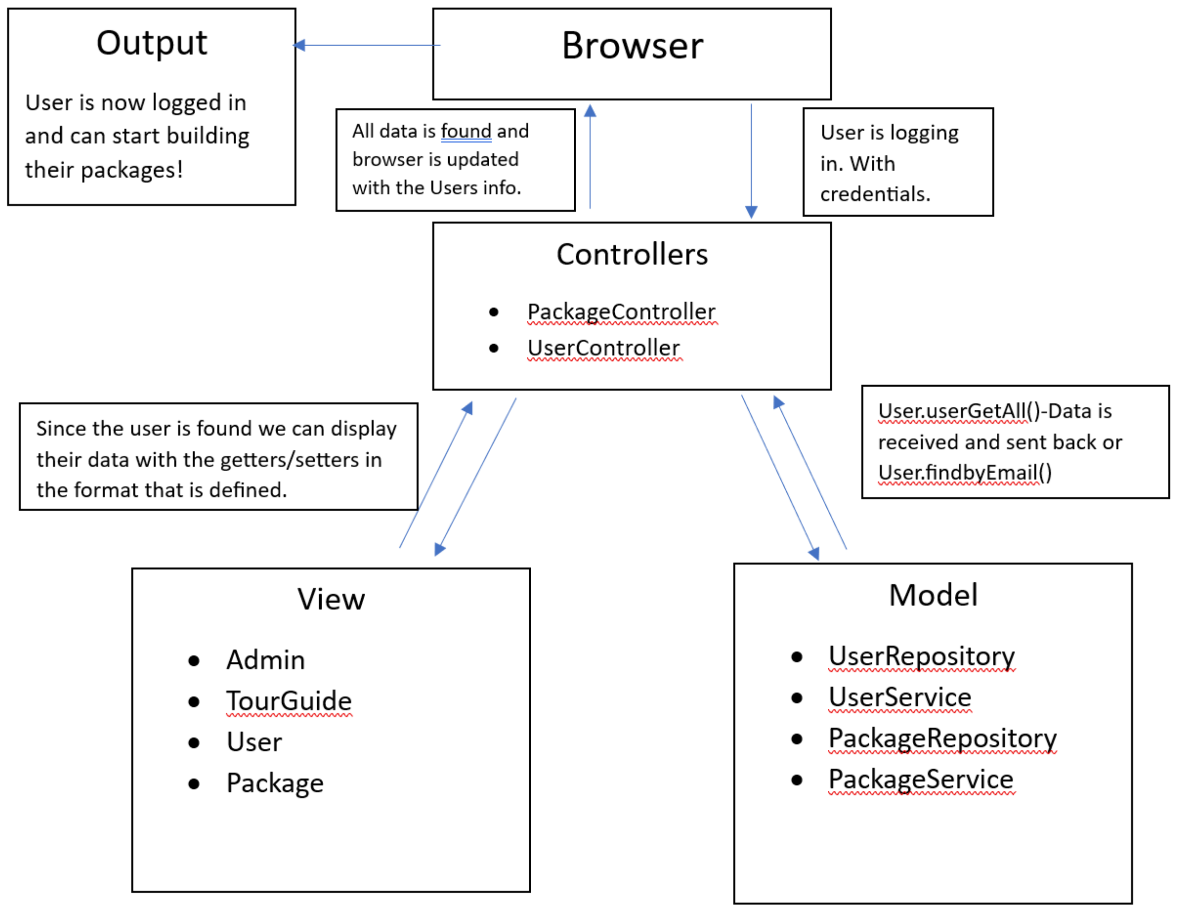
5.9.2 Actor: Bob Jones (Duncan Norment)

- Use-Case Name: Create Tour Package
● Initial Assumption: Bob is logged in.
● Normal: Bob will search through and add attractions/places to go in their tour package.
● What Can Go Wrong: Certain attractions or places are not present in the API used.
● Other Activities: Able to update/delete their packages.
● System State on Completion: A new tour package is added to the database.
- Use-Case Name: Accept/Decline Clients
● Initial Assumption: Bob is logged in.
● Normal: Bob will be able to see any new applications that were submitted by clients and he will be able to decide whether he wants to accept or decline them.
● What Can Go Wrong: Network issues, and someone might have accidentally applied to their service.
● Other Activities: Bob will be able to view applicant names.
● System State on Completion: Decision is made and client will either be stored or remain the same with a declined notification and the application will be deleted.

