VIETNAMESE – GERMAN UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER SCIENCE DEPARTMENT

JAVA PROJECT REPORT

<BUS TICKET SYSTEM>

Module 61CSE215: Object Oriented Programming in JAVA

1. <Tran Gia Huan – 10423043>
2. <Cao Dang Tri – 10423112>
3. <Ngo Trung Kien - 10423065>

Lecturer: Dr. Tran Hong Ngoc

VGU, WS2025

This page intentionally left blank.

Table of Contents

1 INTRODUCTION

In this project, we evaluate the business requirements of a **Bus Ticket Management System** for our university transit system. The currently in operation system makes use of Google Forms to record tickets, a method that is prone to mistakes and naturally time-consuming. Therefore, this project aims to compliment the current system with a dedicated desktop application in hopes of providing a fast and stable system that manages tickets and trips for both students and staff.

The application in this project provides the users with the following functionalities:

- The user can register a new account and log into their account using a password.
- The user can purchase various types of tickets.
- The user can query and see all purchased tickets that are tied to their account.

There is naturally always room for more features and further development. That being said, we focus mainly on GUI design, Java classes, and database implementation for this project.

1.1 Stakeholder Roles

No.	Role	Description
1	Admin/Staff	Manages the bus system and is responsible for any changes in the application.
2	Student/Professor	Uses the system to create an account, create bookings and view ticket information.

2 CLASS ANALYSIS

2.1 Object Analysis

No	Object Name	States (Attributes)	Behaviours (Methods)
1	Student1	first_name = “Huan” last_name = “Tran” email = “abc@vgu.edu.vn” password = “***” id = “10423043”	getFirstName() getLastName() getEmailAddress() getPassword() getID()
2	LongTermTicket1	id = 10423043 start_date = 11-25-2025 location = “Turtle Lake” ticket_type = ONEWAY direction = FROM	getID() getStartDate() getLocation() getTicketType() getDirection()
3	DatabaseController1		close() insert(User u), insert(Ticket t) search_tickets(String id) search_user(String id)
4	OneWayDetails1	id = 10423043 start_date = 11-25-2025 location = “Turtle Lake” ticket_type = ONEWAY price = 150000 type = ONEWAY direction = FROM pickuptime = 16:30	returnTicketDetails()

2.2 Grouping and Analysis

- **Student1** is an instance of **Student**, a subclass of **User**.
- **LongTermTicket1** instances of **LongTermTicket**, both are subclasses of **Ticket**.
- **OneWayDetails1** is an instances of **OneWayDetails** – a subclass of **TicketDetails**.
- **DatabaseController1** is an instance of **DatabaseController**.

