Jessica Damasco Ty
Sri Rama Mohana Rao Medisetti
Martin Sarmiento
Aditya Vadrevu
Hemanth Vemula

SE 577 Group 1

Use Case 1: **Search tickets**	
Description	This use case describes how the user can search tickets by providing the following information: origin, destination, and date.
Primary Actor(s)	User
Secondary Actor(s)	
Preconditions	The user is logged into the application.
Flow of Events	1. The application displays a form for origin, destination, and date. 2. The user provides origin, destination, and date. 3. The database is queried. 4. The application displays the available tickets corresponding to the provided information.
Alternative Flows	
Postconditions	The application displays tickets which the user can choose from.
User Stories	1. The user wants to search for all the available tickets for a trip that they are planning, so the user inputs their origin location, the location of the desired destination for the trip, and the date that the trip is planned on. The system then displays all the available tickets for the following trip locations on the following date to the user. 2. The user wants to plan a day trip ahead of time during their christmas break, so the user checks for all the dates during their break to see all the different tickets she can get to buy. 3. The user after having input their desired locations and dates are shown a list of all the tickets with the given times.

Use Case 2: **Display available itineraries**	
Description	This use case describes how the application can display the available itineraries according to the user's search.
Primary Actor(s)	User
Secondary Actor(s)	
Preconditions	The user is logged into the application and has completed Use Case 1: Search tickets.
Flow of Events	1. The application displays the available itineraries corresponding to the completed search. 2. The user selects from the following sorting methods: Departure Time, Arrival Time, Travel Time, and Price. 3. The application displays the available itineraries according to the user selected sorting method.
Alternative Flows	
Postconditions	The available itineraries are sorted accordingly.
User Stories	1. The user wants to sort the list of all the tickets they searched for by departure time because they want to leave at the latest possible time to make the most of their vacation, so they sort the items by departure time in order for the user to choose the latest time of departure. 2. The user has an important meeting to attend to at the destination of their trip, so the user decides to arrive earlier than the meeting time. They sort the tickets by arrival time to make sure that they will arrive early for their meeting. 3. The user is on a budget and wants to try visiting a location, so the user tries sorting the available tickets by price in order to get the cheapest one available.

Use Case 3: **Add to cart**	
Description	This use case describes how the user can choose a ticket from the list of available itineraries to buy.
Primary Actor(s)	User
Secondary Actor(s)	
Preconditions	The user is logged into the application and has completed Use Case 1: Search tickets.
Flow of Events	1. The application displays the available itineraries corresponding to the completed search. 2. The user clicks on the + button. 3. The application increments the number of tickets to be bought. 4. The user clicks on the Add to cart button.
Alternative Flows	Instead of step 2, the user decides to decrement the number of tickets to be bought, 1. The user clicks on the - button. 2. The application decrements the number of tickets to be bought. 3. The user clicks on the Add to cart button.
Postconditions	The ticket(s) are added to the user’s cart.
User Stories	1. The user has decided on the details of their trip, but they still want to keep buying additional tickets. So the user adds the ticket to their cart while they keep browsing. 2. The user is planning a long trip consisting of multiple tickets, so the user adds all of those tickets for different itineraries into their cart. 3. The user is getting tickets for their family, so the user has to buy multiple duplicate tickets for each family member. They then add all the duplicate tickets into the cart.

Use Case 4: **Manage cart**	
Description	This use case describes how the user can view, clear, or modify their cart.
Primary Actor(s)	User
Secondary Actor(s)	
Preconditions	The user is logged into the application and is in the Cart page.
Flow of Events	1. After the user has clicked on the Cart tab, the application displays a list of tickets added by the user. 2. The user clicks on the + or - button to increment or decrement the number of tickets to be purchased. 3. The application edits the number of tickets to be purchased accordingly.
Alternative Flows	Instead of step 2, the user decides to delete a ticket, 1. The user clicks on the Delete button corresponding to the ticket to be deleted. 2. The user confirms after being prompted to confirm.
Postconditions	The cart reflects the changes the user has decided for the tickets to be purchased.
User Stories	1. The user changes their mind about the trip that they were planning, so the user having placed some tickets into their cart wants to remove a certain ticket from their cart. The user then presses the Delete button next to the corresponding ticket they want to remove. 2. The user having mistakenly added a duplicate ticket into their cart wants to reduce the ticket quantity backt to one, so the user presses the - button next to the duplicated ticket to reduce it to one. 3. The user having finished adding all the desired tickets into their cart wants to proceed to checkout to purchase all the tickets, so they press the CheckOut button to do so.

Use Case 5: **Check out**	
Description	This use case describes how the user can purchase and book the tickets present in the cart.
Primary Actor(s)	User
Secondary Actor(s)	
Preconditions	The user is logged into the application and has completed Use Case 3: Add to cart and Use Case 4: Manage cart.
Flow of Events	1. After the user clicks on the Check out button from the Cart page, the application displays a summary of the cart items and a form for the following information: first name, last name, address, and phone number. 2. The user provides the corresponding information. 3. The user selects a payment method. 4. The application displays forms corresponding to the selected payment method. 5. The user provides the corresponding information for the payment method. 6. The user clicks on the Check out button.
Alternative Flows	If the user provides invalid information or payment details, 1. The forms are emptied and the application displays a message that the provided information is invalid. The tickets remain unbooked.
Postconditions	The user makes the payment and books the tickets.
User Stories	1. The user is picky when it comes to purchasing things online. They prefer to use their desired payment option, so once they proceed to checkout they choose the option that coincides with their desired payment option to purchase their tickets. 2. The user having bought a ticket before on the website has moved to another address due to work, so once they check out their tickets for paying they can input the new address connected to their new card. 3. The user having recently lost their phone, wants to redirect the recite message on another phone number. So on checkout they input the new phone number in order to receive the notification

