Technical Report for Spartan Ride Application

The Bash Bashers (Ashmitha, Bavishna, Jihye, Shreya)

Table of Contents

1. Introduction

The Spartan Ride is a fully developed, campus-based ride-sharing platform designed to improve transportation accessibility for members of the San José State University community. Tailored to the needs of both students and faculty, the system offers a reliable, affordable, and university-operated alternative to commercial ride-hailing services, focusing on short-distance travel within and around campus.

The platform provides a streamlined digital interface through which users can easily request rides, whether commuting between campus buildings, connecting to nearby transit hubs, or attending local events. It emphasizes simplicity, transparency, and role-based access to ensure that each user group—students, faculty, drivers, and administrators—can interact with the system according to their specific needs.

Riders can sign up, log in, schedule rides, review trip history, submit feedback, and manage their wallet and account details through dedicated modules. Drivers have access to tools that allow them to view assigned routes, update ride statuses in real time, track their trip logs, calculate earnings, and monitor rider feedback. Administrators can oversee the entire system through a centralized dashboard, analyze ride data by time and region, monitor service quality, and manage user records across all roles.

By integrating key functions—ride booking, route management, user feedback, and operational analytics—into a unified platform, the Spartan Ride Management System delivers a robust, scalable, and user-friendly solution for campus mobility. It not only enhances individual convenience but also supports efficient, data-driven decision-making to help the university deliver safe, responsive, and sustainable transportation services.

2. Data Sources

Most of the tables used in the Spartan Ride were initially generated using Mockaroo (https://www.mockaroo.com), which allowed for quick prototyping of relational data through its table fabrication tools. These mock datasets provided a strong foundation for populating the operational database during development and UI testing.

However, Mockaroo's limitations—such as constraints on cross-table relationships, large-scale batch generation, and text diversity—made it difficult to simulate more dynamic or contextual data like trip logs, feedback content, or behavioral history. To address these gaps, the team utilized the educational version of GPT to generate supplementary data. This allowed for the creation of thousands of additional entries, enriched with more varied booking patterns, feedback messages, and route activity, ultimately enhancing the depth and realism of the system’s dataset.

By combining Mockaroo’s schema-driven data generation with GPT’s language-based synthesis, the team was able to produce a comprehensive dataset suitable for both functional implementation and analytical testing.

3. Application Design

The tools used during development include Python, MySQL Workbench, PyQt Designer, ERD Plus, and Jupyter Notebook. It requires additional packages to be installed:

- Pandas
- Numpy
- Matplotlib
- PyQt5 and PyQt5-Tools
- Seaborn
- MySQL Connector for Python

3.1 Operational Module

The Operational Module of the Spartan Ride serves as the core engine for day-to-day campus transportation activities. It supports three primary user roles—riders (students and faculty), drivers, and administrators—each with a dedicated interface tailored to their tasks and responsibilities.

Riders can seamlessly sign up, log in, and book rides through an intuitive interface. They are able to choose from available routes, select a specific stop and time, and confirm their reservation with a single click. In addition to scheduling rides, riders can view their trip history, leave feedback with optional star ratings, and monitor their wallet balance. The Profile page allows them to verify or update their contact details. All ride activities are logged with timestamps and status indicators, enabling transparent record-keeping and accountability.

Drivers use the operational module to access and manage their daily assignments. After logging in, drivers can view their assigned routes, track upcoming stops, and update the real-time status of ongoing rides. The My Routes page provides a breakdown of stop names, scheduled times, and passenger counts. The Trip Logs feature summarizes weekly driving activity and estimates total working minutes and salary, while the Notifications section displays rider-submitted feedback related to their recent trips. This structured workflow helps drivers stay organized and performance-aware.

Administrators are provided with a centralized control panel to oversee system-wide operations. Through role-specific dashboards, they can monitor ride volumes, booking trends, and user activity across students, faculty, and drivers. Admins also have access to data entry and maintenance screens, where they can view, edit, or delete user records. These tools enable them to ensure service reliability, balance resource allocation, and address user-reported issues effectively.

By tightly integrating rider services, driver operations, and administrative oversight, the Operational Module ensures seamless scheduling, real-time coordination, and continuous performance tracking. It forms the backbone of the Spartan Ride platform, supporting reliable, scalable, and role-aware transportation across the campus.