3 CLASS DESIGN

3.1 Class Diagrams

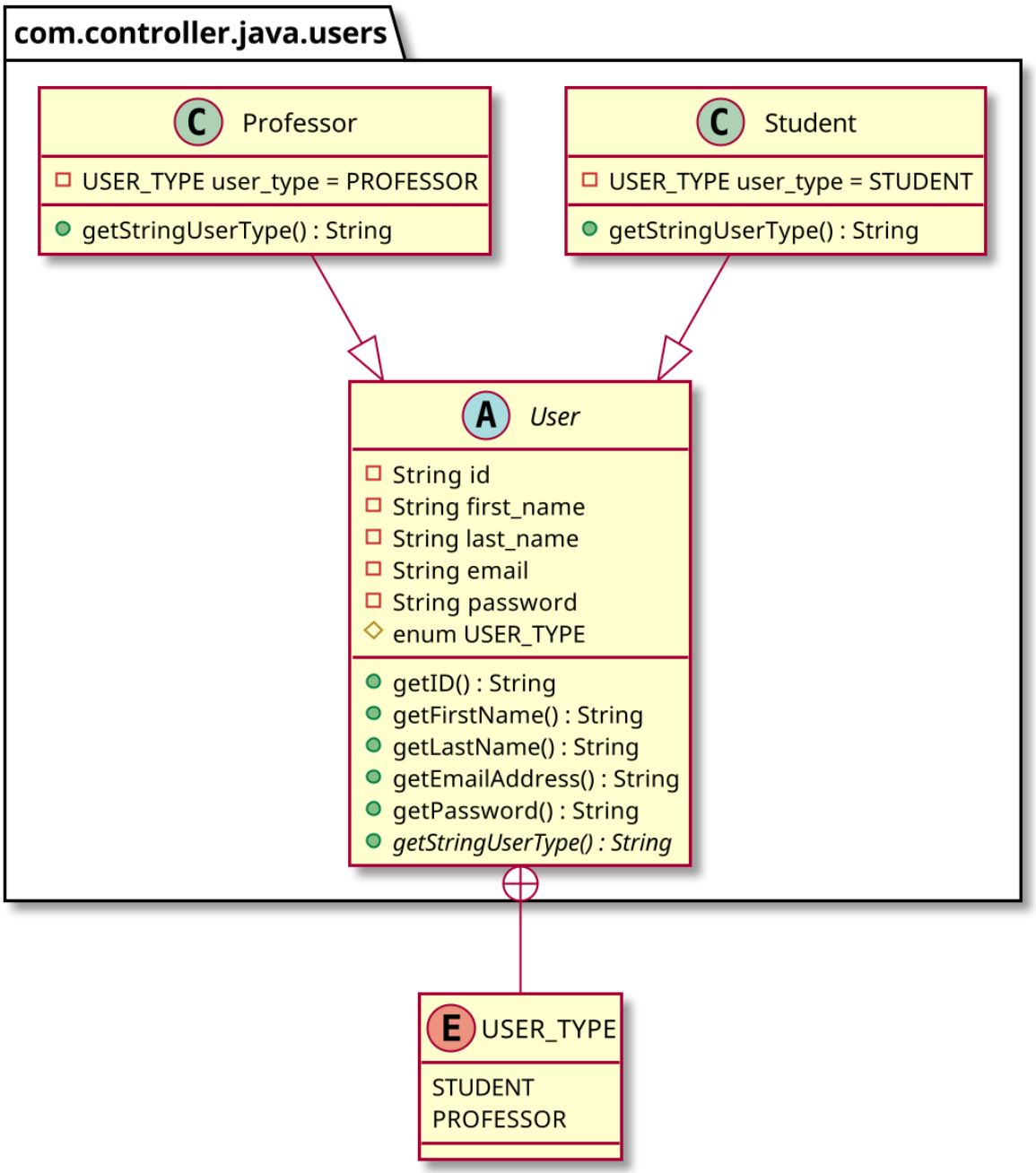

Figure 1: UML class diagram for package *com.controller.java.users*

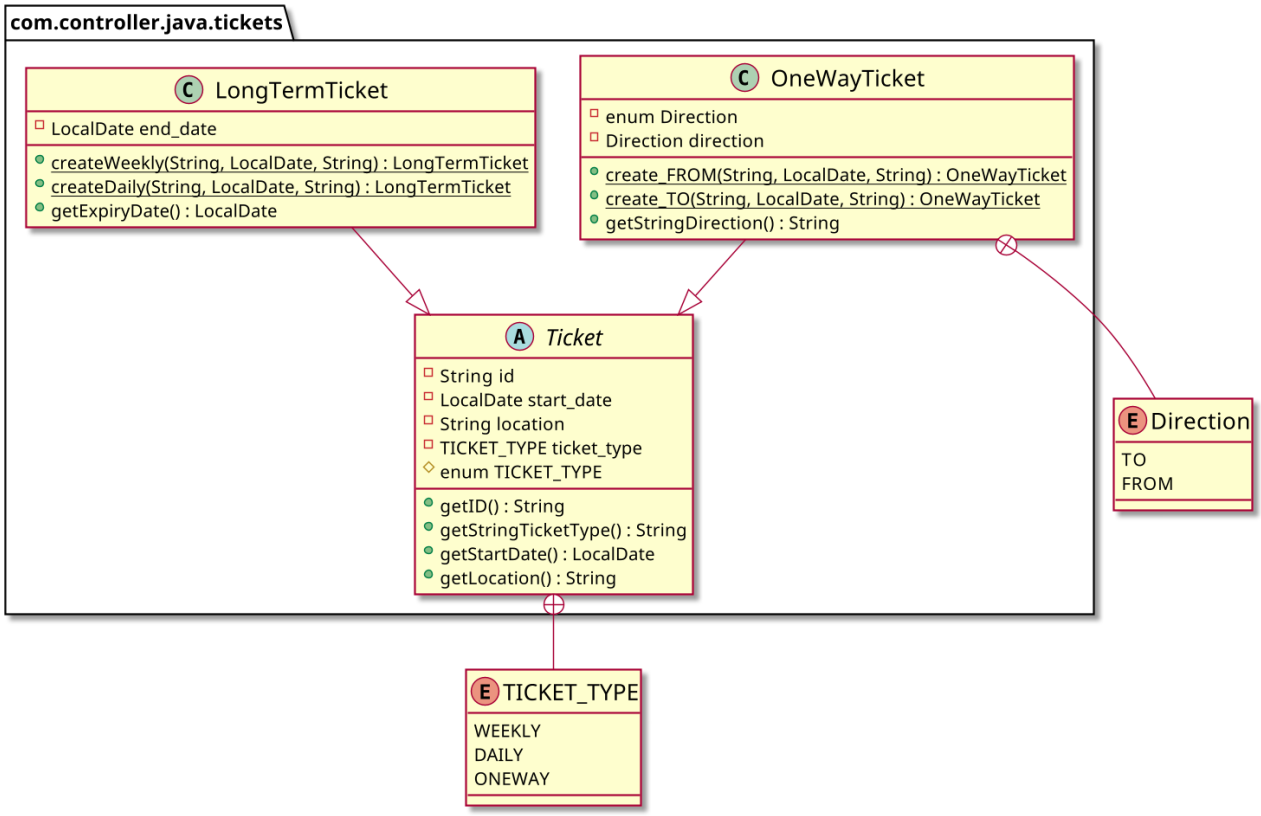

Figure 2: UML class diagram for *com.controller.java.ticket*

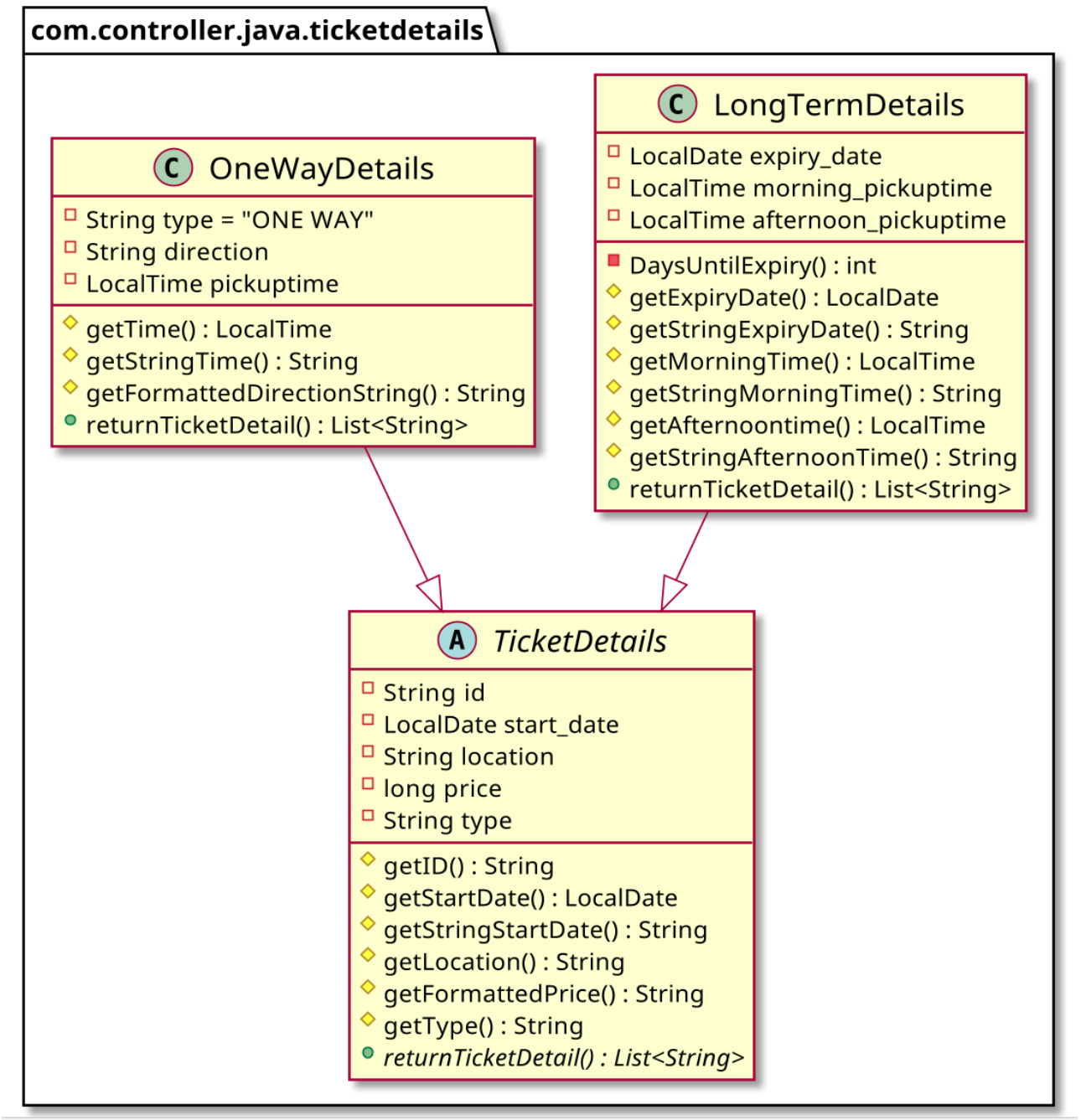

Figure 3: UML class diagram for *com.controller.java.ticketdetails*

3.2 Class Details

No	Class	Instance Variables	Methods	Description
1	User	**private final** String id **private final** USER_TYPE user_type **private final** String first_name **private final** String last_name **private final** String email; **private final** String password **private final** double user_price_multiplier **protected** enum USER_TYPE{STUDENT, PROFESSOR}	**private** User() **public** String getID() **public** String getUserType() **public** String getFirstName() **public** String getLastName() **public** String getEmailAddress() **public** String getPassword() **public** double getPriceMultiplier()	Base abstract class for all User objects.
2	Student		**public** Student()	Subclass of User.
3	Professor		**public** Professor()	Subclass of Professor
4	Ticket	**private final** String id **private final** LocalDate start_date **private final** String location **private final** TICKET_TYPE ticket_type **protected** enum TICKET_TYPE{WEEKLY, DAILY, ONEWAY}	**private** Ticket() **public** String getID() **public** String getTicketType() **public** LocalDate getStartDate() **public** String getLocation()	Base abstract class for all Ticket objects. Used for inserting new tickets into database.
5	LongTermTicket	**private final** LocalDate end_date	**private** LongTermTicket() **public** static LongTermTicket createWeekly() **public** static LongTermTicket createDaily() **public** LocalDate	Subclass of Ticket. This includes long-term weekly and daily tickets.

			getExpiryDate()	
7	OneWayTicket	**private** enum Direction{TO, FROM} **private final** Direction direction	OneWayTicket() create_FROM() create_TO() **public** String getDirection()	Subclass of Ticket. This is used for one-way trips.
8	TicketDetails	**private final** String id **private final** LocalDate start_date **private final** String location **private final** long price **private final** String type	**protected** final String getID() **protected** final LocalDate getStartDate() **protected** String getFormattedPrice() **protected** String getType() **public** abstract List<String> returnTicketDetail();	Base abstract class for all TicketDetails object. Used for querying ticket information retrieved from database.
9	LongTermDetails	**private final** LocalDate expiry_date **private final** LocalTime morning_pickuptime **private final** LocalTime afternoon_pickuptime	**public** LongTermDetails() **private** int DaysUntilExpiry() **protected** String getExpiryDate() **protected** String getMorningTime() **protected** String getAfternoonTime() **public** final List<String> returnTicketDetail()	Subclass of TicketDetails. Used for showing information of all weekly and daily tickets.