Use Case 6: **Sign in/Register**	
Description	This use case describes how the user can sign in or register to the application.
Primary Actor(s)	User
Secondary Actor(s)	
Preconditions	If performing sign in, the user owns an existing account.
Flow of Events	If performing sign in, 1. The application displays a form for email and password. 2. The user provides the corresponding information. 3. The user clicks on the Sign in button. If performing register, 1. The application displays a form for the following information: first name, last name, address, email address, phone number, and password. 2. The user provides the corresponding information. 3. The user clicks on the Register button.
Alternative Flows	If the user provides invalid information with the wrong credentials, 1. The forms are emptied and the application displays a message that the provided information is invalid.
Postconditions	1. After successfully signing in, the user is redirected to the home page of the application. 2. After successfully registering, the user is redirected to the login page. 3. After unsuccessfully signing in or registering, the corresponding message alert page will be displayed.
User Stories	1. The user wants to create an account on the website, so they press the Register button. They then fill up the corresponding required fields to create the account, once the user has completed all the required fields with valid inputs the account will be created. 2. The user who has already created an account on the website wants to login. They enter the email address and password that is linked to their account, and they then are logged into the system. 3. The user wants to login into the system; however, they mistakenly mistyped their credentials. So the user having been notified by the system that what they entered is wrong has to retype their credentials into the system.

Use Case 7: **Manage user profile**	
Description	This use case describes how the user can modify his profile information.
Primary Actor(s)	User
Secondary Actor(s)	
Preconditions	The user is logged into the application and is in the Profile page.
Flow of Events	1. The application displays a set of forms corresponding to the following information of the user: first name, last name, email address, password, address, and phone number. 2. Through the set of forms, the user edits a field. 3. The user clicks on the Save button..
Alternative Flows	If the user is not able to update his profile then the user can reach out to the support team for further help.
Postconditions	After the user successfully updates and saves the edited profile information, the updated information should be reflected in the user profile.
User Stories	1. The user has recently changed their email address, so they want to change the email address registered on their account. The user then goes through their profile manager, and edits the field for their email address to reflect their new email address. 2. The user wants to update all of the passwords they use for all their accounts, so the user quickly logs into their profile to change the password connected on their account. 3. The user having edited their profile to their desired state wants to save all the changes they made, so they press the Save button to finalize all their edits to their profile.

Use Case 8: **Manage booked itineraries**	
Description	This use case describes how the user can manage booked itineraries.
Primary Actor(s)	User
Secondary Actor(s)	
Preconditions	The user is logged into the application and is in the Booked itineraries page.
Flow of Events	1. The application displays the user’s booked itineraries. 2. The user selects the itinerary to be edited. 3. The application displays the details of the ticket. 4. The user clicks on the Delete button.
Alternative Flows	
Postconditions	The deleted itineraries will no longer be booked under the user’s profile.
User Stories	1. The user having bought the tickets already wants to be able to view all their pending itineraries to remind themselves of their plans. So the user goes through all the booked itineraries that they have to see what they have planned. 2. The user is having some doubts on one of the trips they have booked, so they go through the list of all the itineraries that they have booked and quickly cancel that ticket. 3. The user suddenly wants to change the plan of one of their booked itineraries, so they go through the itineraries manager and edit the details of the booked itineraries, which will incur any penalty for sudden changes in ticket detail. 4. The user wants to recreate a past trips experience, so they go through the list of all their booked itineraries which include all the present, and past booked itineraries. 5. The user wants to download their whole itinerary for a trip to be able to save it for them to check on offline.

Use Case 9: **Manage orders**	
Description	This use case describes how the admin can set the status of the orders.
Primary Actor(s)	Admin
Secondary Actor(s)	
Preconditions	The admin is logged into the application using administration credentials.
Flow of Events	1. The use case starts when the admin enters the Manage Orders page through the administration page. 2. The application displays a list of orders with the following information: transaction ID, user's first name, user's last name, origin, and destination. 3. Through the list of orders, the admin clicks on the Details button of the corresponding order that will be edited. 4. The application displays the order details with a form to edit the order status between the following states: Processed, Cancelled, Refunded, Moved. 5. The admin selects an order status. 6. The admin clicks on the Save button.
Alternative Flows	
Postconditions	The order edited will reflect the new order status to the respective user.
User Stories	1. The order of a user must be cancelled due to the itinerary being cancelled, so the admin sets the order status to cancelled. 2. The order of a user has been cancelled and a refund has been placed, so the admin sets the order status to refunded. 3. The order of a user includes a trip whose arrival time has been moved due to traffic construction, so the admin sets the order status to moved.

Use Case 10: **Manage users**	
Description	This use case describes how the admin can edit the type of group a user is in.
Primary Actor(s)	Admin
Secondary Actor(s)	
Preconditions	The admin is logged into the application using administration credentials.
Flow of Events	1. The use case starts when the admin enters the Manage Users page through the administration page. 2. The application displays a list of users with the following information: ID, first name, last name, and groups. 3. Through the list of users, the admin selects from a drop down menu of the types of groups. 4. The admin selects a group type. 5. The admin clicks on the Save button.
Alternative Flows	When the admin selects Create a New User button instead of step 2, 1. The admin enters details in a form prompting for user first name, user last name, ID, and type of group. 2. The admin clicks on the Save button. When the admin selects the Delete button corresponding to a specific user instead of step 2, 1. The admin confirms after being prompted to confirm. 2. The respective user is deleted.
Postconditions	The group type edited will reflect the new group type to the other admins.
User Stories	1. The admin gets a request to change the group of one of the users in the system, so the admin looks for that specific user through the list of users, and changes the group of the following user. 2. The admin is instructed to create a user account for the business shareholders, so the admin creates the new user with all the relevant access required for a business shareholder. 3. The admin wants to clear up the list of users on the system, so

	the admin removes all the users that are no longer being used.

Use Case 11: **Manage customers**	
Description	This use case describes how the admin can edit a specific customer's profile details.
Primary Actor(s)	
Secondary Actor(s)	
Preconditions	The admin is logged into the application using administration credentials.
Flow of Events	1. The use case starts when the admin enters the Manage Customers page through the administration page. 2. A list of all the customer accounts are then displayed to the admin 3. Each customer listed will have their corresponding account ID, Email Address, and the customer's First and Last name. 4. The admin can then click on the Detail button which can be found next to each listed customer. 5. From there the system displays information specific to the account such as the age, birthday, password, contact information, first Name, last Name, and their email address. 6. The admin can then edit any of the given fields. 7. After all the desired changes have been made the admin can save all the changes by pressing the Save button
Alternative Flows	When the admin selects to delete the account instead of step 4, 1. The admin clicks the Delete button 2. The system will remove the deleted account
Postconditions	
User Stories	1. The admin wants to delete certain customer accounts from the system, so the admin looks through the list of customer accounts and deletes those accounts. 2. The admin needs to check the details of a certain customer account to help the customer retrieve their account, so the user looks for the customer account through the list of accounts and checks the details for that account.