3.2 Analytical Module

The analytical module supports administrators and university decision-makers by transforming operational data into actionable insights. Data collected through the operational database—including ride bookings, route selections, feedback ratings, and driver performance—is regularly extracted, cleaned, and structured into an analytical data warehouse. Using a STAR schema design, the module enables time-based, user-based, and route-based analysis. This supports use cases such as tracking peak hours, identifying underused routes, and evaluating service satisfaction trends. The insights are visualized through dashboards, allowing administrators to optimize operations and plan resource allocation effectively.

4. Database Design

The Spartan Ride is built on a carefully normalized relational database that underpins the platform's reliability, efficiency, and scalability. From the initial design phase, the database schema was structured with industry-standard best practices, ensuring referential integrity, data consistency, and logical clarity across all entities.

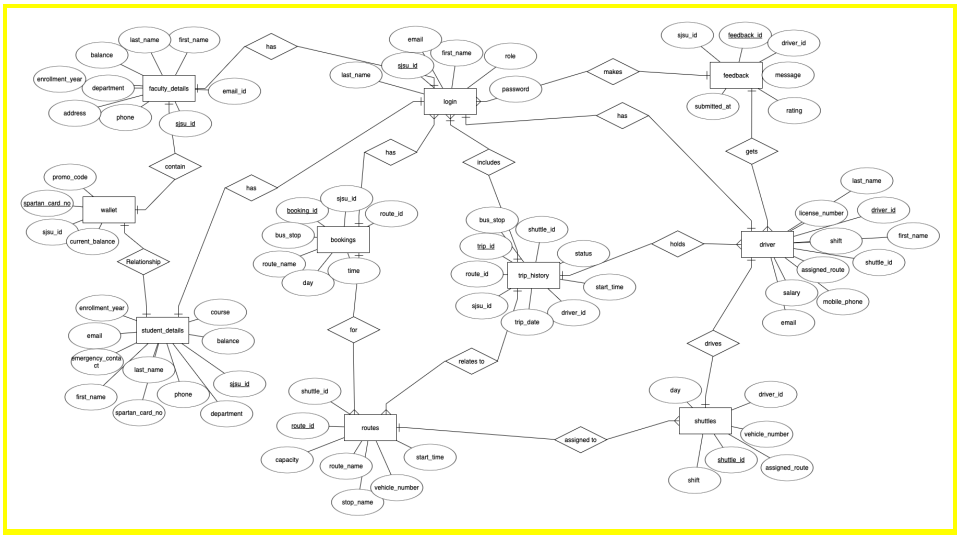
Normalization principles were rigorously applied to eliminate data redundancy and ensure that each table, such as student, driver, route, booking, and trip_history, serves a distinct and cohesive purpose. By defining clear foreign key relationships between tables (e.g., linking booking to both user and route), the system minimizes duplication and facilitates accurate, real-time queries without performance bottlenecks.

This clean, modular schema improves maintainability: developers can easily extend the system, for instance, by adding a new user role, expanding the route network, or integrating additional analytics modules, without restructuring existing tables. It also simplifies debugging and future-proofing, as updates or schema migrations can be performed with minimal disruption.

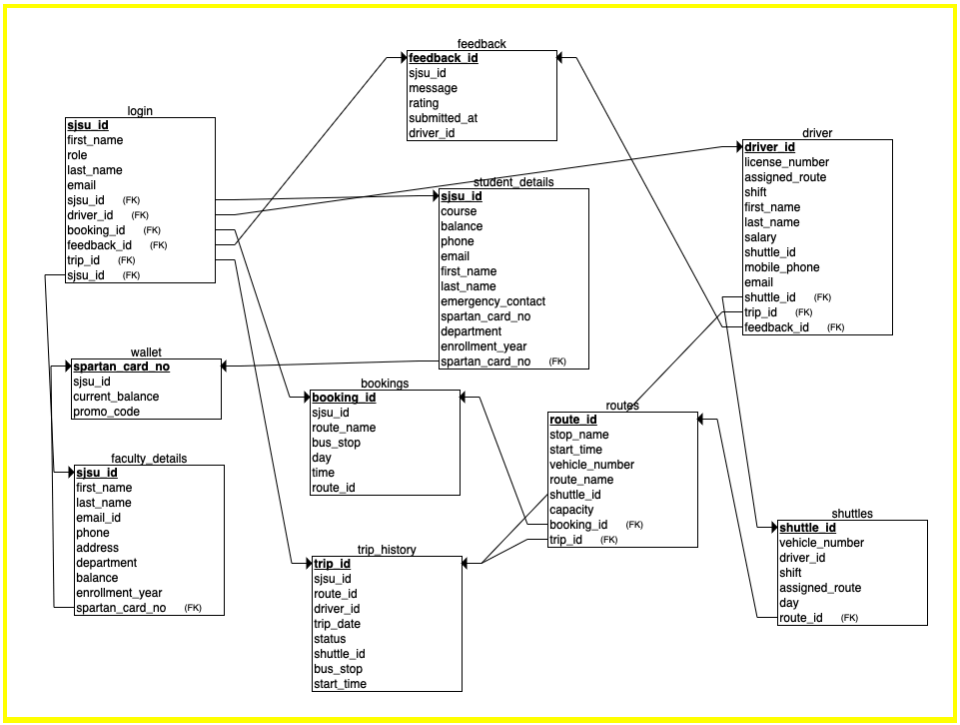
As a result, the database remains both lightweight and scalable, capable of handling increased user volume and data complexity while preserving responsiveness. This architecture supports all operational modules—rider booking, driver coordination, admin dashboards—as well as analytical processes such as ETL and dashboard visualizations.

