SEN 303 – SOFTWARE ENGINEERING Group ID:5 Ticket Automation System of Transport Companies

İÇİNDEKİLER

1.INTRODUCTION

1.1.1 Purpose and Scope

The aim of the project is to control the process of ticket buy/sell more quickly and effectively and controll the vehicles and routes.

1.1.2 Scope of The Project

This project is specially produced for each travel agency and is open to the use of manager or admin of the agency. It has been produced to inform the management unit about the seat, route situation. The project does not include very complex and complex structures.The system works on the side of customer and agency.

1.2 Goals and Success Criteria

Goals:

- Fast and simple program
- High availability
- Comply with travel agency conditions
- Obtaining an original and easy to use design
- Customer happiness
- Delivery as fast as possible

Success Criteria:

- Full run of the program
- Timely delivery
- Good planning
- Follow the project stages
- Coordinated work environment

1.3 Overview

1.3.1 System requirements:

- Must be a admin login
- Must be a user register and user login pages
- Vehicle info must be stored (brand, plate, maintanance date etc.)
- One seat can be sold by only one customer
- Firm must can edit the vehicles, add new vehicles or delete the existing one.
- Customer information must be stored (gender, age, seat no, phone number)

1.3.2 Member Requirements:

- Every member must have a username and password
- Making the authorization
- Members must can buy/cancel tickets
- Members must able to see their tickets' info

1.3.3 Test Items:

1.3.3.1 Features To Be Tested

- Login/logout
- Edit vehicles
- Interface
- Records
- Security
- Buying/Cancelling seats
- Error messages

1.3.3.2 Features Not To Be Tested

- Overloads
- Hardware errors

1.3.3.3 Item Pass/Fail Criteria

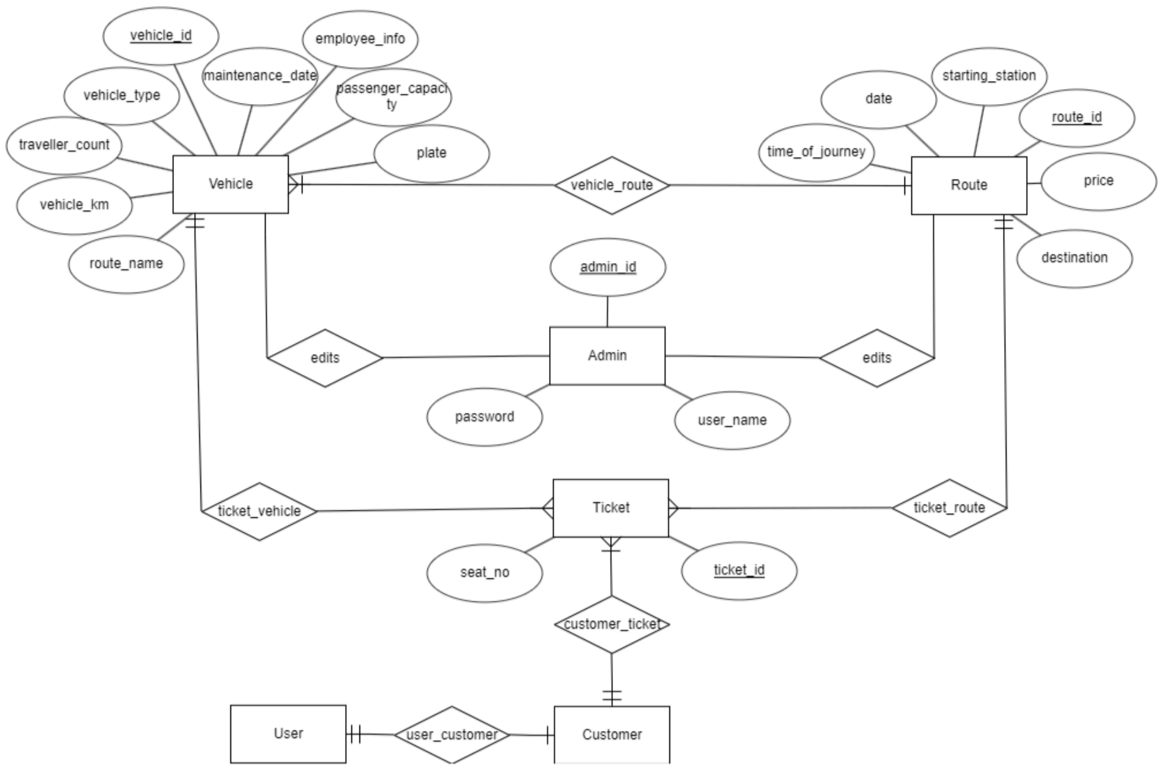
- Have security
- Correct functioning of functions
- Correct storage of data
- Being able to do the desired situations
- Successfully buying/cancelling tickets
- Successfully editing vehicles

1.3.4 Software Risk Issues:

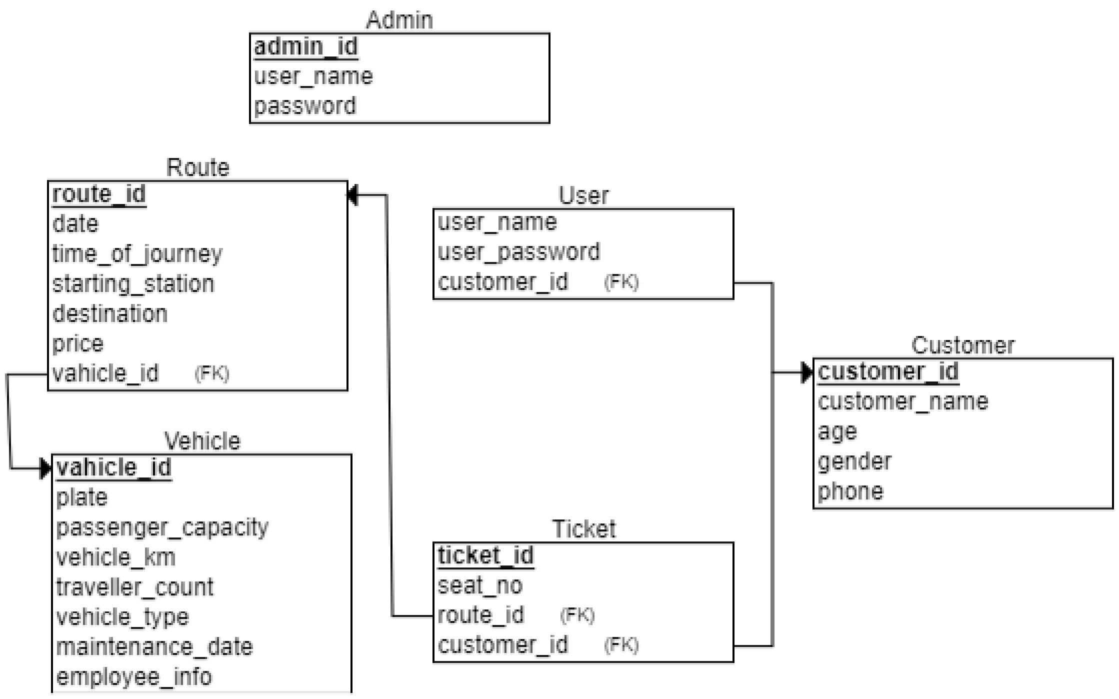
- Complex code
- Logic errors
- Bad database design
- Nonsecure code design

1.4 Data Models

1.4.1 E-R Diagram



1.4.2 Relational Tables



1.5 Software Processes and Process Models

1.5.1 Selected 5 Software Process Models and Reasons

- **Waterfall Model**: This model is suitable for simple and small projects that have clear and stable requirements. It follows a linear and sequential approach, where each phase depends on the deliverables of the previous one. It is easy to understand and manage, but it does not allow for much flexibility or feedback from the user.