5.9.3 Actor: John Adkins (Kenneth Alvarado)
- Use-Case Name: Manage User Accounts
● Initial Assumption: John is logged.
● Normal: John will be able to create a new user, update, and delete them if desired.
● What Can Go Wrong: Network issues.
● Other Activities: John will also be able to access messages.
● System State on Completion: A new user added to the database, deleted, or updated.
- Use-Case Name: Customer Service
● Initial Assumption: John is logged in.
● Normal: John will be able to receive messages from tour guides/users about any concerns, and will act appropriately.
● What Can Go Wrong: Network issues and spam.
● Other Activities: John will be able to view others' messages.
● System State on Completion: Depends on any action performed while helping a user.

6. Design Documents

6.1 Software architecture


Output
User is now logged in and can start building their packages!
Browser
All data is found and browser is updated with the Users info.
User is logging in. With credentials.
Controllers
PackageController
UserController
Since the user is found we can display their data with the getters/setters in the format that is defined.
User.userGetAll()-Data is received and sent back or User.findbyEmail()
View
Admin
TourGuide
User
Package
Model
UserRepository
UserService
PackageRepository
PackageService

1. An existing User navigates to the login page, enters their credentials, and clicks a login button.
2. The browser will send a request to the UserConroller to access the User's account.
3. The UserController will ask the model to find the user's info, either in the UserRepository or UserService, depending on the method used like userGetAll() or findbyEmail().
4. Model, either UserRepository or UserService, returns the user's data to the controller.
5. If the data received from the model is sufficient then the controller will ask the view, in this case, the User section to get the new screen with that user's information and the user logged in.
6. The view takes the data received from the controller and renders the users new HTML

User

-userId: long

-email: string

-password: string

user(userId, email, password)

getters

setters

TourGuide

-tourGuideId: long

-email: string

-password: string

TourGuide(tourGuideId, email, password)

getters

setters

Admin

-adminId: long

-email: string

-password: string

Admin(adminId, email, password)

getters

setters

Package

-packageId: long

-name: String

-city: String

-description: String

package(packageId, name, city, description)

getters

setters

UserController

+userLogin(user: User)

+userCreate(user: User)

+userGetAll(model: Model)

+userGetId(id: long, mode: Model)

+userDelete(id: long, mode: Model)

UserService

+userGetAll()

+userGetId(id: long)

+userSave(user: User)

+userDelete(user: User)

UserRepository

repo: UserRepository

+findByEmail(email: String)

+search(keyword: String)

PackageController

+packageGetAll(model: Model)

+packageGet(id: long, mode: Model)

+packageCreate(package: Package)

+packageDelete(id: long, mode: Model)

+packageUpdate(package: Package)

+packageNewForum(model: Model)

+packageUpdateForum(id: long, mode: Model)

PackageService

+packageGetAll()

+packageGet(id: long)

+packageSave(package: Package)

+packageDelete(id: long)