Use Case 12: **Manage groups**	
Description	This use case describes how the admin edits the types of users.
Primary Actor(s)	Admin
Secondary Actor(s)	
Preconditions	The admin is logged into the application using administration credentials.
Flow of Events	1. The use case starts when the admin enters the Manage Groups page. 2. The application displays a list of groups for users. 3. Through the list of groups, the admin clicks on the Details button of the corresponding group that will be edited. 4. The application displays a form to edit the name of the group. 5. The admin edits the name of the group through a text form. 6. The admin clicks on the Save button.
Alternative Flows	After step 2, if the admin clicks on Create a new group button, 1. The application displays a new group with a text form for the name of the group. 2. The admin enters the name of the group. 3. The admin clicks on the Save button.
Postconditions	The group name will be updated accordingly.
User Stories	1. The admin wants to find out all the existing groups available to the system, so the admin checks out the conveniently displayed list provided by the system. 2. The admin wants to add a new group for users into the system, so the admin inputs all the necessary fields to create the group, and partitions the level of accessibility. 3. The admin not wanting to lose all the edits that the admin has made on a group, wants to save all their progress by pressing the Save button.

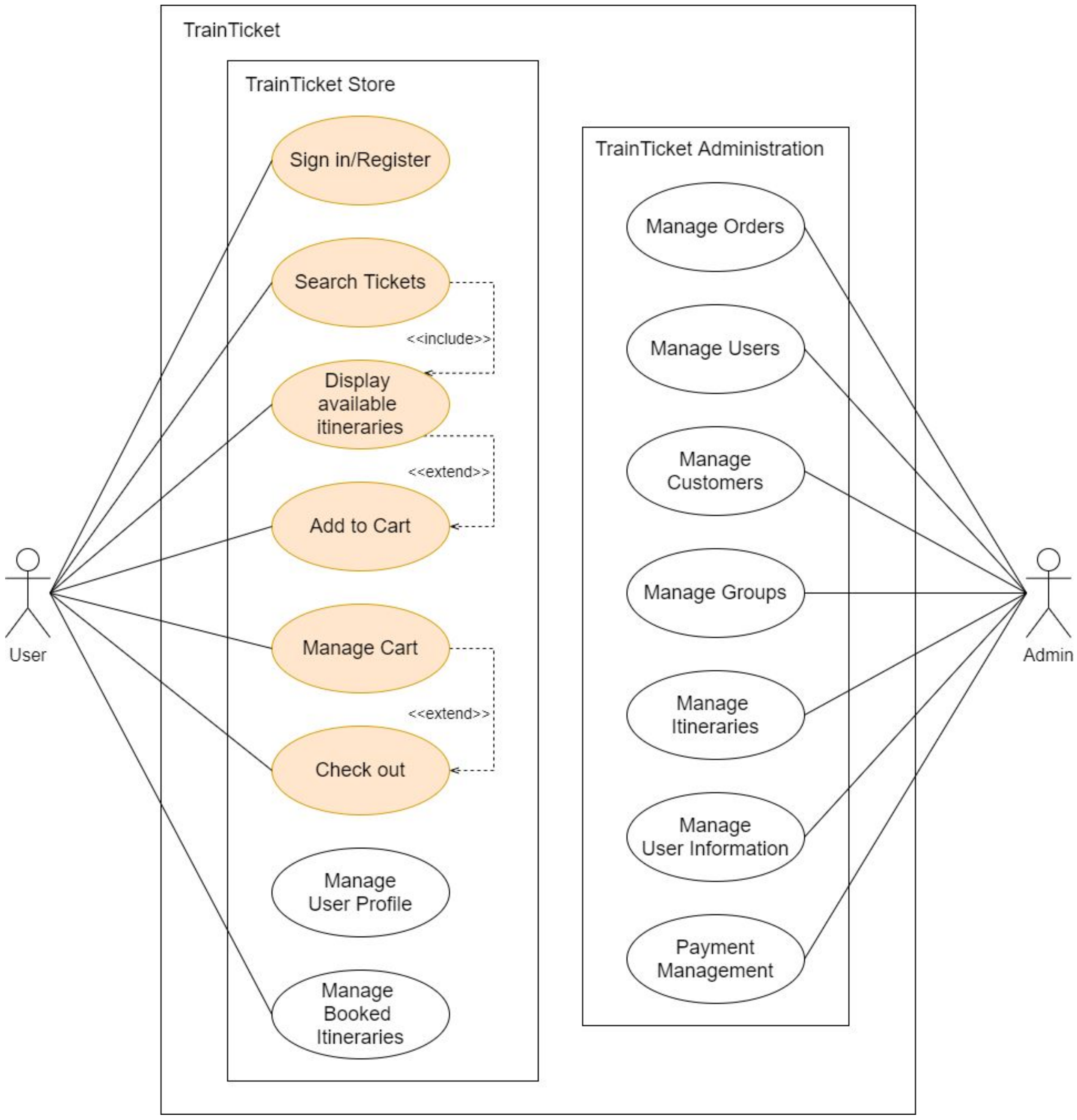
Use Case 13: **Manage itineraries**	
Description	This use case describes how the admin can edit or delete specific itineraries.
Primary Actor(s)	Admin
Secondary Actor(s)	
Preconditions	The admin is logged into the application using administration credentials.
Flow of Events	1. The use case starts when the admin enters the Manage Itineraries page through the administration page. 2. The application displays a list of orders displaying the following information: trip ID, origin, destination, departure time, arrival time, and date. 3. Through the list of orders, the admin clicks on the Details button of the corresponding itinerary that will be edited. 4. The admin is presented with a form to edit the origin, destination, departure time, arrival time, date, and traveler capacity. 5. The admin edits the corresponding field with the respective data. 6. The admin clicks on the Save button.
Alternative Flows	After step 3, the admin is also presented with a delete button. 1. The admin clicks on the Delete button. 2. The server prompts the admin to confirm this action. 3. The admin confirms. 4. The respective itinerary is deleted.
Postconditions	The itinerary edited will reflect the changes to users purchasing tickets.
User Stories	1. A certain itinerary will no longer be profitably feasible to offer due to the trip not regularly not having enough travelers, so the admin deletes the itinerary to cancel it. 2. A certain itinerary does not have enough travelers in the middle of the night, so the admin changes the time to earlier in the evening. 3. A certain itinerary destination needs to be changed to a station relatively close due to an accident, so the admin changes the destination of the trip.