- **V Model:** This model is an extension of the waterfall model, where each phase of development is associated with a corresponding phase of testing. It ensures high quality and verification of the system, but it also requires clear and fixed requirements. It is more efficient than the waterfall model, but it still does not support changes or user involvement.

- **Agile Model:** This model is based on iterative and incremental development, where the system is delivered in small and frequent releases. It emphasizes collaboration and communication between the developers and the users, and it welcomes changes and feedback. It is adaptive and responsive to the customer needs, but it also requires more discipline and commitment from the team.

- **Spiral Model:** This model combines the features of the waterfall and the iterative models, where each cycle consists of four phases: planning, risk analysis, engineering, and evaluation. It allows for early prototyping and testing, and it focuses on identifying and resolving the risks. It is suitable for complex and large projects, but it also requires more time and resources.

- **RAD Model:** This model stands for Rapid Application Development, and it aims to deliver the system in a short time frame. It uses a modular approach, where each module is developed and integrated separately. It involves the user throughout the development process, and it relies on tools and techniques to speed up the development. It is effective for fast and dynamic projects, but it also requires skilled and experienced developers.

1.5.2 Decided Software Process Model and Reason

RAD Model: This model allows us to deliver the system in a short time frame, which is suitable for our noncomplex Project and for a travel agency it is important to making updates and maintenance in a short time. It uses a modular approach, where each module is developed and integrated separately. It involves the user throughout the development process, and it relies on tools and techniques to speed up the development. It is effective for fast and dynamic projects. Compared to other models, the RAD model is more adaptive and responsive to the customer needs, but it also requires more discipline and commitment from the team. This is a disadvantage but a corparate company must have this property.

2. Requirements

This section defines the functional requirements of the project, which describe system features that will meet user expectations. Fundamental functions such as user operations, ticket management, route, and vehicle management are among these requirements. Additionally, non-functional requirements are specified, including areas such as security, usability, performance, supportability, and system constraints. Security measures involve protecting user information; usability focuses on an intuitive interface; performance ensures swift and efficient operations; supportability includes effective maintenance and updates; system constraints cover specific limitations and capacity features.

2.1 Functional Requirements

This section contains the core functional requirements of the project, expressing the functionality that will meet user expectations.

User Operations:

- Users must be able to log in.
- Users should be able to create accounts.
- Ticket search operations should be available.
- Ticket selection functionality should exist.

- Ticket purchasing operations should be available.
- Purchased tickets should be viewable.
- Users should be able to modify Personal Information.

Ticket Management and Operations:
- Tickets must be purchasable.
- Tickets must be cancellable.
- Tickets should be searchable based on route, vehicle, and date.

Route and Vehicle Management:
- Route information (departure points, arrival points, passenger count, travel duration, fare, travel date) must be editable (editing, adding, deleting).
- Scheduling, viewing, and tracking of trips should be possible.
- Vehicle information (capacity, type, plate, maintenance mileage, employee information) must be manageable (editing, adding, deleting).

Administrator Operations:
- Administrators should be able to manage user accounts (editing, adding, deleting).
- Administrators should have the authority to view and modify Ticket, User, Route, Vehicle, and trip information.
- Updating employee information.
- Ability to manage database tables (adding, editing, deleting).

2.2 Non-Functional Requirements

This section expresses the non-functional requirements of the project, covering aspects like security, usability, performance, etc.

2.2.1 In Terms of Security:

- Contains security-related requirements such as safeguarding user data, authorization, and authentication.
  - Encryption of user information.
  - Restricting table accessibility via encapsulations.
  - Open to admin access.
  - Users will have a password and username.

2.2.2 In Terms of Usability:

- Specifies requirements related to usability, including a user-friendly interface and easy navigation.
- Must have a user-friendly and easily usable interface design.
- Compatibility across different devices and browsers.
- Will have a login screen.
- The system is available 24/7.
- Session management through cookies.

2.2.3 In Terms of Performance:

- Expresses requirements related to performance, such as system speed, response time, etc.
- Speed and understandable screen interface.
- Ticket searching screen load time should not exceed two seconds for users.
- Server response should be swift.
- Database operations should be swift and efficient.
- The website should have fast loading and response times.

2.2.4 In Terms of Supportability (Maintenance):

- Includes supportability-related requirements like software maintenance and issue resolution.
- Planning should be done.
- Should be open to updates.
- Adequate infrastructure should be provided for regular software maintenance and updates.

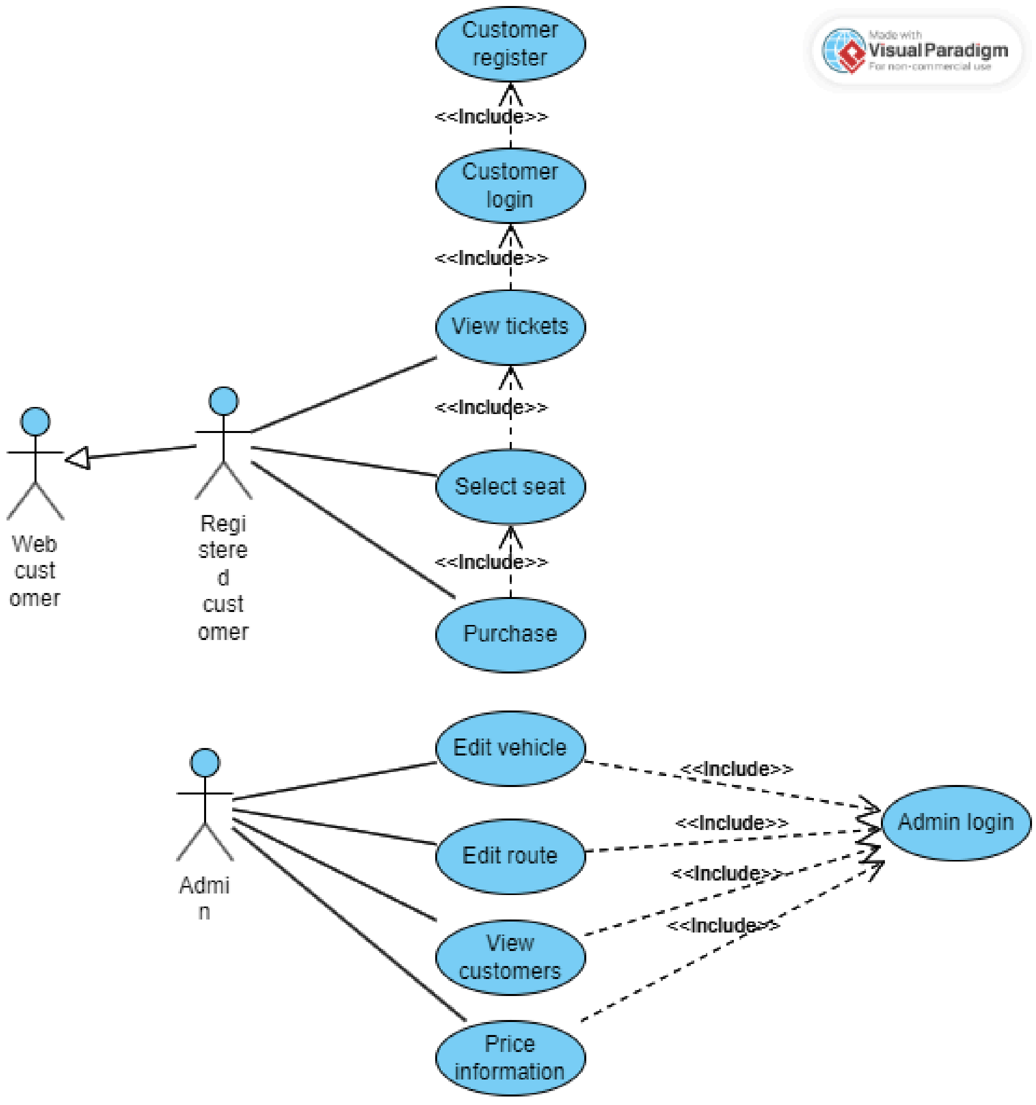
2.2.5 In Terms of Constraints (System Constraints):

- Indicates requirements related to system limitations, such as access restrictions for specific user roles.
- Access restrictions should be defined for specific user roles.