PackageRepository

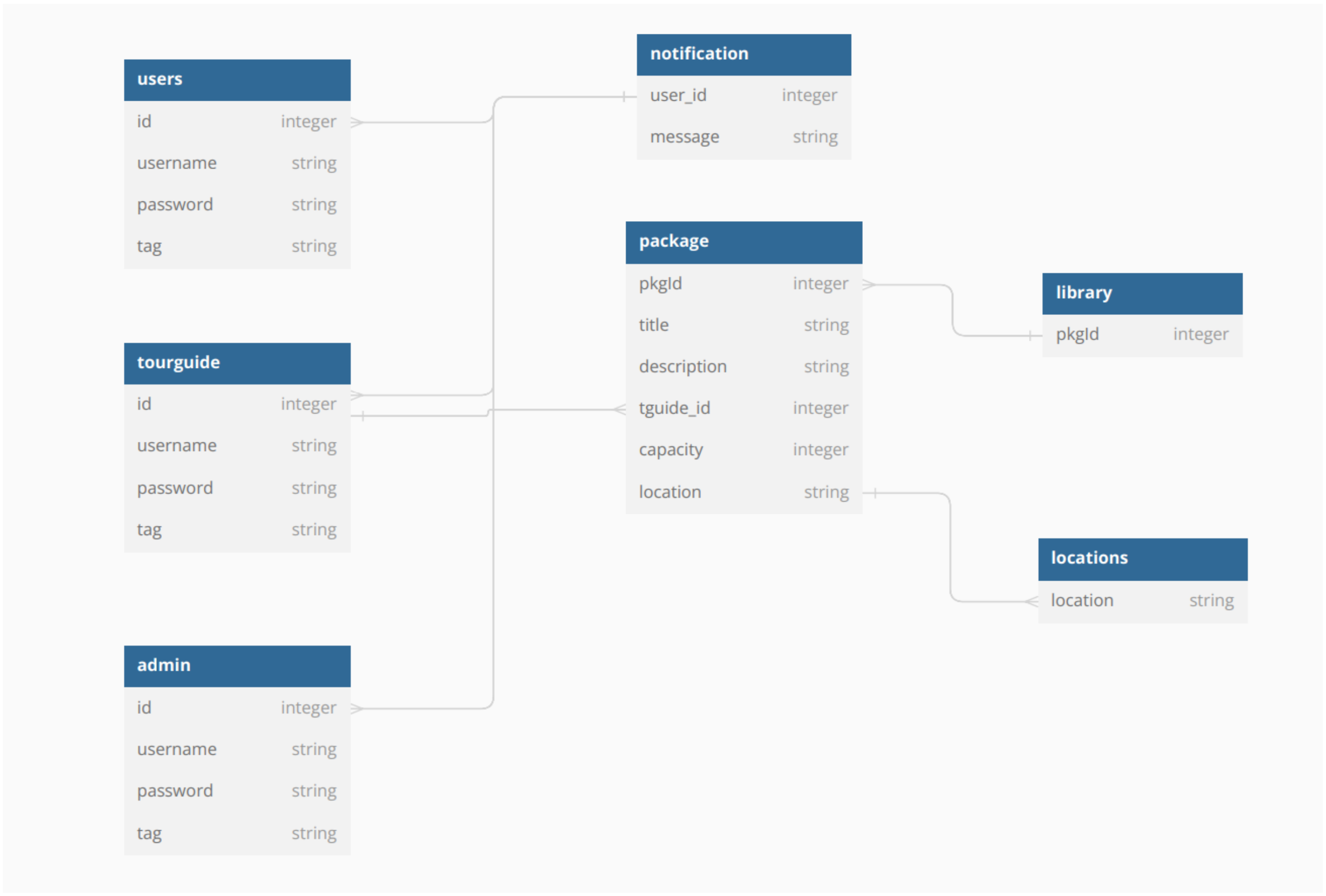
repo: PackageRepository

+findByName(name: String)

+search(keyword String)