Use Case 14: **Manage user information**	
Description	This use case describes how the admin can edit the admin profile details.
Primary Actor(s)	Admin
Secondary Actor(s)	
Preconditions	The admin is logged into the application using administration credentials.
Flow of Events	1. The user case starts when the admin enters the User Information page through the administration page. 2. The application displays a set of forms corresponding to the following information: first name, last name, email address, password, and group type 3. Through the set of forms, the admin edits a field. 4. The admin clicks on the Save button.
Alternative Flows	
Postconditions	The edited fields will reflect in the admin's profile.
User Stories	1. The admin wants to change the email address of one of the users of the system, so they look through the list of users and find the user they want to change the email address of, and edit in the new email address. 2. The admin gets a request to change the passwords of one of the users of the system because the following user forgot theirs, so the admin goes through that users profile to change the user's password. 3. The admin wants to check the details of each account registered to the users of the system, so the admin browses through the list and checks all the details of each user.

Use Case 15: **Payment management**	
Description	This use case describes how the admin can enable or disable payment methods.
Primary Actor(s)	Admin
Secondary Actor(s)	
Preconditions	The admin is logged into the application using administration credentials.
Flow of Events	1. The use case starts when the admin enters the Payment Management page through the administration page. 2. The application displays a set of radio buttons corresponding to several payment methods. 3. Through the set of radio buttons, the admin clicks on a payment method without a check mark. 4. The respective payment method is enabled in the application.
Alternative Flows	Instead of step 3, if the admin clicks on a payment method with a check mark, 4. The respective payment method is disabled.
Postconditions	The enabled payment methods will be available to users.
User Stories	1. The admin finds out that one of the payment methods has been buggy, so the admin quickly disabled the payment method so that any customer will not be able to choose it as an option. 2. The admin wants to edit a certain type of payment method in the system, so they go through that payment method’s details to edit some of its parameters. 3. The admin wants to look at all the available payment methods in order to check whether the system is accepting all desired payment options