By balancing structural rigor with flexibility, the Spartan Ride database provides a stable foundation for long-term growth, system reliability, and evolving institutional needs.

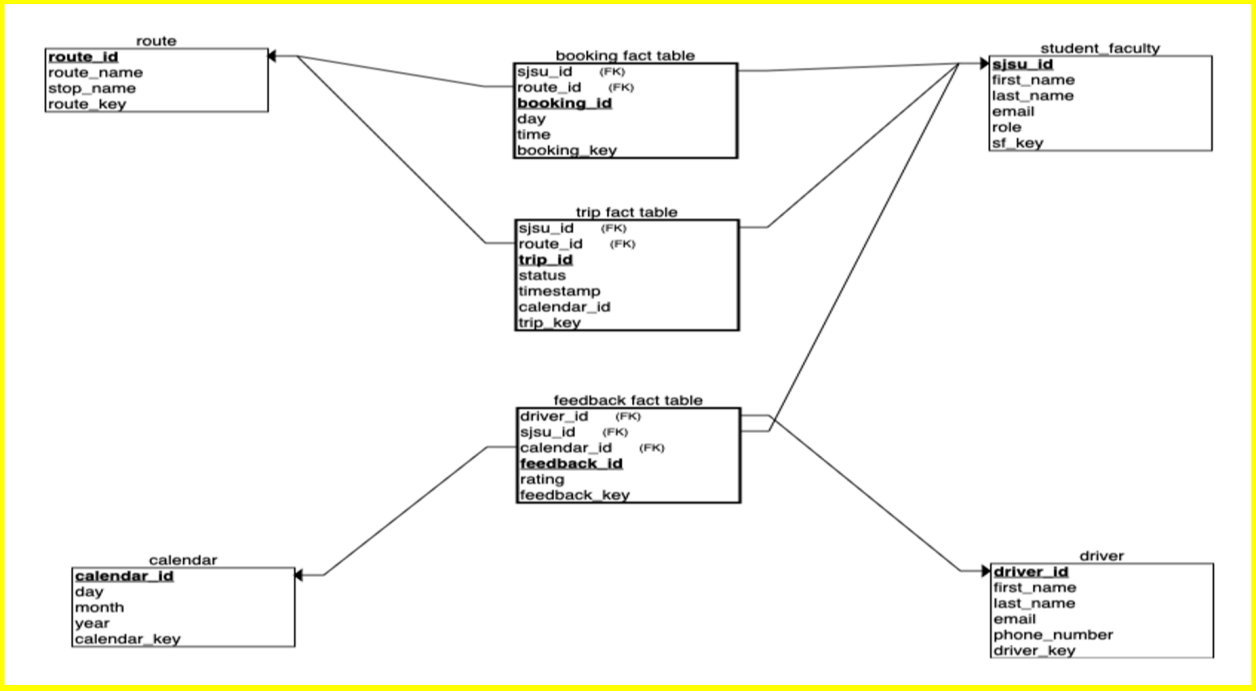
- ER Diagram



- Relational Schema



- Analytical Database STAR Schema



5. Working of the Operational Module

The Spartan Ride is structured around three key user roles: Rider, Driver, and Administrator. Each role is equipped with features tailored to its responsibilities, enabling efficient coordination and seamless operation of campus ride services.