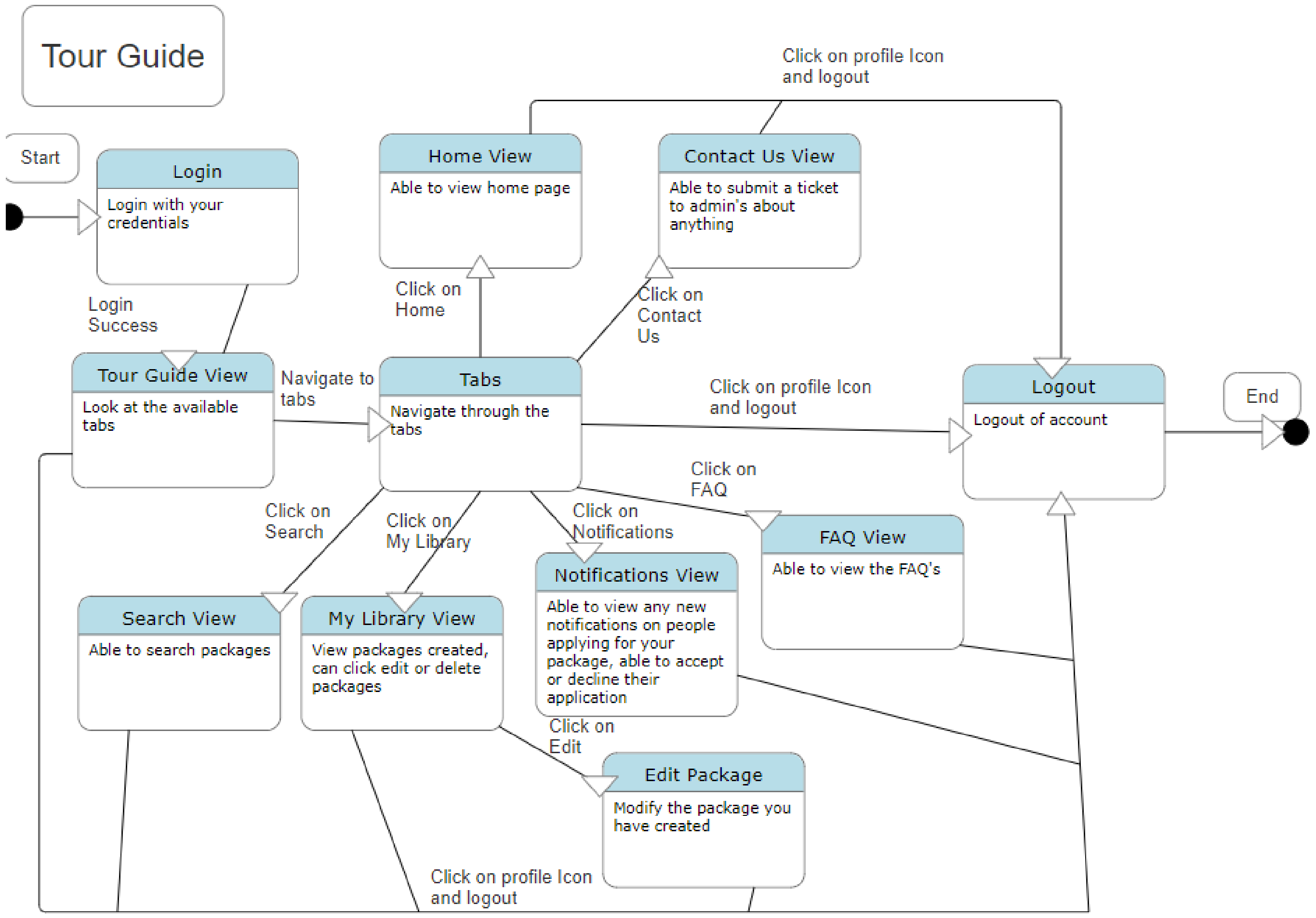
- A more detailed look is available in 6.4, where you can see all the methods and pages, including how they are linked.