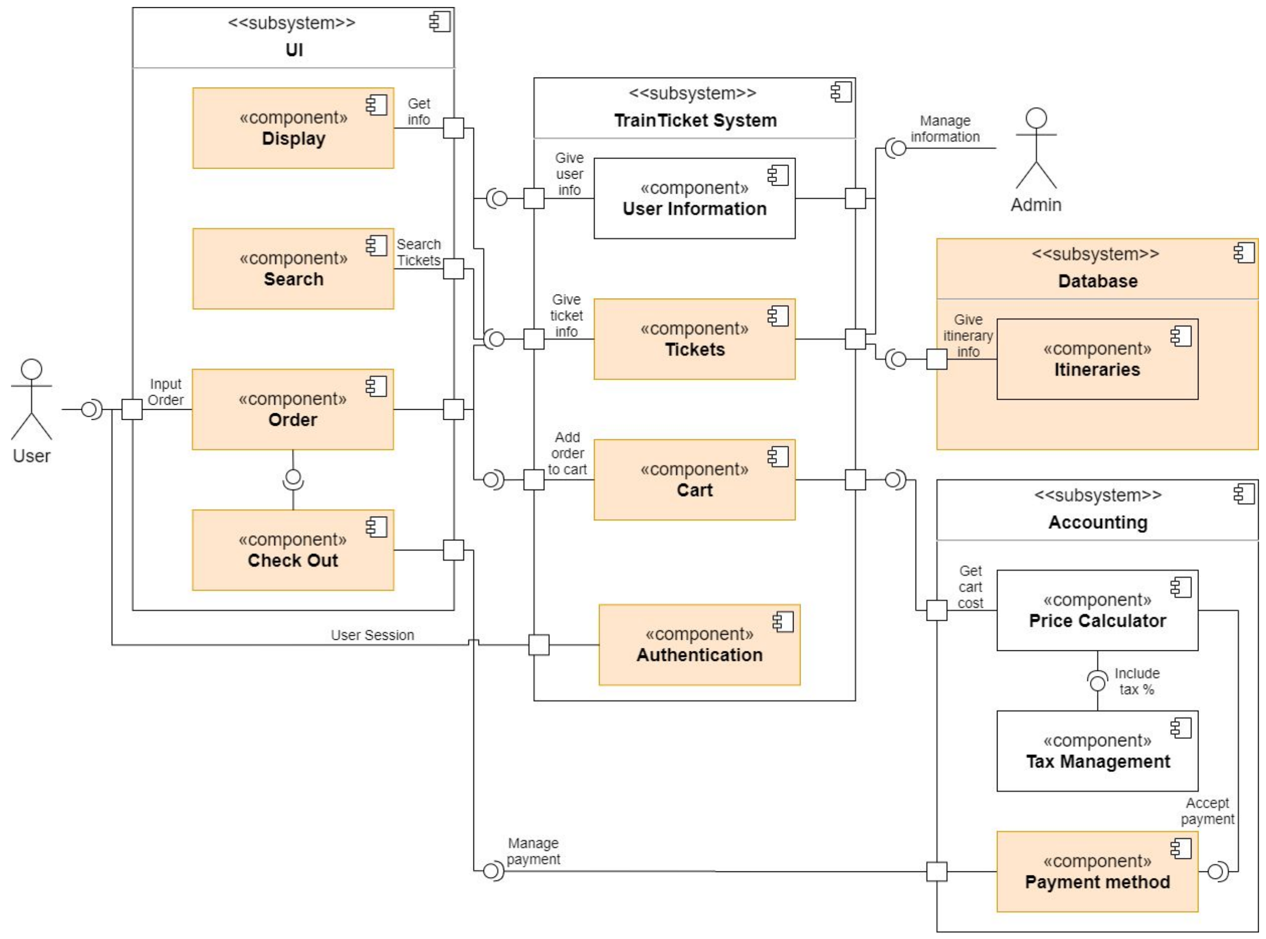
Use Case Diagram

Legend: Orange highlighted cases are implemented

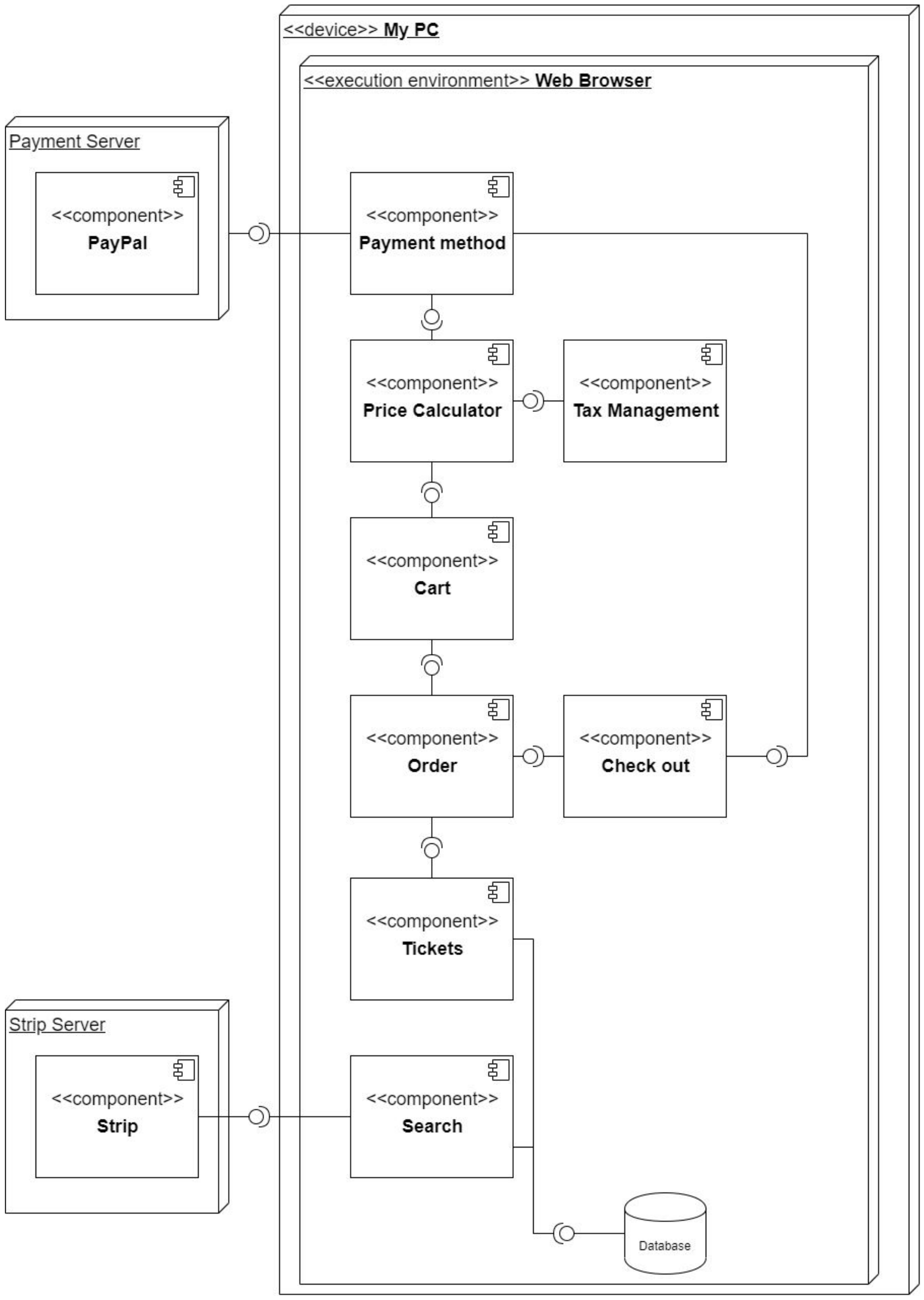


Component Diagram

Legend: Orange highlighted cases are implemented

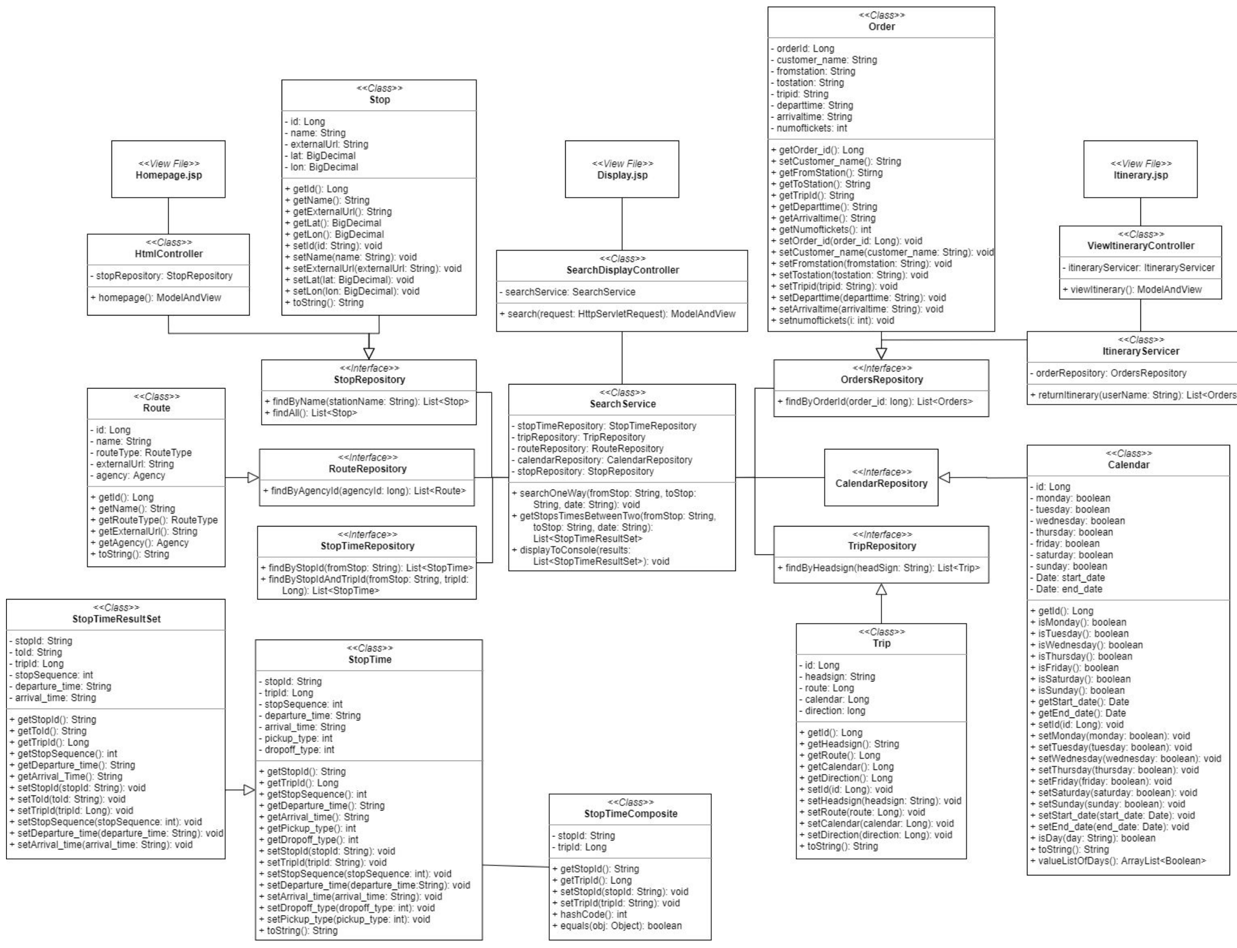


Deployment Diagram

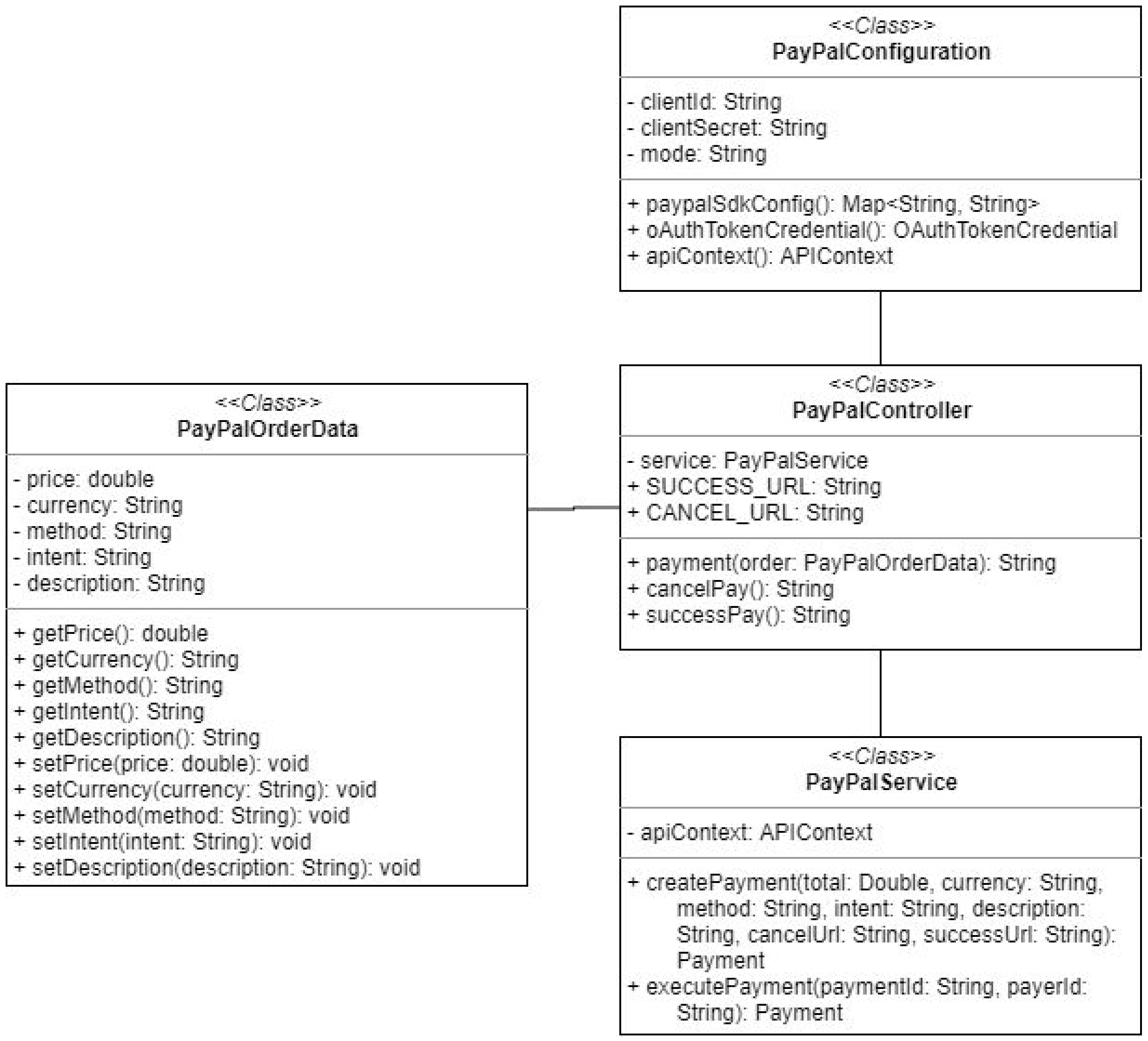


Class Diagrams

Search

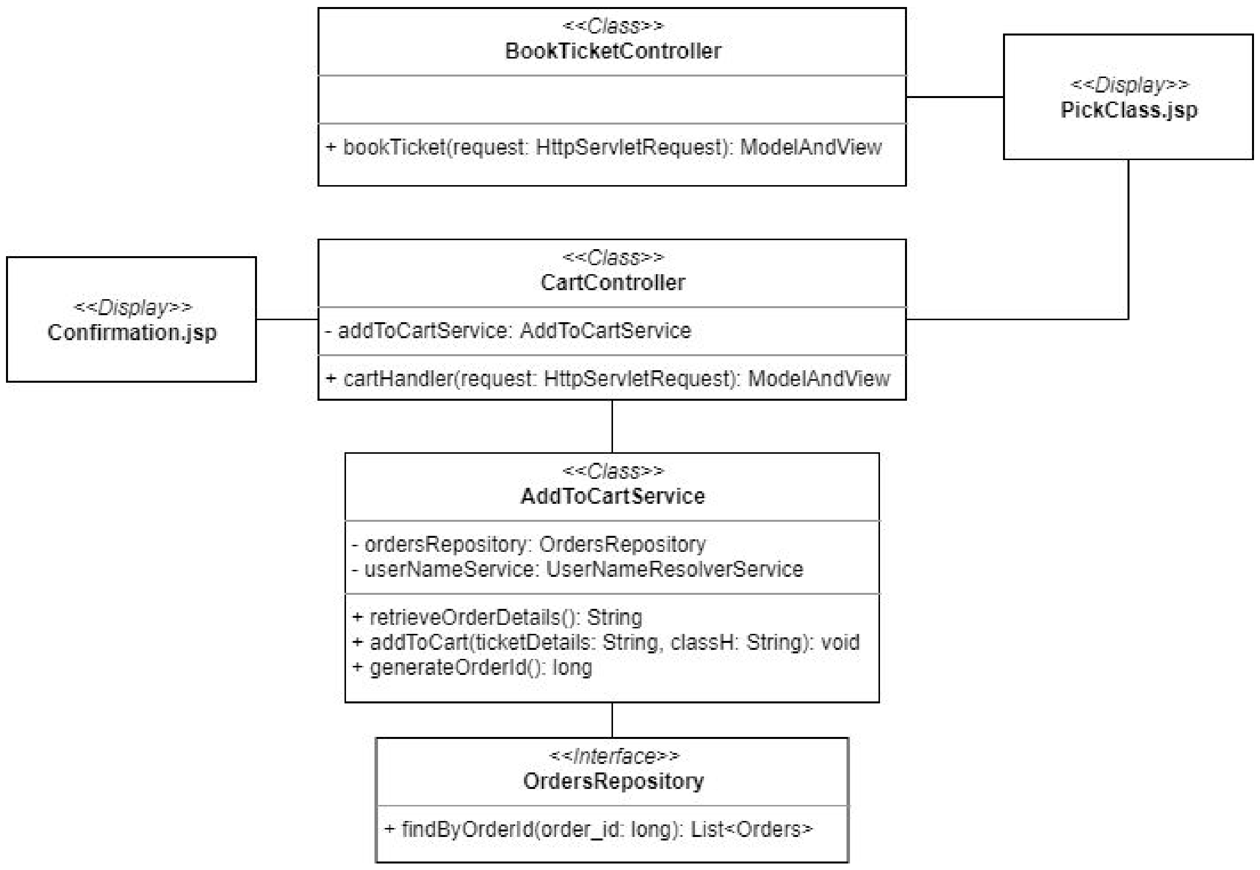

<<View File>>
Homepage.jsp
<<Class>>
HtmlController
- stopRepository: StopRepository
+ homepage(): ModelAndView
<<Class>>
Stop
- id: Long
- name: String
- externalUrl: String
- lat: BigDecimal
- lon: BigDecimal
+ getId(): Long
+ getName(): String
+ getExternalUrl(): String
+ getLat(): BigDecimal
+ getLon(): BigDecimal
+ setId(id: String): void
+ setName(name: String): void
+ setExternalUrl(externalUrl: String): void
+ setLat(lat: BigDecimal): void
+ setLon(lon: BigDecimal): void
+ toString(): String
<<View File>>
Display.jsp
<<Class>>
SearchDisplayController
- searchService: SearchService
+ search(request: HttpServletRequest): ModelAndView
<<Class>>
Order
- orderId: Long
- customer_name: String
- fromstation: String
- tostation: String
- tripid: String
- departtime: String
- arrivaltime: String
- numoftickets: int
+ getOrder_id(): Long
+ setCustomer_name(): String
+ getFromStation(): Stirng
+ getToStation(): String
+ getTripId(): String
+ getDeparttime(): String
+ getArrivaltime(): String
+ getNumoftickets(): int
+ setOrder_id(order_id: Long): void
+ setCustomer_name(customer_name: String): void
+ setFromstation(fromstation: String): void
+ setTostation(tostation: String): void
+ setTripid(tripid: String): void
+ setDeparttime(departtime: String): void
+ setArrivaltime(arrivaltime: String): void
+ setnumoftickets(i: int): void
<<View File>>
Itinerary.jsp
<<Class>>
ViewItineraryController
- itineraryServicer: ItineraryServicer
+ viewItinerary(): ModelAndView
<<Interface>>
StopRepository
+ findByName(stationName: String): List<Stop>
+ findAll(): List<Stop>
<<Class>>
SearchService
- stopTimeRepository: StopTimeRepository
- tripRepository: TripRepository