The Rider role includes both students and faculty members who use the platform to book rides around campus and nearby areas. Riders can log in to schedule rides, review their ride history, and submit feedback. Additionally, they can access a personal profile page where they can view and manage their account details and check the current status of their wallet balance, which reflects ride credits or payments made within the platform. These tools provide transparency and control, ensuring a smooth and personalized experience for every rider.

The Driver plays a central role in the operational flow of the system. Drivers can view their assigned routes and update ride statuses in real time. They can also access their ride log to track past trips and check a personal profile page that includes detailed information such as contact info and vehicle data. Importantly, drivers can review their salary information and see the ratings and feedback they've received from riders, giving them insight into their performance and helping them maintain high-quality service.

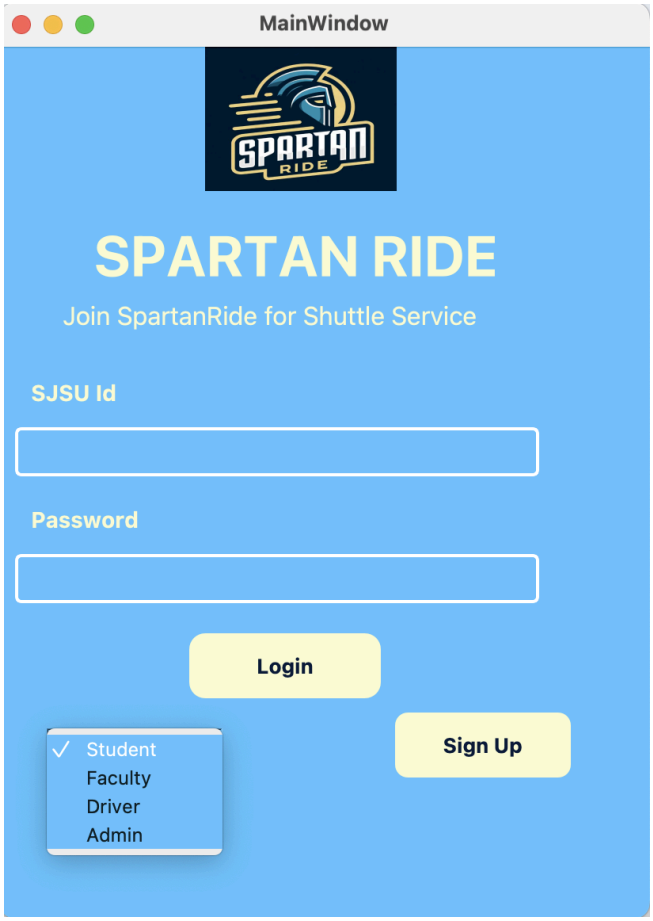
The Administrator oversees system usage and monitors overall performance. While not directly involved in ride execution, administrators use analytics dashboards to track system activity, including ride volume and usage patterns by time and region. These insights support data-driven decision-making and operational improvements.

By assigning clearly defined and functional capabilities to each user type, the Spartan Ride ensures a well-integrated, scalable, and user-focused mobility solution for the university community.