10	OneWayDetails	**private final** String direction **private final** LocalTime pickuptime	**protected** String getTime() **protected** String getDirection() **public** final List<String> returnTicketDetail()	Subclass of TicketDetails. Used for showing information of all one-way tickets.
11	DatabaseController		DatabaseController() **public** void close()	Database controller that

			public void insert(Ticket t) **public** void insert(User u) **public** Map<String, List> search_tickets(String id) **public** User search_user(String id)	interacts with GUI and underlying Database for CRUD operations.

3.3 Abstract Classes

No.	Abstract class	Abstract methods	Concrete methods	Description
1	**User**	None	**private** User() **public** String getID() **public** String getUserType() **public** String getFirstName() **public** String getLastName() **public** String getEmailAddress() **public** String getPassword() **public** double getPriceMultiplier()	Used by Student and Professor. This class cannot be directly instantiated and its variables are defined by its subclasses.
2	**Ticket**	None	**private** Ticket() **public** String getID() **public** String getTicketType() **public** LocalDate getStartDate() **public** String getLocation()	Used by **LongTermTicket** and **OneWayTicket**. This class cannot be directly instantiated and its variables are defined by its subclasses.
3	**TicketDetails**	**public** abstract List<String> returnTicketDetail()	**protected** final String getID() **protected** final LocalDate getStartDate() **protected** String getFormattedPrice() **protected** String getType()	Used by **LongTermDetails** and **OneWayDetails**. It has an abstract method that changes based on the created subclass.

3.4 Object-Orientated Programming Techniques

Inheritance

We use several inheritance-related techniques in this project:

- **User** extends **Student** and **Professor**.
- **Ticket** extends **LongTermTicket** and **OneWayTicket**.
- **TicketDetails** extends **LongTermDetails** and **OneWayDetails**.

These superclasses are declared as **abstract** such that they cannot be instantiated directly.

Overloading

com.database.CRUD.CRUD_Tickets.java

```
// Overloading Inserts:

// Long-term insert
public static void insert_ticket(Connection conn, LongTermTicket lt) throws TicketInsertionException{
    // Eligibility check
    if (!CRUD_TicketHelpers.eligible(conn, lt.getID())){
        throw new TicketInsertionException("User already has an active long-term ticket!");
    }

    String insertStmt = "INSERT INTO public.longterm(id, start_date, end_date, ticket_type, location_name) VALUES (?, ?, ?, ?, ?)";

    try (PreparedStatement stmt = conn.prepareStatement(insertStmt)) {
        stmt.setString(1, lt.getID());
        stmt.setDate(2, Date.valueOf(lt.getStartDate()));
        stmt.setDate(3, Date.valueOf(lt.getExpiryDate()));
        stmt.setString(4, lt.getStringTicketType());
        stmt.setString(5, lt.getLocation());

        stmt.executeUpdate();
    }
    catch (SQLException sqle) {
        System.out.println(sqle.getMessage());
        throw new TicketInsertionException();
    }
}
```