- routeRepository: RouteRepository
- calendarRepository: CalendarRepository
- stopRepository: StopRepository
+ searchOneWay(fromStop: String, toStop: String, date: String): void
+ getStopsTimesBetweenTwo(fromStop: String, toStop: String, date: String): List<StopTimeResultSet>
+ displayToConsole(results: List<StopTimeResultSet>): void
<<Interface>>
OrdersRepository
+ findByOrderId(order_id: long): List<Orders>
<<Class>>
ItineraryServicer
- orderRepository: OrdersRepository
+ returnItinerary(userName: String): List<Orders>
<<Class>>
Route
- id: Long
- name: String
- routeType: RouteType
- externalUrl: String
- agency: Agency
+ getId(): Long
+ getName(): String
+ getRouteType(): RouteType
+ getExternalUrl(): String
+ getAgency(): Agency
+ toString(): String
<<Interface>>
RouteRepository
+ findByAgencyId(agencyId: long): List<Route>
<<Interface>>
CalendarRepository
<<Class>>
Calendar
- id: Long
- monday: boolean
- tuesday: boolean
- wednesday: boolean
- thursday: boolean
- friday: boolean
- saturday: boolean
- sunday: boolean
- Date: start_date
- Date: end_date
+ getId(): Long
+ isMonday(): boolean
+ isTuesday(): boolean
+ isWednesday(): boolean
+ isThursday(): boolean
+ isFriday(): boolean
+ isSaturday(): boolean
+ isSunday(): boolean
+ getStart_date(): Date
+ getEnd_date(): Date
+ setId(id: Long): void
+ setMonday(monday: boolean): void
+ setTuesday(tuesday: boolean): void
+ setWednesday(wednesday: boolean): void
+ setThursday(thursday: boolean): void
+ setFriday(friday: boolean): void
+ setSaturday(saturday: boolean): void
+ setSunday(sunday: boolean): void
+ setStart_date(start_date: Date): void
+ setEnd_date(end_date: Date): void
+ isDay(day: String): boolean
+ toString(): String
+ valueListOfDays(): ArrayList<Boolean>
<<Interface>>
StopTimeRepository
+ findByStopId(fromStop: String): List<StopTime>
+ findByStopIdAndTripId(fromStop: String, tripId: Long): List<StopTime>
<<Interface>>
TripRepository
+ findByHeadsign(headSign: String): List<Trip>
<<Class>>
StopTimeResultSet
- stopId: String
- toId: String
- tripId: Long
- stopSequence: int
- departure_time: String