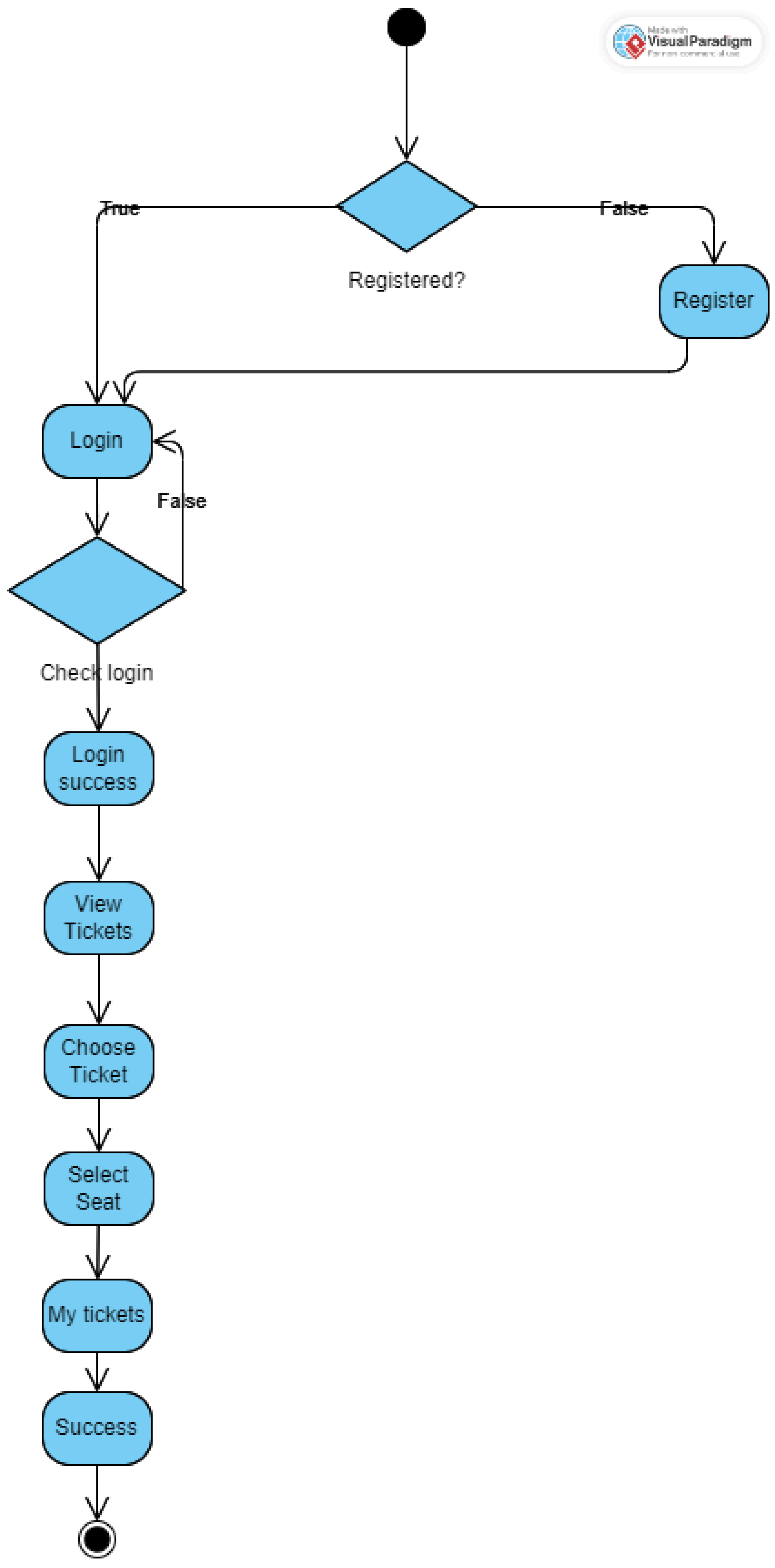
3. System Modelling with UML

3.1 Behavioral Diagrams

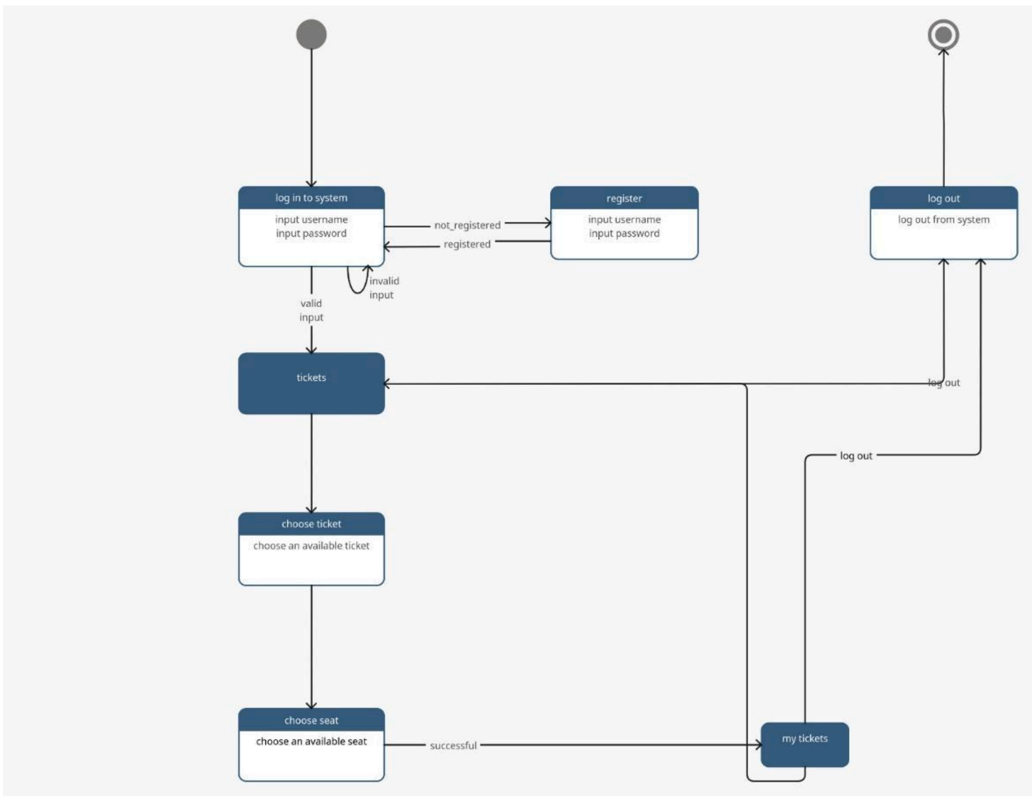
3.1.1 Use-Case Diagram



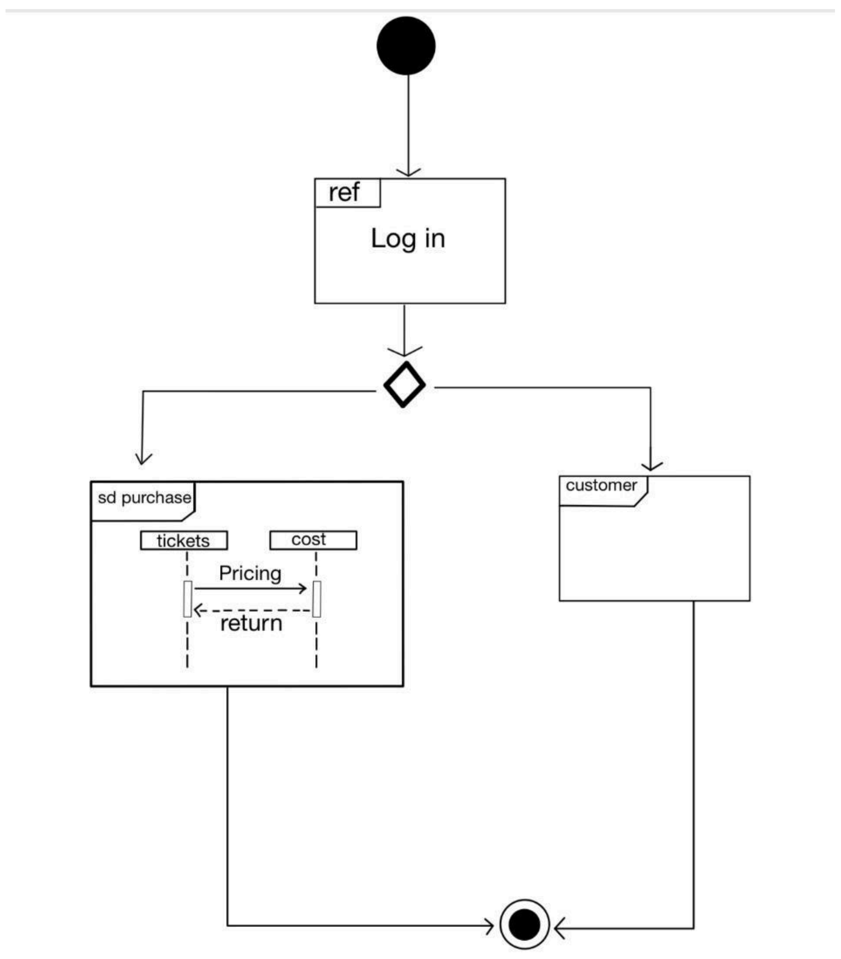
3.1.2 Activity Diagram



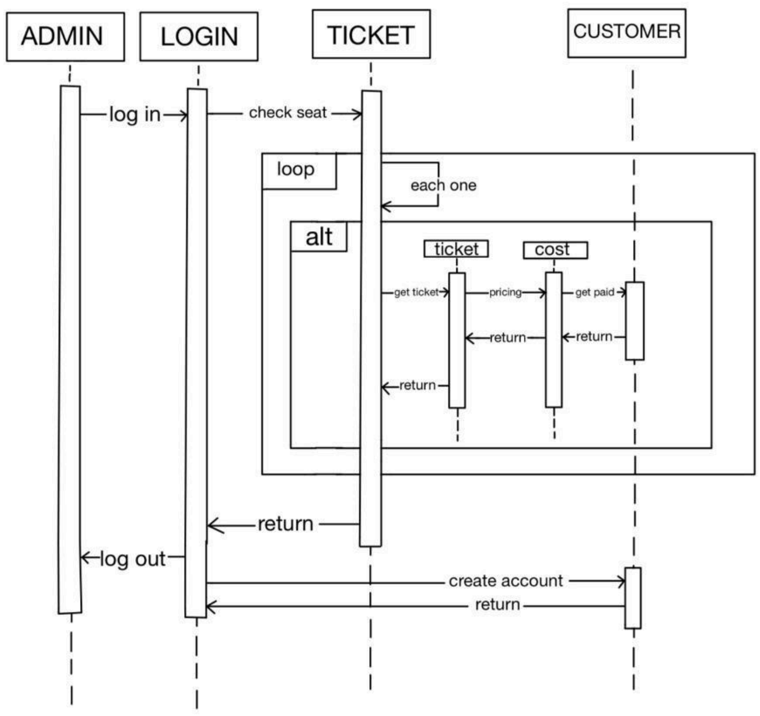
3.1.3 State (State Machine) Diagram



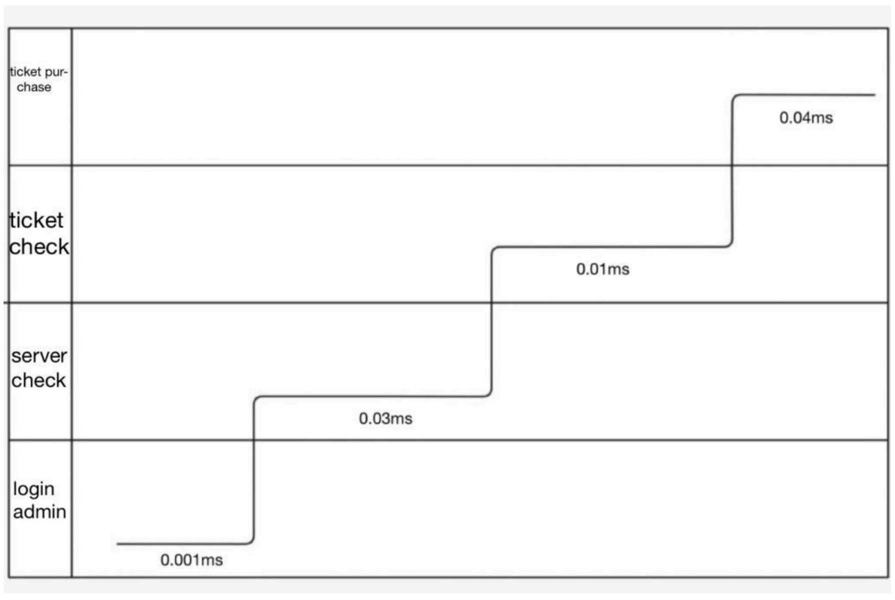
3.1.4 Interaction Diagram



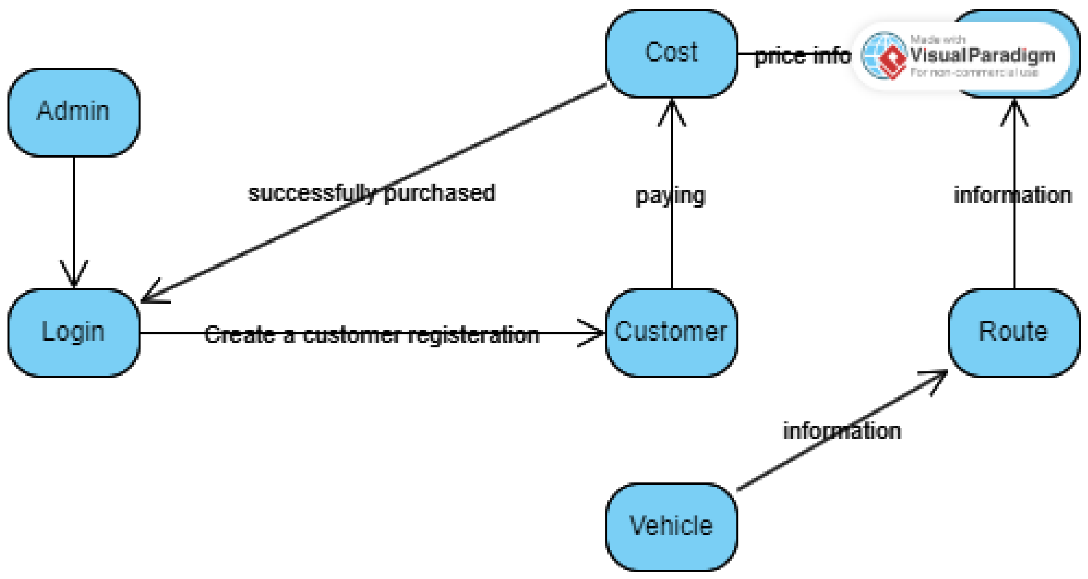
3.1.5 Sequence Diagram



3.1.6 Timing Diagram

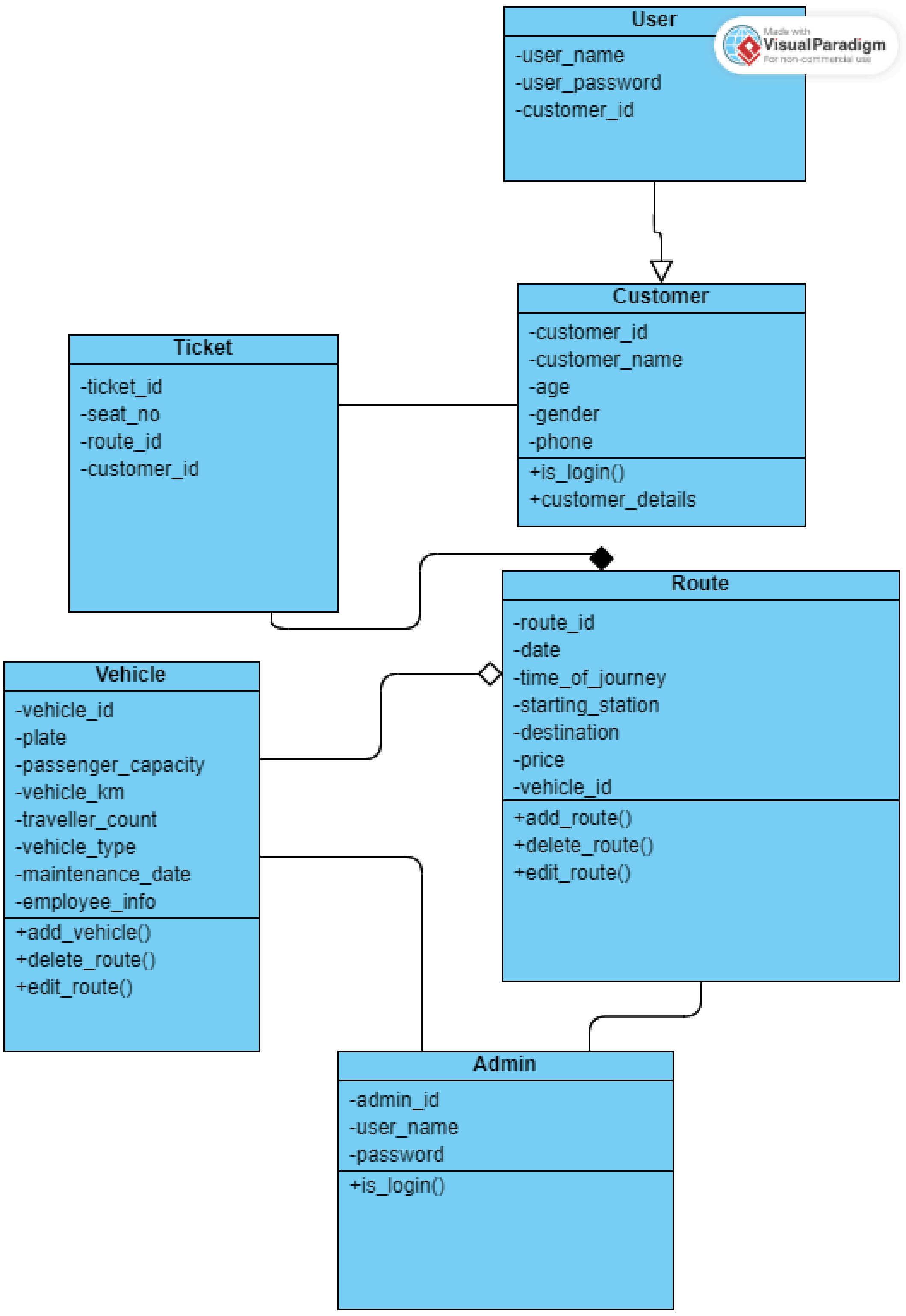


3.1.7 Collaboration Diagram (Communication Diagram)

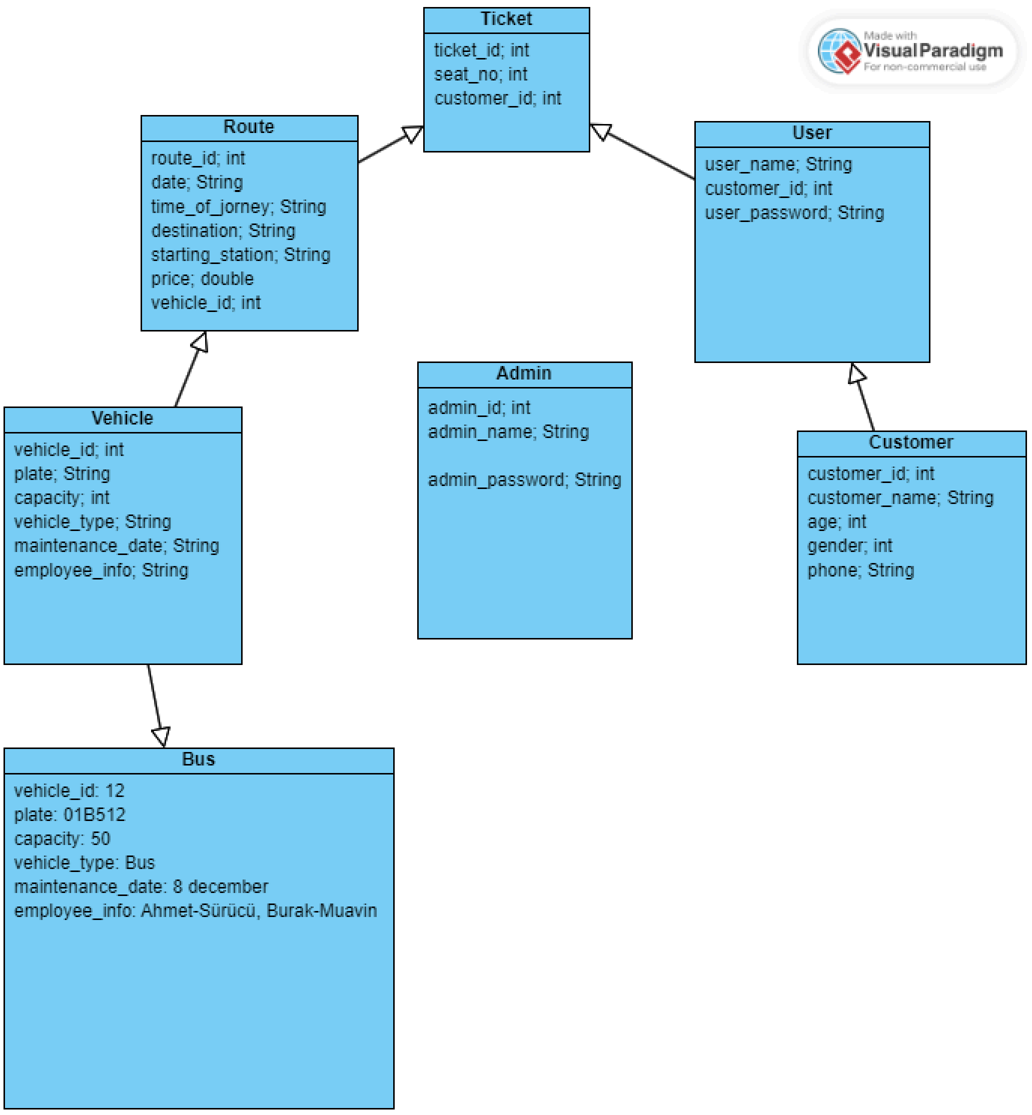


3.2 Structural Diagrams

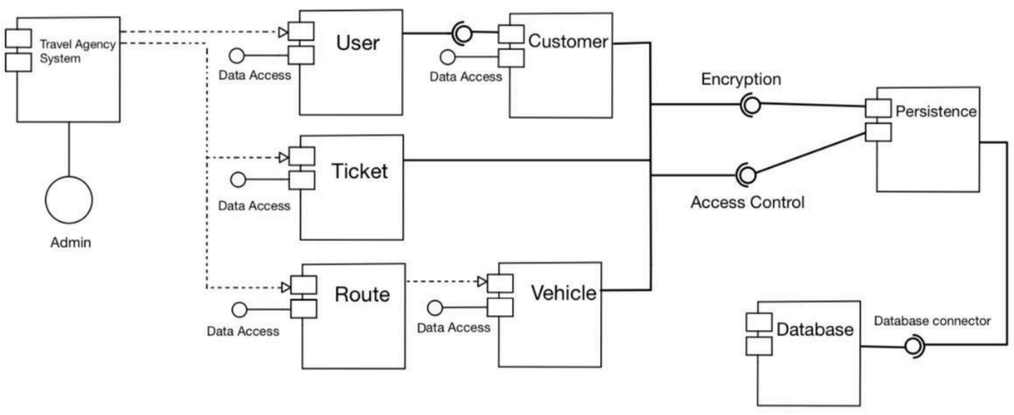
3.2.1 Class Diagram



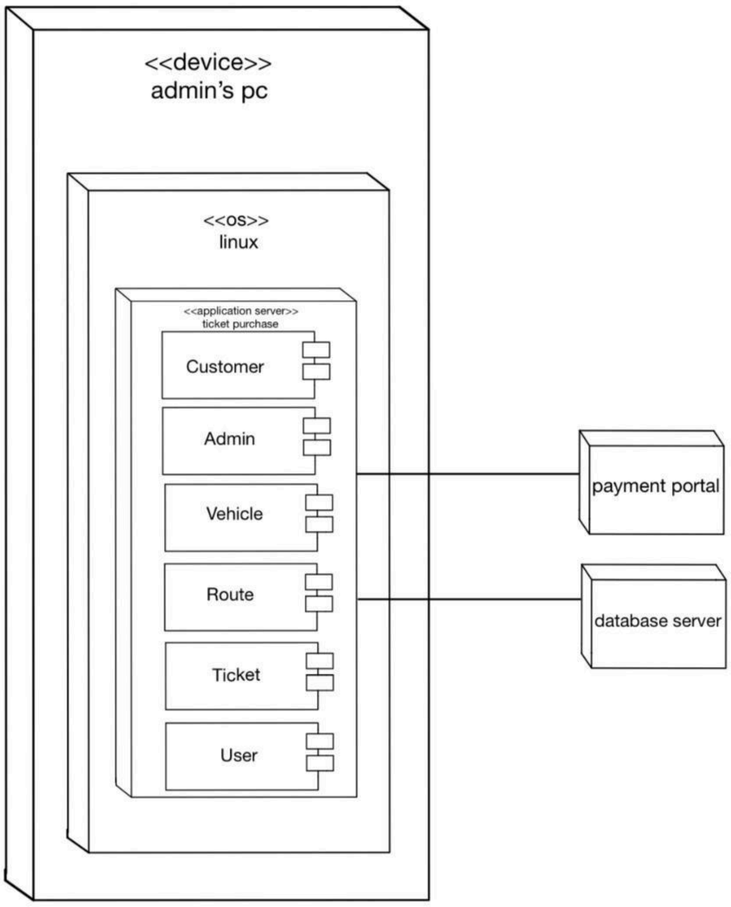