Figure 4: Overloaded insert_ticket(LongTermTicket)

```
// One-way insert
public static void insert_ticket(Connection conn, OneWayTicket owt) throws TicketInsertionException{
    String insertStmt = "INSERT INTO public.oneway(id, departure_date, location_name, ticket_type, direction) "
            + "VALUES (?, ?, ?, ?, ?::direction_type)";

    try (PreparedStatement stmt = conn.prepareStatement(insertStmt)){
        stmt.setString(1, owt.getID());
        stmt.setDate(2, Date.valueOf(owt.getStartDate()));
        stmt.setString(3, owt.getLocation());
        stmt.setString(4, owt.getStringTicketType());
        stmt.setString(5, owt.getStringDirection());

        stmt.executeUpdate();
    }
    catch(SQLException sqle){
        System.out.println(sqle.getMessage());
        throw new TicketInsertionException("Only support 2 oneways per day!");
        }
}
```

Figure 5: Overloaded insert_ticket(OneWayTicket)

Overriding

```java
public abstract class TicketDetails {
    // Print ticket
    public abstract List<String> returnTicketDetail();
}
```

Figure 6: Abstract returnTicketDetail()

```java
public class LongTermDetails extends TicketDetails{
    @Override
    public final List<String> returnTicketDetail(){
        List<String> ticket_detail = new ArrayList<>();
        ticket_detail.add(getID());
        ticket_detail.add(getType());
        ticket_detail.add(getFormattedPrice());
        ticket_detail.add(getLocation());

        ticket_detail.add(getStringMorningTime());
        ticket_detail.add(getStringAfternoonTime());

        ticket_detail.add(getStringStartDate());
        ticket_detail.add(getStringExpiryDate());
        String expiry_status = (DaysUntilExpiry() < 0) ? "EXPIRED" : "ACTIVE";
        ticket_detail.add(expiry_status);

        return ticket_detail;
    }
}
```

Figure 7: Overriding in LongTermDetails.java

```java
public class OneWayDetails extends TicketDetails{
    @Override
    public final List<String> returnTicketDetail(){
        List<String> ticket_detail = new ArrayList<>();
        ticket_detail.add(getID());
        ticket_detail.add(getType());
        ticket_detail.add(getFormattedPrice());

        ticket_detail.add(getFormattedDirectionString());
        ticket_detail.add(getStringStartDate());
        ticket_detail.add(getStringTime());


        return ticket_detail;
    }
}
```

Figure 8: Overriding in OneWayDetails.java

Three-Tier Model

Conceptually, the system follows a three-tier architecture to separate concerns:

- **Presentation Layer**
  - **GUI** interacts with the user and calls methods from database controller and ticket-related classes.
- **Controller Layer**
  - **Domain classes i**mplement business rules such as ticket validity, journey direction, and printing ticket details.
  - **Database controller** acts as a bridge between GUI and Database by passing user data to underlying database system.
- **Database Layer**
  - **Database connector** establishes connection with database.
  - **CRUD operations** defines and enacts SQL functionalities (insert, select).

4 PACKAGE DESIGN

- ***gui***
  - Classes: **MainMenu**, **UserLogIn**, **NewUser**, **UserMenu**, **BusBooking**.
  - Provides a graphical interface for better user experience.
- ***controller.java.users***
  - Classes: **User**, **Student**, **Professor**.
  - Represents user entities and user-related logic.
- ***controller.java.tickets***
  - Classes: **Ticket**, **LongTermTicket**, **OneWayTicket**.
  - Represents ticket entities and ticket-specific logic.
- ***controller.java.ticketdetails***
  - Classes: **TicketDetails**, **LongTermDetails**, **OneWayDetails**.
  - Represents how ticket information is formatted, presented, and printed.
- ***controller.database***
  - Class: **DatabaseController**.
  - Manages communication with the database by delegating low-level operations to DatabaseConnection.
- ***database***
  - Class: **DatabaseConnection**.
  - Connects to database and calls CRUD operations.
- ***database.CRUD***
  - Classes: **CRUD_Tickets**, **CRUD_TicketHelper**, **CRUD_Users**
  - Defines and enacts CRUD operations.
- ***exceptions***
  - Classes: **DatabaseConnectionException, NewUserException, LogInException**, **TicketInsertionException**, **UserSelectionException**, **TicketSelectionException.**
  - Models different error situations in a clear and structured way.

5 INTERFACE DESIGN

We use interface to store database configurations (like database username, url and password). DatabaseConnection accesses this interface to retrieve credentials for establing a connection with the underlying PostgreSQL Database.

```
interface ConfigInterface{
    // Interface variables
    String url = "jdbc:postgresql://localhost:5432/BusTicketManagementSystem";
    String username = "postgres";
    String password = "huan";

    // Interface methods
    String getUrl();
    String getUsername();
    String getPassword();
}
```

Figure 9: ConfigInterface

6 ACCESS CONTROL

We analyse and discuss the access control applied to variables (data) and methods in the system. In general, sensitive or identity-related data is declared **private final** to enforceencapsulation and immutability, while controlled access is provided through **public** getters or well-defined service methods (e.g., database operations). Data types that are only relevant within their packages are declared as **protected**.

6.1 Data Access Control

No	Data	Class	Access Modifier	Description
1	id, user_type, first_name, last_name, email, password, user_price_modifier	**User**	**private final**	Sensitive data regarding user's account and identity. Cannot be modified and can only be accessed via getters.
2	enum USER_TYPE {STUDENT, PROFESSOR}	**User**	**protected**	Specify user type (Student or Professor). Only relevant within **User** package.
3	id, start_date, location, ticket_type.	**Ticket**	**private final**	Sensitive ticket data
4	enum TICKET_TYPE {WEEKLY, DAILY, ONEWAY}	**Ticket**	**protected**	Specify ticket type. Only relevant within **Ticket** package.
5	end_date	**LongTermTicket**	**private final**	Sensitive data of long-term tickets' expiry date
6	direction	**OneWayTicket**	**private final**	Sensitive data of one-way ticket's direction
7	enum Direction{TO, FROM}	**OneWayTicket**	**private**	Specify direction of one-way tickets. Only relevant within **OneWayTicket** class.
8	id, start_date, location, price, type	**TicketDetails**	**private final**	Sensitive queried ticket data
9	expiry_date, morning_pickuptime, afternoon_pickuptime	**LongTermDetails**	**private final**	Sensitive long-term ticket data
10	direction, pickuptime	**OneWayDetails**	**private final**	Sensitive one-way ticket data