6.2. High Level Schema

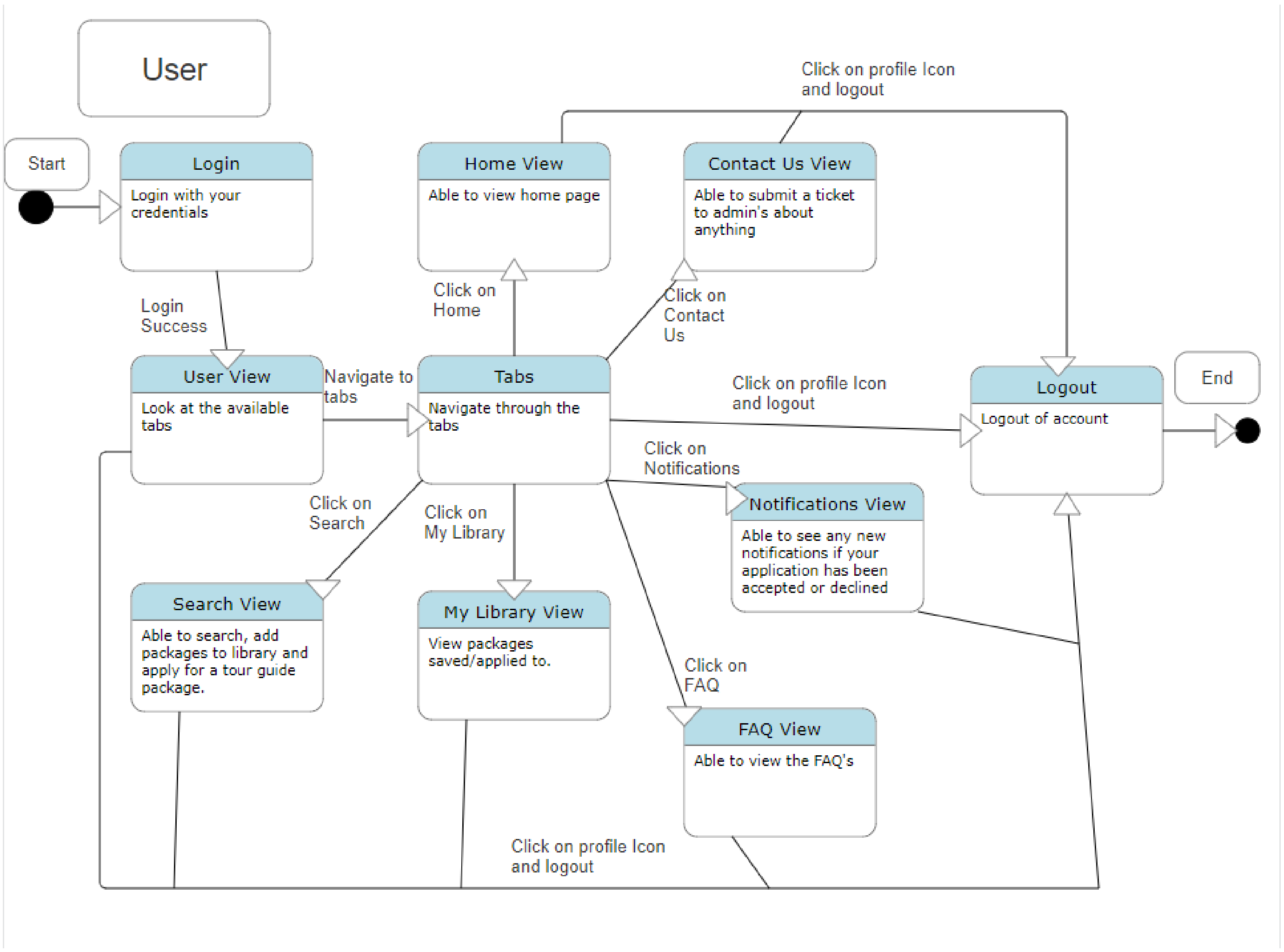


6.3. Software Design - State Machine Diagram

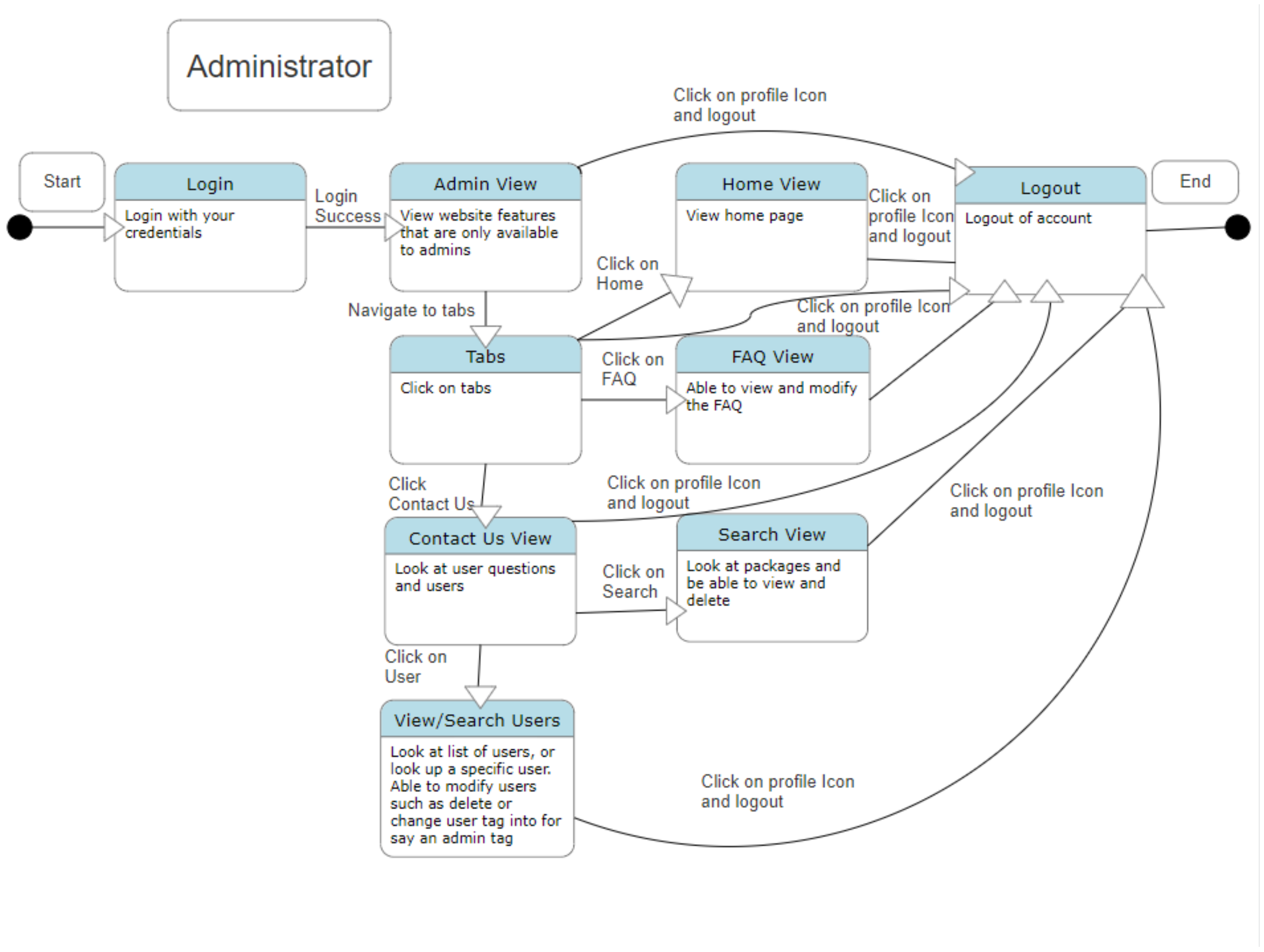
Tour Guide


Tour Guide
Start
Login
Login with your credentials
Login Success
Tour Guide View
Look at the available tabs
Navigate to tabs
Home View
Able to view home page
Click on Home
Contact Us View
Able to submit a ticket to admin's about anything
Click on Contact Us
Click on profile Icon and logout
Tabs
Navigate through the tabs
Click on profile Icon and logout
Logout
Logout of account
End
Click on FAQ
FAQ View
Able to view the FAQ's
Click on Search
Click on My Library
Click on Notifications
Search View
Able to search packages
My Library View
View packages created, can click edit or delete packages
Notifications View
Able to view any new notifications on people applying for your package, able to accept or decline their application
Click on Edit
Edit Package
Modify the package you have created
Click on profile Icon and logout