3.2.2 Object Diagram



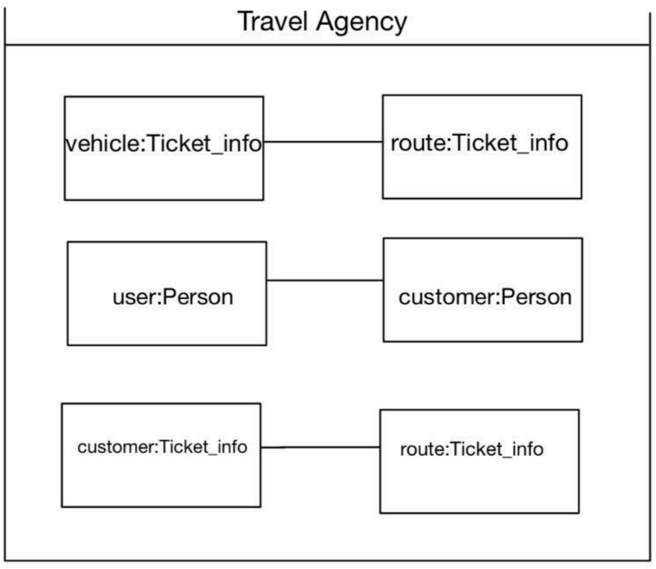
3.2.3 Component Diagram



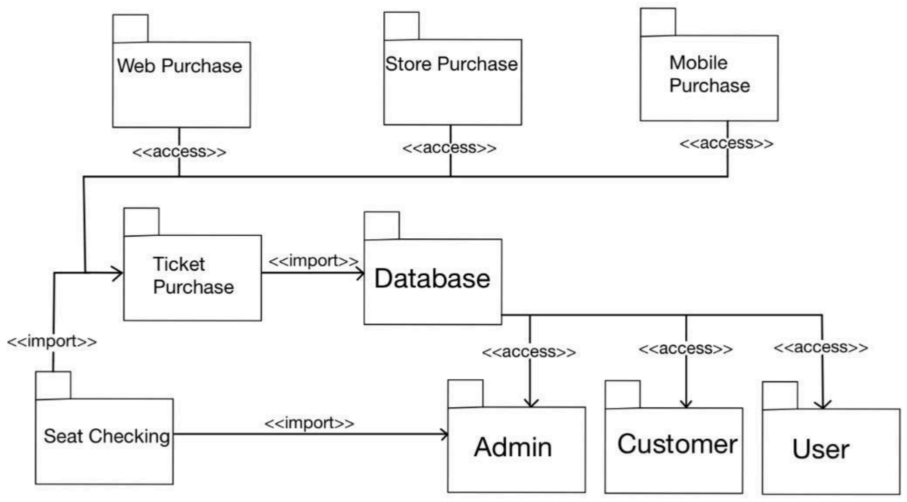
3.2.4 Deployment Diagram



3.2.5 Composite Structure Diagram



3.2.6 Package Diagram



4. Software Testing

4.1 Development Testing

Positive Test Cases:
1. Register a user.

2. Check if registration constraints such as validity of the username and the password are checked.

3. Login as a user.

4. Check if login screen is working properly.

5. Check if the user login is successful.

6. Check if index page is working properly.

   6.1 Check if source_city and destination_city sections take input correctly.

   6.2 Check if the given input is submitted successfully.

7. User fills the form and views available tickets.

8. Can the user view the all available tickets from source_city to destination_city?

9. User chooses a ticket.

10. Check only the available are shown to the user.

11. User selects an available seat.

  11.1 Check if the given seat is saved for later usage correctly.

12. User makes the purchase.

  12.1 Check if the purchase is successful.

  12.2 Check if the purchased ticket is recorded succesfully.

13. User redirected to "My Tickets" page.

  13.1 Check the page is only showing the bought tickets with correct information to the user properly.

14. User log outs.

  14.1 Check the user log outs succesfully.

  14.2 Check if the session is terminated succesfully.

15. Login back and check if there is a problem with the user data.

Negative Test Cases:

These are the tests that developers/testers can apply to the system and check the result of the action.

→ Input invalid username and password in the "Login" screen.

→ Input wrong username and password in the "Login" screen.

→ Input invalid source_city and destination_city in the "Index" screen.

→ Try to select the same seat twice.

Result:

The main goal of putting the system to the test is to see the outcome in terms of accuracy, robustness, and, performance. These routines must be done by the developers/testers during the development. Any problem should not occur after the deployment of the project. And the end user should be using the service without having an issue.