6.2 Method access control

No.	Method	Class	Modifier	Description
1	getID(), getType(), getFirstName(), getLastName(), getEmailAddress(), getPassword(), getPriceMultiplier()	**User**	**public**	Getters to access aforementioned **private** variables in **User**.
2	User()	**User**	**protected**	**Protected** constructor since only constructors in **Student** and **Professor** are supposed to be called, which will call this constructor using **super()**
3	Student(), Professor()	**Student** **Professor**	**public**	These constructors will be called when a new user is created.
4	getID(), getTicketType(), getStartDate(), getLocation()	**Ticket**	**public**	Getters to access the aforementioned **private** variables in **Ticket**.
5	Ticket()	**Ticket**	**protected**	**Protected** constructor since only constructors in **LongTermTicket** and **OneWayTicket** are supposed to be called, which will call this constructor using **super()**
6	getExpiryDate()	**LongTermTicket**	**public**	Getter to access **private** variable **end_date** in **LongTermTicket.**
7	LongTermTicket()	**LongTermTicket**	**private**	This constructor is not meant to be called directly by other classes. It is only evoked via the two following class methods. Hence, it is only relevant inside the **LongTermTicket** class

				and given a **private** modifier.
8	createWeekly() createDaily()	**LongTermTicket**	**public**	Class methods that call the **private** constructor in **LongTermTicket** and defines ticket_type based on which one is called.
9	getDirection()	**OneWayTicket**	**public**	Getter to access **private** variable **direction** in **LongTermTicket.**
10	OneWayTicket()	**OneWayTicket**	**private**	This constructor is not meant to be called directly by other classes. It is only evoked via the two following class methods. Hence, it is only relevant inside the **OneWayTicket** class and given a **private** modifier.
11	create_TO() create_FROM()	**OneWayTicket**	**public**	Class methods that call the **private** constructor in **OneWayTicket** and defines direction based on which one is called
12	getID(), getPrice() getStartDate(), getLocation(), getTicketType()	**TicketDetails**	**public**	Getters to access the aforementioned **private** variables in **TicketDetails**.
13	returnTicketDetail()	**TicketDetails**	**public** abstract	Calling this method to make use of dynamic runtime polymorphism to flexibly print ticket details
14	returnTicketDetail()	**LongTermDetails** **OneWayDetails**	**public**	This method overrides the abstract method in **TicketDetails**

7 ENCAPSULATION, INHERITENCE AND POLYMORPHISM

7.1 Encapsulation

For encapsulation, we enclose data using **private final** fields, and provide **public** getters for variable access. For example, the **Ticket** class keeps most of its variables **private** and provides **public** getters. This ensures that once a ticket is created, its data can only be accessed through **public** getters and cannot be modified without permission.

Certain elements are set to **protected** as they are only meant to be used within their package. For example, the **Ticket** class has a **protected** enumeration for ticket types as this enumeration is only meant to be used for ticket creation and nowhere else. This enumeration is therefore visible to every file in its package to make use of it (**Ticket**, **LongTermTicket**, **OneWayTicket**) but not to external files.

7.2 Inheritance

Inheritance allow us to have multiple related classes containing the same variables and methods. For instance: **Student** and **Professor** are subclasses of **User**.

7.3 Polymorphism

Polymorphism is used to generalize different subclasses by assigning their references to a parent class object. For example, in ***com.gui.BusBooking***, a **Ticket** object is created and then passed references of its subclasses depending on the option picked.

```
// Create ticket obj
Ticket t;

String location = (String) DestinationComboBox.getSelectedItem();
LocalDate start_date = Calendar.getDate().toInstant().atZone(ZoneId.systemDefault()).toLocalDate();

if (DailyButton.isSelected()){        /*daily*/
    t = LongTermTicket.createDaily(id, start_date, location);

}
else if (WeeklyButton.isSelected()){    /*weekly*/
    t = LongTermTicket.createWeekly(id, start_date, location);
}
else{
    if (DirectionComboBox.getSelectedIndex() == 1){ /*one way*/
        t = OneWayTicket.create_FROM(id, start_date, location);
    }
    else{
        t = OneWayTicket.create_TO(id, start_date, location);
    }
}
```

Figure 10: Example of Polymorphism

8 EXPERIMENT

8.1 Project functions

No	Name of function	Description
1	Support for different user types	User can log in as a **Student** or a **Professor**, which affects the pricing of tickets.
2	Account register	User can register for an account, which requires them to input necessary data like name, ID, password and email.
3	Log In	User must have an existing account and provide the correct password in order to log in
4	Buy Tickets	After logging in, user can buy tickets by specifying station, ticket type, departure date, and in the case of one-way tickets, direction.
5	See Tickets	User can see all tickets that they have purchased in a summary table.