- arrival_time: String
+ getStopId(): String
+ getToId(): String
+ getTripId(): Long
+ getStopSequence(): int
+ getDeparture_time(): String
+ getArrival_Time(): String
+ setStopId(stopId: String): void
+ setToId(toId: String): void
+ setTripId(tripId: Long): void
+ setStopSequence(stopSequence: int): void
+ setDeparture_time(departure_time: String): void
+ setArrival_time(arrival_time: String): void
<<Class>>
StopTime
- stopId: String
- tripId: Long
- stopSequence: int
- departure_time: String
- arrival_time: String
- pickup_type: int
- dropoff_type: int
+ getStopId(): String
+ getTripId(): Long
+ getStopSequence(): int
+ getDeparture_time(): String
+ getArrival_time(): String
+ getPickup_type(): int
+ getDropoff_type(): int
+ setStopId(stopId: String): void
+ setTripId(tripId: String): void
+ setStopSequence(stopSequence: int): void
+ setDeparture_time(departure_time:String): void
+ setArrival_time(arrival_time: String): void
+ setDropoff_type(dropoff_type: int): void
+ setPickup_type(pickup_type: int): void
+ toString(): String
<<Class>>
Trip
- id: Long
- headsign: String
- route: Long
- calendar: Long
- direction: long
+ getId(): Long
+ getHeadsign(): String
+ getRoute(): Long
+ getCalendar(): Long
+ getDirection(): Long
+ setId(id: Long): void
+ setHeadsign(headsign: String): void
+ setRoute(route: Long): void
+ setCalendar(calendar: Long): void
+ setDirection(direction: Long): void
+ toString(): String
<<Class>>
StopTimeComposite
- stopId: String
- tripId: Long
+ getStopId(): String
+ getTripId(): Long
+ setStopId(stopId: String): void
+ setTripId(tripId: String): void
+ hashCode(): int
+ equals(obj: Object): boolean

Checkout



<<Class>>
PayPalConfiguration
- clientId: String
- clientSecret: String
- mode: String
+ paypalSdkConfig(): Map<String, String>
+ oAuthTokenCredential(): OAuthTokenCredential
+ apiContext(): APIContext
<<Class>>
PayPalOrderData
- price: double
- currency: String
- method: String
- intent: String
- description: String
+ getPrice(): double
+ getCurrency(): String
+ getMethod(): String
+ getIntent(): String
+ getDescription(): String