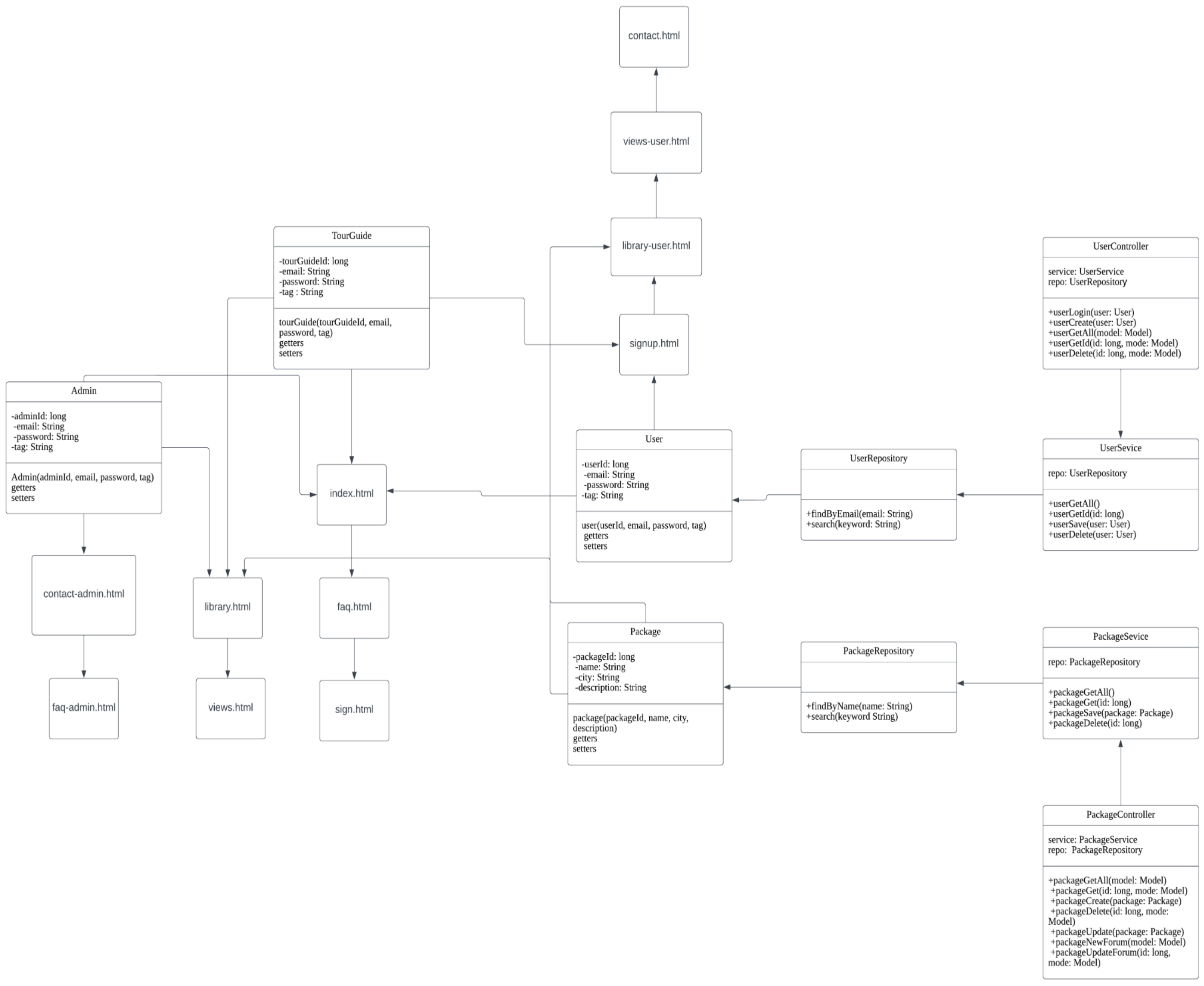
User


User
Start
Login
Login with your credentials
Login Success
Home View
Able to view home page
Click on Home
Contact Us View
Able to submit a ticket to admin's about anything
Click on Contact Us
Click on profile Icon and logout
User View
Look at the available tabs
Navigate to tabs
Tabs
Navigate through the tabs
Click on profile Icon and logout
Logout
Logout of account
End
Click on Notifications
Notifications View
Able to see any new notifications if your application has been accepted or declined
Click on Search
Click on My Library
Search View
Able to search, add packages to library and apply for a tour guide package.
My Library View
View packages saved/applied to.
Click on FAQ
FAQ View
Able to view the FAQ's
Click on profile Icon and logout

Admin


Administrator
Start
Login
Login with your credentials
Login Success
Admin View
View website features that are only available to admins
Click on profile Icon and logout
Home View
View home page
Click on profile Icon and logout
Logout
Logout of account
End
Click on Home
Navigate to tabs
Click on profile Icon and logout
Tabs
Click on tabs
Click on FAQ
FAQ View
Able to view and modify the FAQ
Click Contact Us
Click on profile Icon and logout
Click on profile Icon and logout
Contact Us View
Look at user questions and users
Click on Search
Search View
Look at packages and be able to view and delete
Click on User
View/Search Users
Look at list of users, or look up a specific user. Able to modify users such as delete or change user tag into for say an admin tag