8.2 Database design

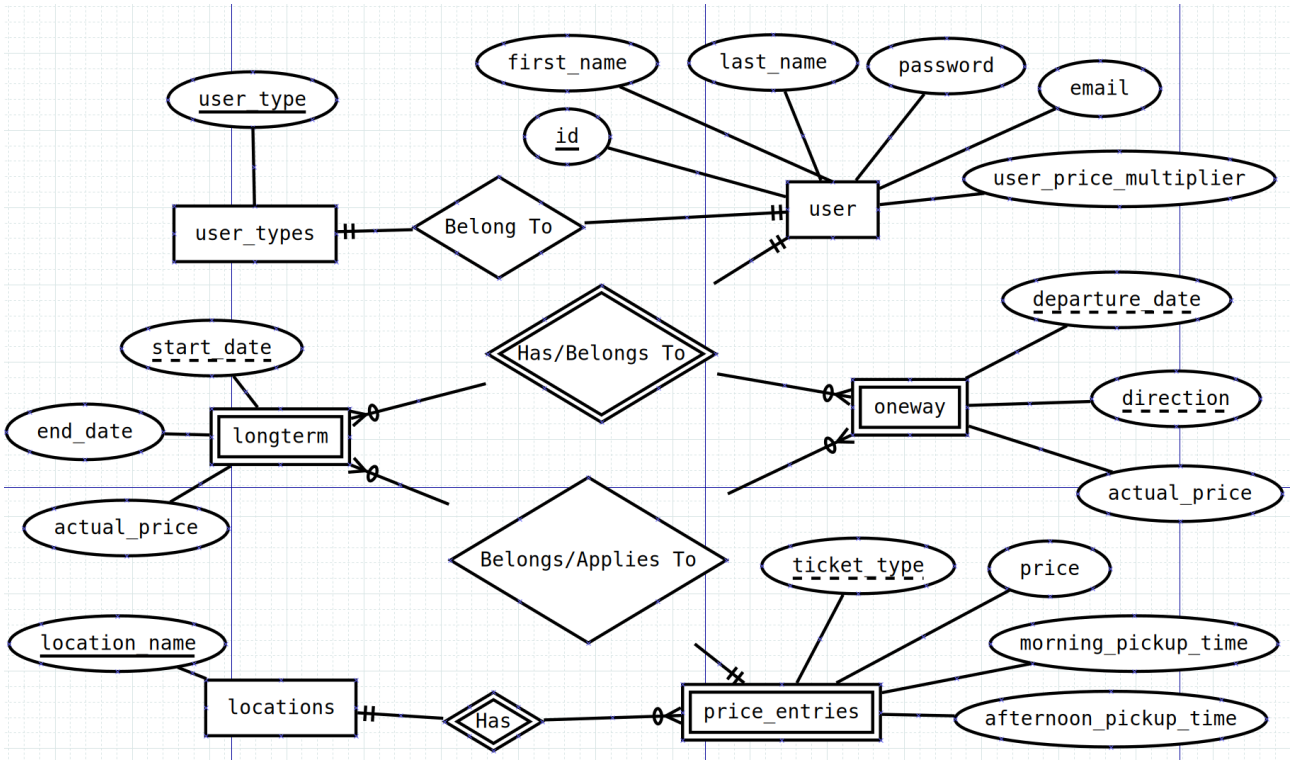

Figure 11: Entity – Relationship Diagram of Database

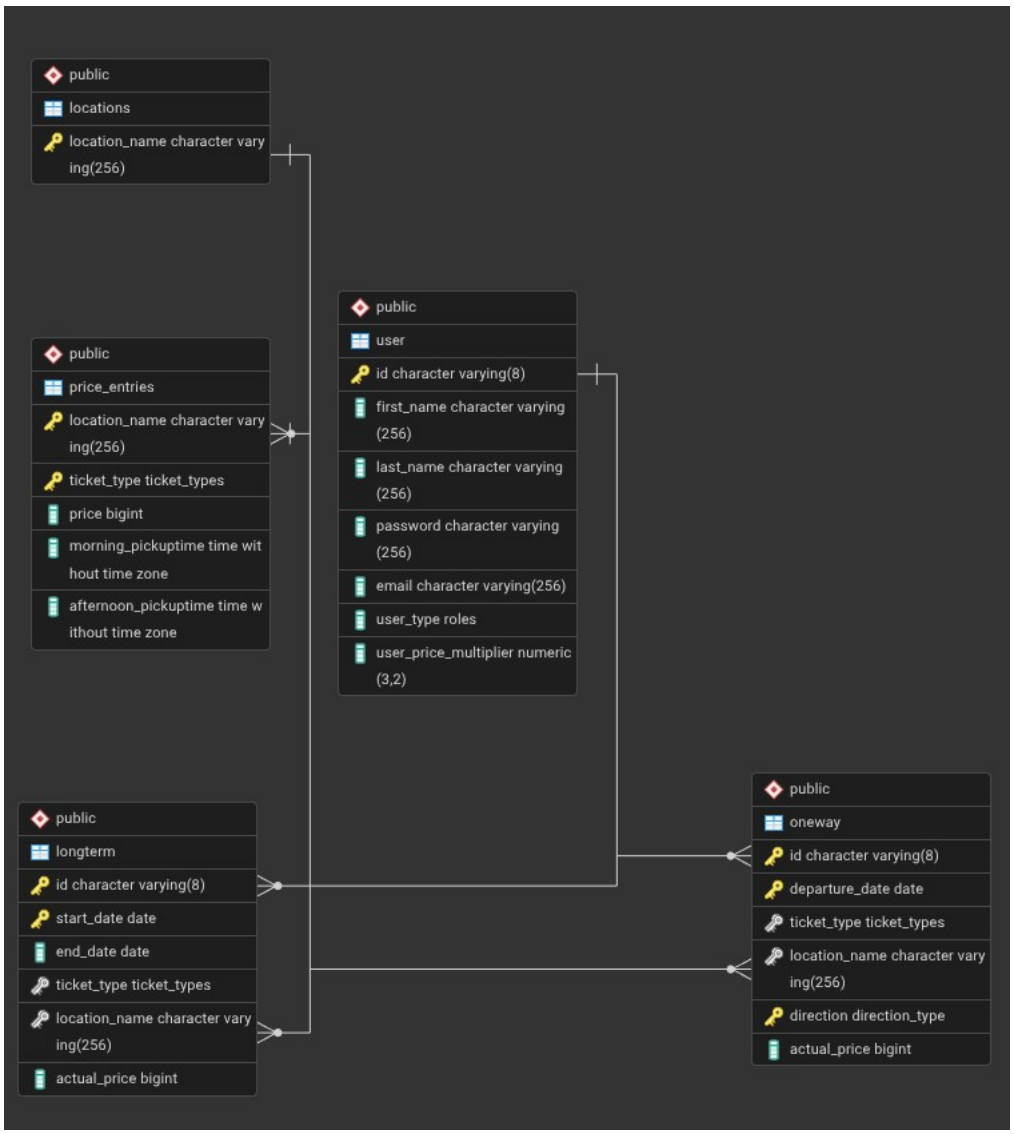

Figure 12: Relational Diagram of Database

8.3 GUI

No	Name	Description	Floats Into
1	MainMenu	- User picks between **Log In as Student** and **Log In as Professor**. - GUI keeps track of user type and passes it to the next window.	UserLogIn with user type of option picked
2	UserLogIn	- User inputs ID and password to log in. Requires existing account and valid password. - User can exit to main menu with Back. - User can create a new account with SignUp.	- MainMenu if Back. - NewUp if Signup. - UserMenu with correct credentials.
3	NewUser	- User provides necessary data (ID, first name, last name, password, email) for account registration. Rejects if ID already existed in database.	- UserLogIn if Back.
4	UserMenu	- User can buy tickets with Buy Tickets. - User can see bought tickets with See Tickets. - User can sign out with SignOut.	- UserLogIn if Sign Out. - BusBooking if Buy Tickets.
5	BusBooking	- User fills out form to buy tickets.	- UserMenu if Back

Illustrations:

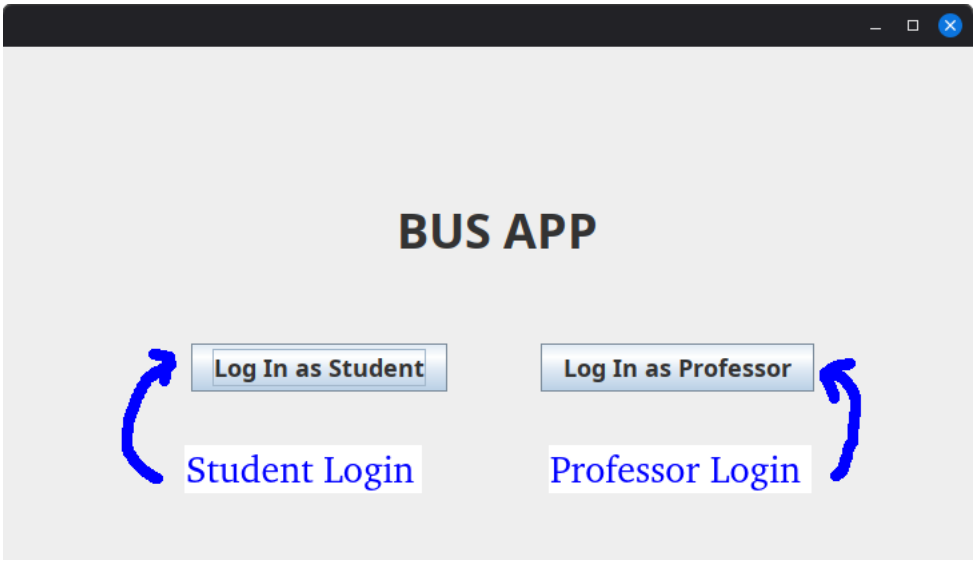

Figure 13: MainMenu

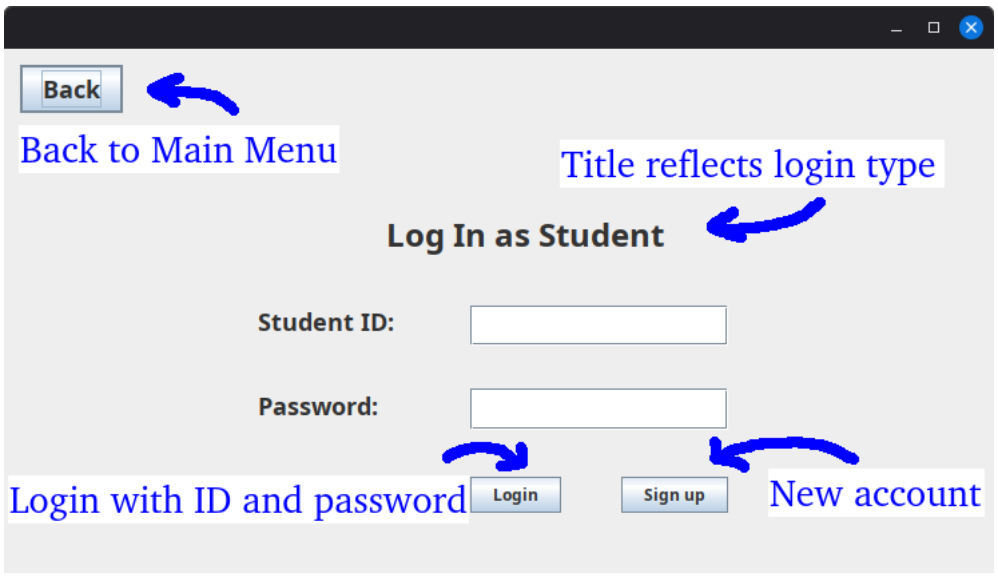

Figure 14: UserLogIn

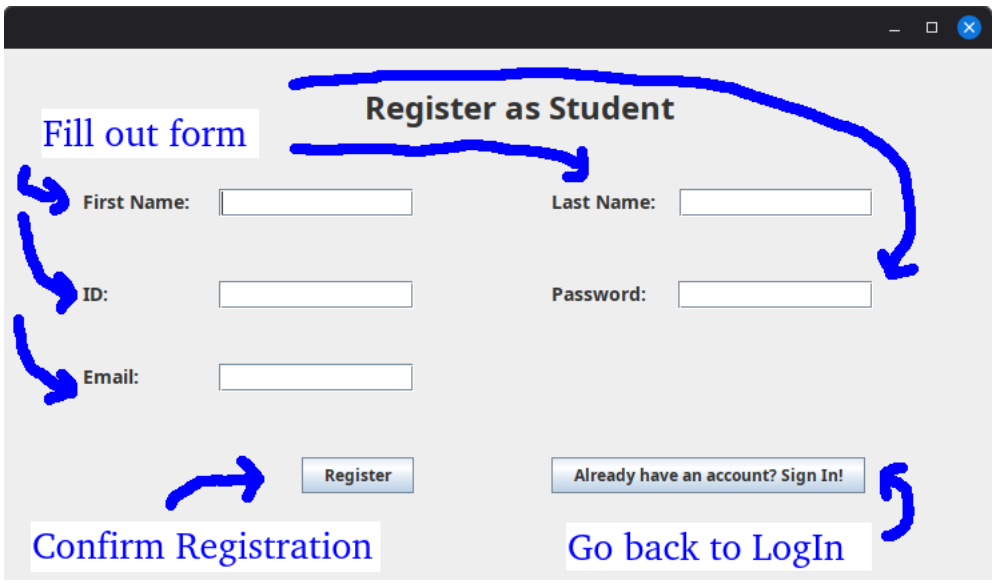

Figure 15: UserLogIn

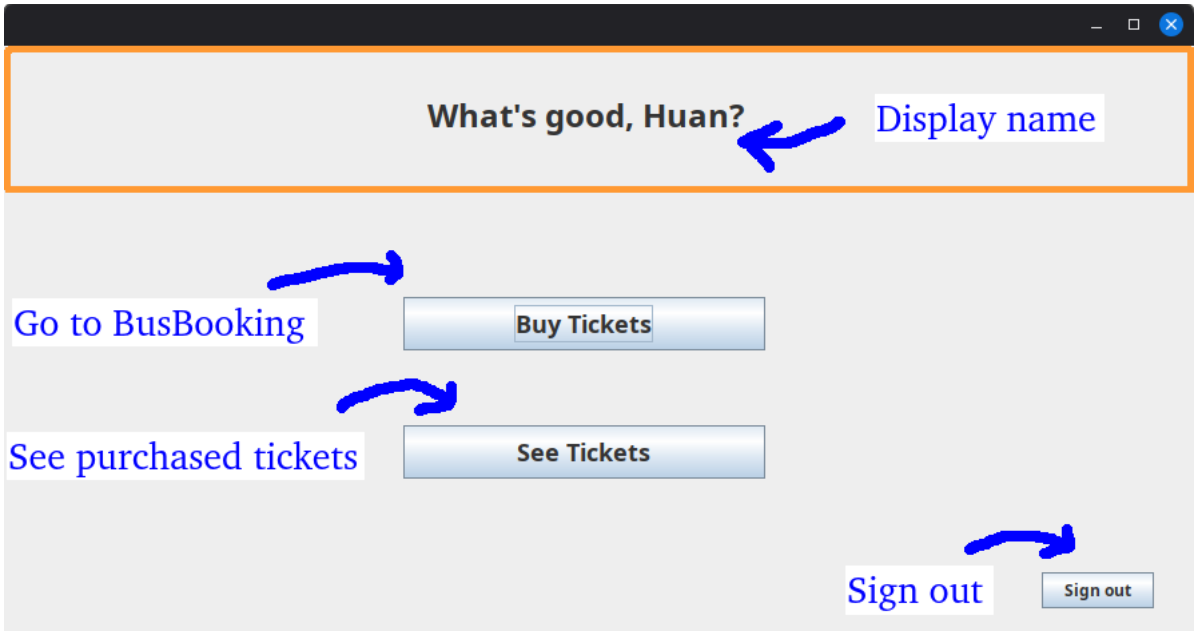

Figure 16: UserMenu

Huan's Ticket

Long-Term Tickets

ID	Type	Price	Bus Stop	Morning Pick-up	Afternoon Pick-up	Active Date	Expiry Date	Expiry Status
10423043	WEEKLY	4,500,000 VND	Turtle Lake	06:00	16:30	2025-12-01	2026-05-30	ACTIVE

One Way Tickets

ID	Type	Price	Direction	Departure Date	Departure Time
10423043	ONEWAY	150,000 VND	FROM Turtle Lake TO VGU	2025-12-04	06:45
10423043	ONEWAY	150,000 VND	FROM VGU TO Turtle Lake	2025-12-04	16:30

Figure 17: See Tickets

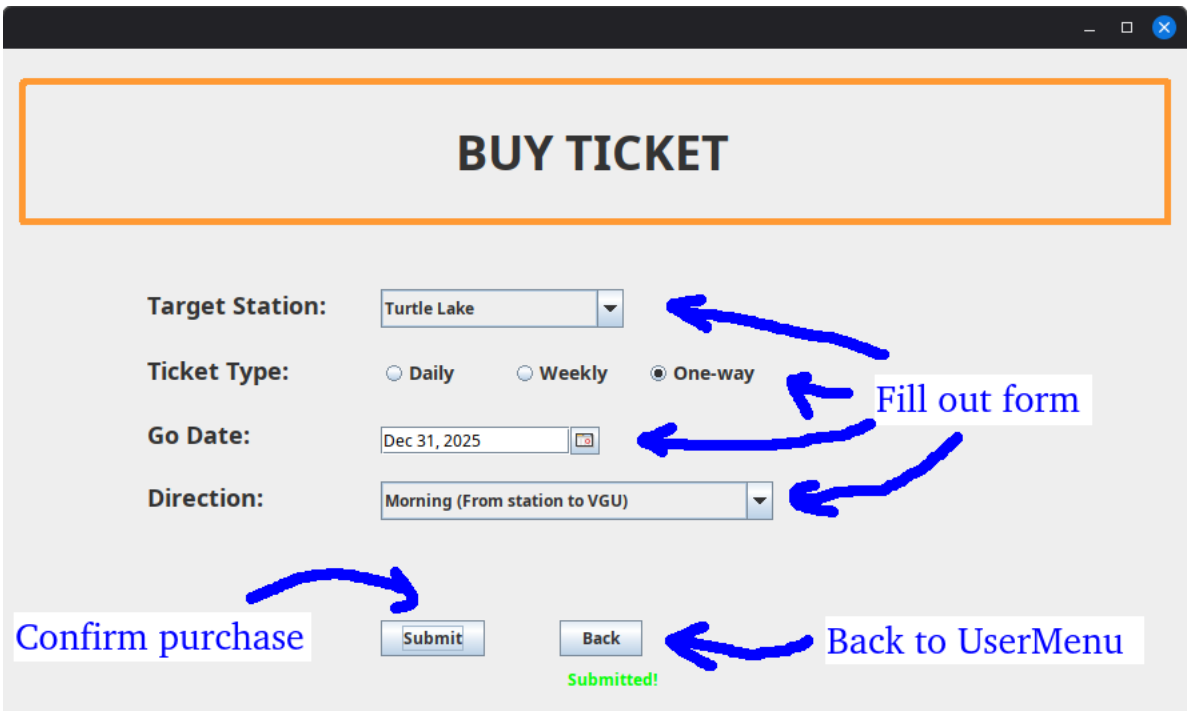

Figure 18: Buy Tickets

9 CONCLUSION

Overall, the application provides some of the most basic functionalities to be expected from a bus ticket app. The user can register for an account, log into such account with a password. Once logged in, the user can purchase tickets and see a log of all purchased tickets.

The project has been tested carefully and many safeguards have been implemented to prevent potential user errors. For example, the user cannot attempt to log into an account that has not been registered in the database. Also they cannot finalize any transactions before completely filling out the form when attempting to purchase a ticket.

The biggest downside of the project is that the graphical user interface is still very basic and admittedly uncomfortable to use, as our main focus is on the functionalities of the application, and not the form. If we ever revisit this project in the future, improving the GUI is one of our highest priorities.

DUTY ROSTER

ID	Task	In Charge	State	Note
1	Design: com.ui	Cao Dang Tri Tran Gia Huan	Done	
2	Design: com.controller.java.users	Ngo Trung Kien	Done	
3	Design: com.controller.database com.controller.java.tickets com.controller.java.ticketdetails com.database com.database.CRUD com.exceptions	Tran Gia Huan	Done	I wrote everything inside those packages and then some btw.
4	Set up, design and refine database architecture Write set up file for database Draw entity – relationship diagram Draw relational diagram	Tran Gia Huan	Done	
5	Connect GUI to controller Connect controller to database	Tran Gia Huan	Done	
6	Fix UI bugs (like adding check conditions to input fields, fixing inconsistent error messages)	Tran Gia Huan Cao Dang Tri	Done	Tri fixes design bugs. Huan fixes logical bugs
7	Set up and manage GitHub repo	Tran Gia Huan	Done	
8	Write first draft of report	Ngo Trung Kien	Done	
9	Rewrite report. Add visuals	Tran Gia Huan	Done	