+ setPrice(price: double): void
+ setCurrency(currency: String): void
+ setMethod(method: String): void
+ setIntent(intent: String): void
+ setDescription(description: String): void
<<Class>>
PayPalController
- service: PayPalService
+ SUCCESS_URL: String
+ CANCEL_URL: String
+ payment(order: PayPalOrderData): String
+ cancelPay(): String
+ successPay(): String
<<Class>>
PayPalService
- apiContext: APIContext
+ createPayment(total: Double, currency: String, method: String, intent: String, description: String, cancelUrl: String, successUrl: String): Payment
+ executePayment(paymentId: String, payerId: String): Payment

Manage Itinerary



<<Class>>
BookTicketController
+ bookTicket(request: HttpServletRequest): ModelAndView
<<Display>>
PickClass.jsp
<<Display>>
Confirmation.jsp
<<Class>>
CartController
- addToCartService: AddToCartService
+ cartHandler(request: HttpServletRequest): ModelAndView
<<Class>>
AddToCartService
- ordersRepository: OrdersRepository
- userNameService: UserNameResolverService
+ retrieveOrderDetails(): String
+ addToCart(ticketDetails: String, classH: String): void
+ generateOrderId(): long
<<Interface>>
OrdersRepository
+ findByOrderId(order_id: long): List<Orders>

Quality Attributes

QA1: Modifiability

Scenario: The developer wishes to change the price of the tickets in the code at design time. The change is made without breaking other parts of the code quickly.

Source code location: Implementation is not yet created. This will most likely be in a file similar to Shopizer's that includes constants used.

QA2: Security

Scenario: The system allows access to the services and data for checkout when the user is logged in. The respective information is displayed only to users logged in.

Source code location: SecurityConfiguration.java

QA3: Interoperability

Scenario: The system connects to PayPal when the user is checking out. The page is redirected to a PayPal login corresponding to the payment account all the time if the checkout is successful.

Source code location: PayPalService.java and PayPalConfiguration.java