Click on profile Icon and logout

6.4. UML Class Diagram


contact.html
views-user.html
library-user.html
signup.html
TourGuide
-tourGuideId: long
-email: String
-password: String
-tag : String
tourGuide(tourGuideId, email, password, tag)
getters
setters
UserController
service: UserService
repo: UserRepository
+userLogin(user: User)
+userCreate(user: User)
+userGetAll(model: Model)
+userGetId(id: long, mode: Model)
+userDelete(id: long, mode: Model)
Admin
-adminId: long
-email: String
-password: String
-tag: String
Admin(adminId, email, password, tag)
getters
setters
index.html
User
-userId: long
-email: String
-password: String
-tag: String
user(userId, email, password, tag)
getters
setters
UserRepository
+findByEmail(email: String)
+search(keyword: String)
UserSevice
repo: UserRepository
+userGetAll()
+userGetId(id: long)
+userSave(user: User)
+userDelete(user: User)
contact-admin.html
library.html
faq.html
faq-admin.html
views.html
sign.html
Package
-packageId: long
-name: String
-city: String
-description: String
package(packageId, name, city, description)
getters
setters
PackageRepository
+findByName(name: String)
+search(keyword String)
PackageSevice
repo: PackageRepository
+packageGetAll()
+packageGet(id: long)
+packageSave(package: Package)
+packageDelete(id: long)
PackageController
service: PackageService
repo: PackageRepository
+packageGetAll(model: Model)
+packageGet(id: long, mode: Model)
+packageCreate(package: Package)
+packageDelete(id: long, mode: Model)
+packageUpdate(package: Package)
+packageNewForum(model: Model)
+packageUpdateForum(id: long, mode: Model)