4.2 Unit Testing

- The user tried to claim a taken username and the system did not check for this condition. A conflict occurred in the “User” table.
    - Result: A condition branch written in registration block. Checks if the username is taken or not.
- The user picks an invalid password. (i.e. a password has a length of less than eight characters) and the data is written into the database. The system should be secured by encapsulating the data of the user.
    - Result: A condition branch written in registration block and the passwords will be stored as hashed values in the database.
- The customer selected a taken seat and the purchase is successful. The purchase system should not allow the user to select a taken seat and is not working properly.
    - Result: Purchase page will only show the empty seats. The customer is not be able to see the taken seats.

4.2.1 White Box Testing

Items	Testing Method	Reasons
Enter both valid username and password	White Box Testing	Registration successful
Enter both correct username and password	White Box Testing	Log in successful
Enter both incorrect username and password	White Box Testing	Log in unsuccesful and the account is temporarily suspended
First enter correct credentials then incorrect ones	White Box Testing	Log in succesful at first, unsuccesfull at second
Enter incorrect credentials five times in a row	White Box Testing	Log in unsuccesful at any attempt, account is temporarily suspended
Enter valid credentials for registration and then log in	White Box Testing	Registration and Log in successful → Saved

Name:	**Position:**	**Version:**	**Date:**
Ahmet Burak Biçer	Tester	1.0.0	25.11.2023
Mert Metin Erdemli	Software Engineer	1.0.1	10.12.2023

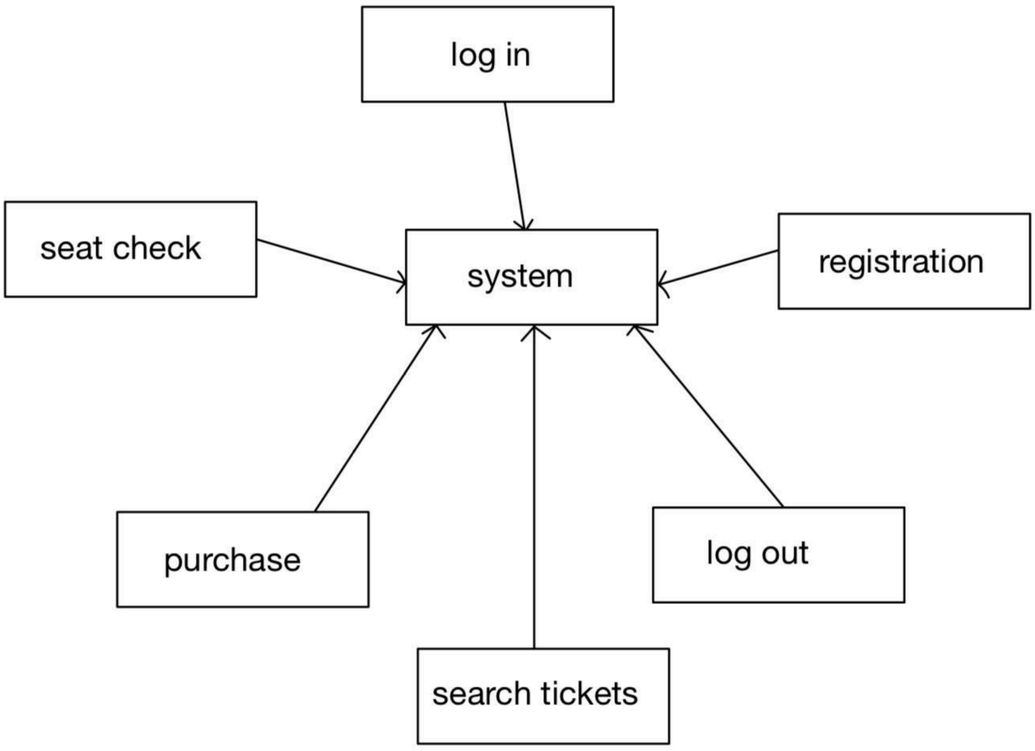
4.3 Integration Testing

The whole system is divided into modules. This provides mobility for testing the whole system. Each module has been tested and passed the cases individually. However, the system may work improperly or cause errors without having a test as a whole. Thus, the integration test aims for the system to work correctly and properly as a whole by testing the modules as combined.

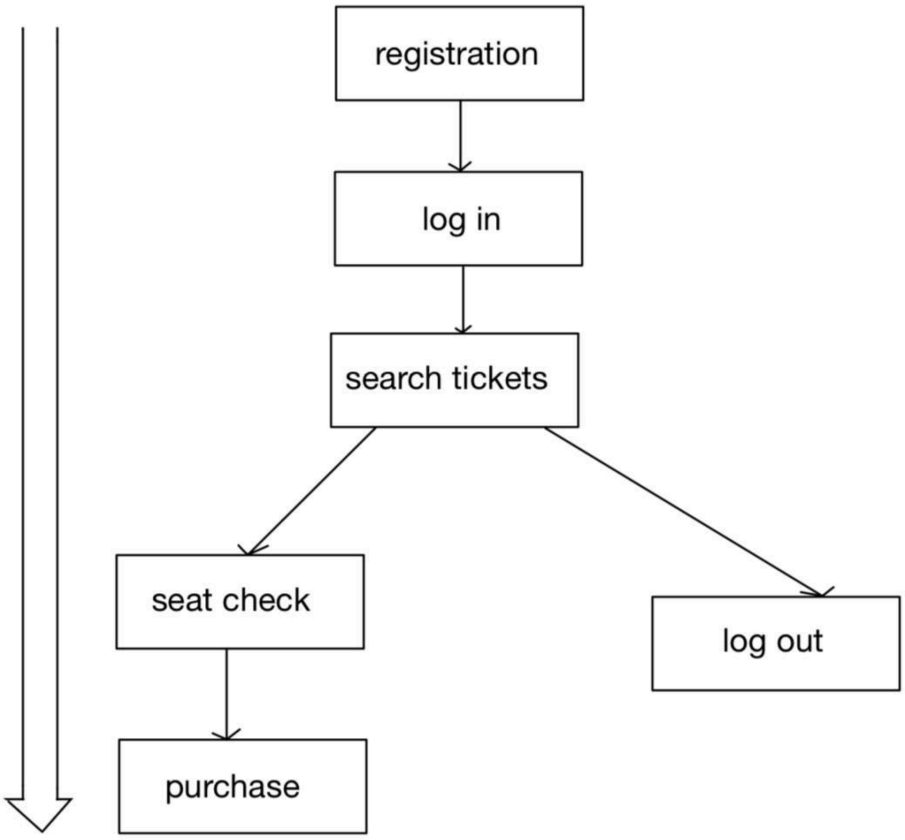
Modules are as follows:

- Login (as Customer/Admin)
- Registration
- Search tickets
- Seat availbility check
- Purchase
- Log out

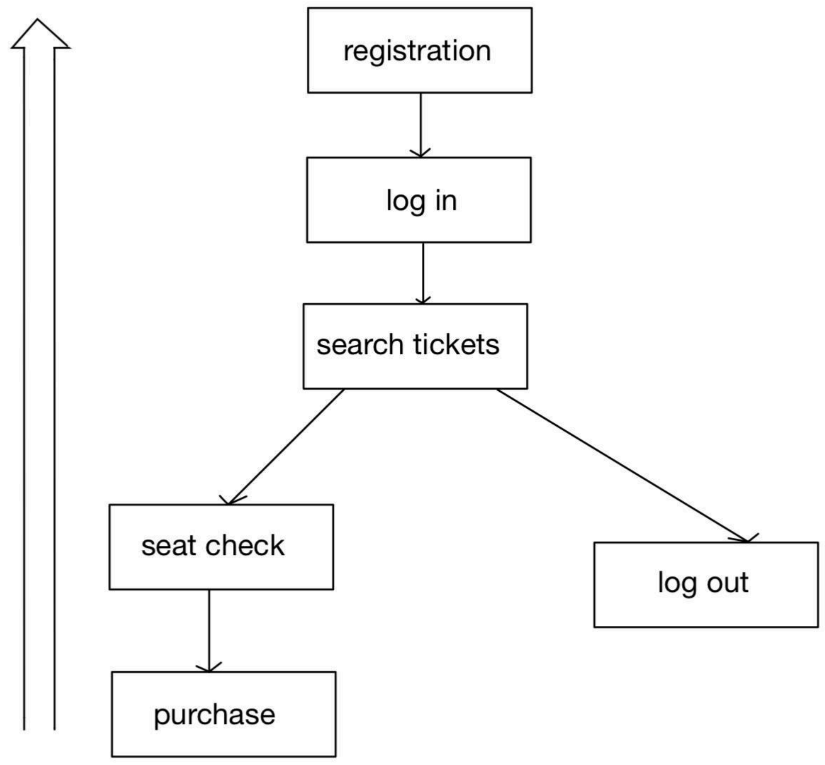
4.3.1 Big-bang



4.3.2 Top-down


registration
log in
search tickets
seat check
log out
purchase

4.3.3 Bottom-up



4.3.4 Hybrid

Hybrid approach is the combination of top-down and bottom-up approaches. These approaches are tested and reported in parallel. This results in a hybrid testing at the same time.

4.4 System Testing

4.4.1 Black Box Testing

Test case:	**Procedure to test:**	**Expected outcome:**
Login	Input wrong username/password	System should return an error.
Registration	Input invalid username/password	System should warn the user.
Add route	Add route from a city from itself	System should check the source and destination points.
Price interval	Set a negative price	System shuold rise an error.
Travel time	Set a negative duration time	System should rise an error.

Name:	**Position:**	**Version:**	**Date:**
Emirhan Balcı	Software Engineer	1.0.3	19.12.2023

4.5 User Testing/Acceptance Testing

User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.

4.5.1 Alpha Testing

In this process of testing, we tested the whole system with testers in terms of robustness, reliability, and performance. Main components taken into account with different world scenerios. The issues are addressed for the engineer team to fix and re-test the issues.

4.5.2 Beta Testing

For beta testing, we included the end-users to see the dynamic behaviour of our system. Since the system provides a travel agency service to the end-users, beta testing process is the backbone of the testing procedure. We tested the system's behaviour under high pressure and necessary precautions are addressed.

4.5.3 User Acceptance Testing

For user acceptance tesing, testers executed real world scenarios, documented issues, and collaborated with the development team for issue resolution. After retesting and confirmation, user approval is obtained, and the documentation is written. With stakeholder sign-off, the travel agency system transitions to production, ensuring it aligns with business requirements and meets user expectations for functionality and usability.

4.5.4 Operational Acceptance Testing

The last step of testing involved validating the system's operational features, including aspects such as performance, reliability, and support capabilities. This phase ensures that the software can seamlessly integrate into the existing operational environment, meet service level requirements, and, guaranteeing a smooth and effective deployment into the production environment.

4.6 Debugging

4.6.1 Debugging by Brute Force

In this phase of debugging process, we're targeting the weak spots of the system. First, our team decided that the ticket purchase system must be the first case.

Case: Ticket Purchase

Issue: Users report an intermittent issue where the ticket purchases sometimes fail without error messages.

Solution:

→ Step 1: Attempted a ticket purchase using different browsers to determine if the issue is browser-specific.

→ Step 2: Attempted a purchase with different payment methods to check if the problem is related to payment processing.

→ Step 3: Analyzed server and database logs for any recurring error messages or anomalies during failed purchase attempts.

→ Step 4: Temporarily disabled specific features (e.g., seat selection) to see if the issue persists, indicating a specific feature causing the problem.

→ Step 5: After making changes to the payment processing logic, continued to perform multiple ticket purchases to confirm that the error no longer occurs.

→ Step 6*:* We detected that the anomalies are somehow connected with seat selection feature. The issue is addressed, fixed and re-tested.

4.6.2 Debugging by Induction

The users reported occasional inconsistencies in displayed routes. A simple backtracking revealed a pattern related to recent changes in the code, particularly an update aimed at . Through code inspection, adjustments were made to the relevant database queries. Tests and controlled environment confirmed that the modified code successfully resolved the issue. The documentation of findings, changes, and the resolution process was updated, ensuring clarity for future developers and enhancing the overall reliability of the travel agency system.

4.6.3 Debugging by Deduction

After detecting an issue, a list of potential causes are systematically addressed. This list identifies patterns and commonalities, review recent code changes, inspect algorithms and data structures, and compare historical data to pinpoint when the problem started.

4.6.4 Debugging by Backtracking

This process of debugging is used in almost every scenario. If users/testers/developers find any issue at any point, we backtrack through the process in order to address the source of the issue. This method reduces the time of the debugging process.

5. Identification of Project Risks and Project Risk List

5.1 Identification of Project Risks

- **Data Integration:** Harmonizing information from different data sources.

- **Security Vulnerabilities:** Lack of security measures to protect user information.

- **Performance Issues:** Decrease in web application performance during high usage.

- **Database Relationships:** Lack of relationships between tables or suboptimal database structures.

- **User Experience:** Shortcomings in interface and functionality.

5.2 Project Risk List

Risk ID	**Description**	**Probability**	**Impact**	**Risk Mitigation Strategy**	**Contingency Plan**
1	Data Integration Challenges	Moderate	High	Adoption of modular integration methods	Use of backup data sources
2	Security Vulnerabilities	High	Moderate	Use of robust encryption and security algorithms	Emergency update and user notification
3	Performance Issues	Moderate	High	Implementation of load balancing and caching strategies	Performance monitoring and rapid response plan
4	Database Relationships	Low	High	Optimization of database structures	Data integrity preservation plan
5	User Experience Shortcomings	High	Moderate	Creation of user tests and feedback loops	Rapid interface improvement process

5.3 Planning Risks and Contingencies

- Data integration delays Pre-planned alternative integration methods.

- **Security breach:** Emergency update and user notification strategy.

- **Performance issues:** Rapid scalability and performance monitoring protocols.

6. Responsibilities of Members

6.1 Position Names of Project Members

- **Project Manager:** Project planning, scheduling, and resource allocation. (Mert Metin Erdemli)

- **System Architecture Designer:** System architecture design and infrastructure setup. (Ahmet Burak Biçer)

- **Database Specialist:** Database management, data integration, and performance optimization. (Mert Metin Erdemli)

- **Backend Developers:** Ensuring database integration and creating APIs. (Emirhan Balcı)

- **Frontend Developers:** Designing and developing the user interface. (Musa Alagöz)

- **Security Expert:** Security testing and vulnerability resolution. (Emirhan Balcı)

- **Testing Specialist:** Creating user test scenarios and analyzing feedback. (Musa Alagöz)

6.2 Task Sections Completed by Project Members

Project Planning and Research	: Ahmet Burak Biçer
Requirement Analysis	: Mert Metin Erdemli
System Modeling with UML	: Emirhan Balcı / Ahmet Burak Biçer
Software Testing Section	: Emirhan Balcı
Project Risk Analysis	: Musa Alagöz / Mert Metin Erdemli

6.3 Schedule of the Project

Tasks	Start Date	End Date	Responsible Member
Database Design	15/11/2023	10/11/2023	Mert Metin Erdemli
Infrastructure Setup	11/11/2023	13/11/2023	Ahmet Burak Biçer
UI Development	14/12/2023	16/12/2023	Musa Alagöz
Security Testing	17/12/2023	18/12/2023	Emirhan Balcı
User Feedback Analysis	19/12/2023	20/12/2023	Musa Alagöz
Database Integration	21/12/2023	22/12/2023	Mert Metin Erdemli
API Development	22/12/2023	23/12/2023	Emirhan Balcı
Performance Optimization	24/12/2023	24/12/2023	Ahmet Burak Biçer
Interface Refinement	25/12/2023	25/12/2023	Musa Alagöz
Security Enhancements	26/12/2023	27/12/2023	Ahmet Burak Biçer