6. Specifications and Usability of Operational Module

6.1. Login and Sign-Up Page

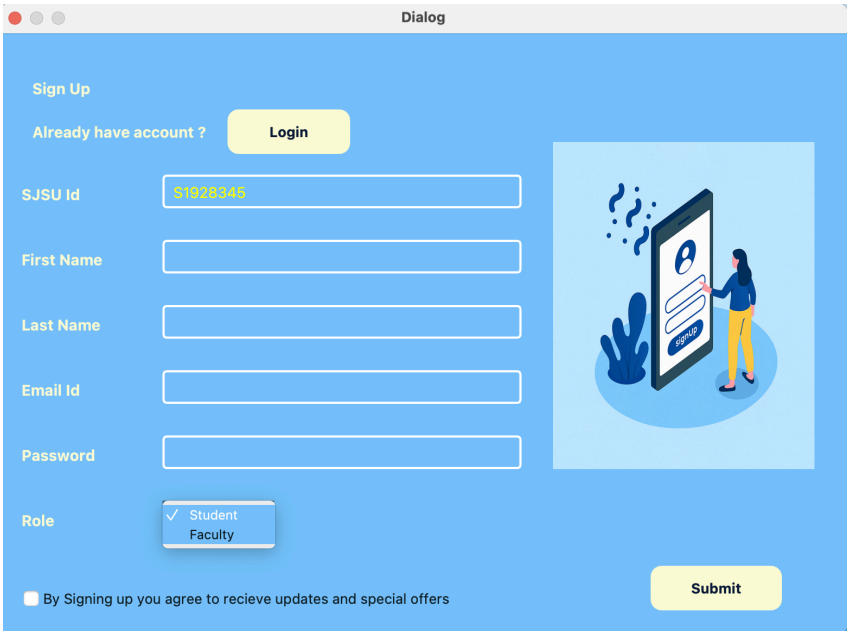
- Login Page



The login page is the entry point for all users of the Spartan Ride Management System. It provides a clean and intuitive interface where users can access the system by entering their SJSU ID, password, and selecting their role from a dropdown menu. The system supports multiple user types, including students, faculty members, drivers, and administrators.

Upon successful login, users are directed to dashboards or modules tailored to their role, ensuring role-based access control and a personalized experience. New users can click the “Sign Up” button to create an account. This design ensures a secure, streamlined authentication process while clearly segmenting access across different user groups.

- Sing-Up Page



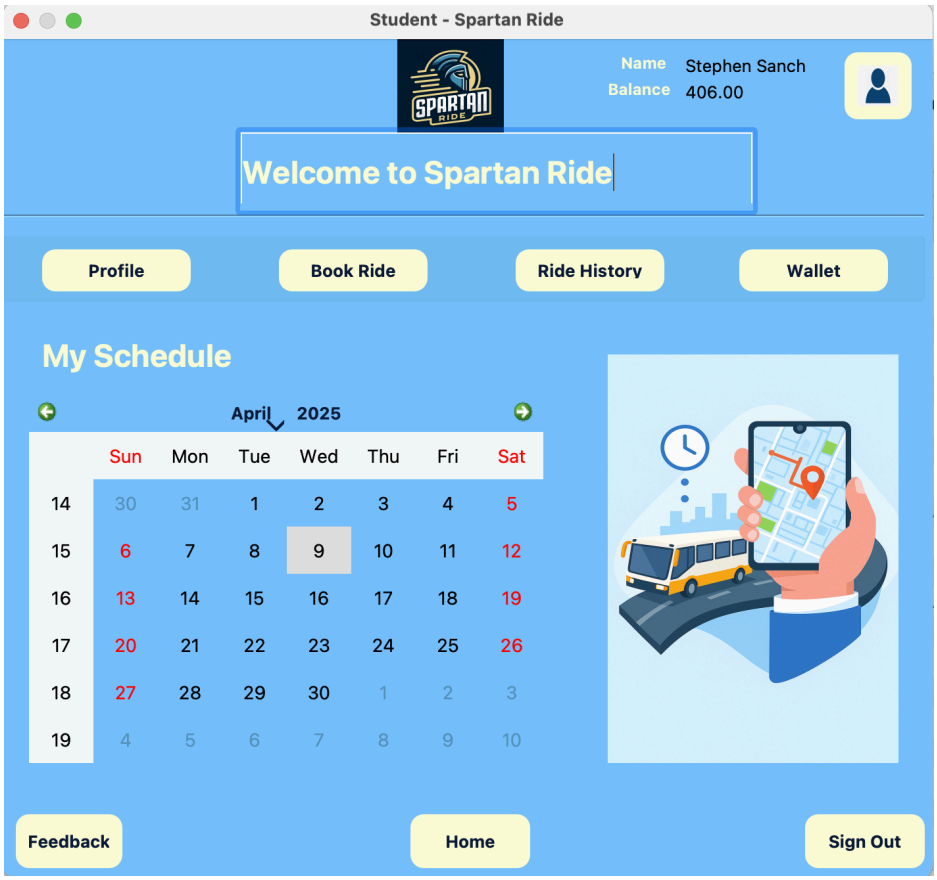
The sign-up page enables new users to register for the Spartan Ride platform. To create an account, users are required to provide their SJSU ID, first name, last name, email address, and a password. In addition, users must select their role from the dropdown menu, which currently includes options for students and faculty.

Once the form is completed, users can click the Submit button to register. A checkbox at the bottom allows users to opt in to receive updates and special offers. The layout also includes a quick link to the login page for returning users, enhancing navigation and usability.

The clear and minimal design ensures a smooth onboarding process, while role-based registration helps tailor the user experience right from the start.

6.2. Student and Faculty Page

- Profile Page

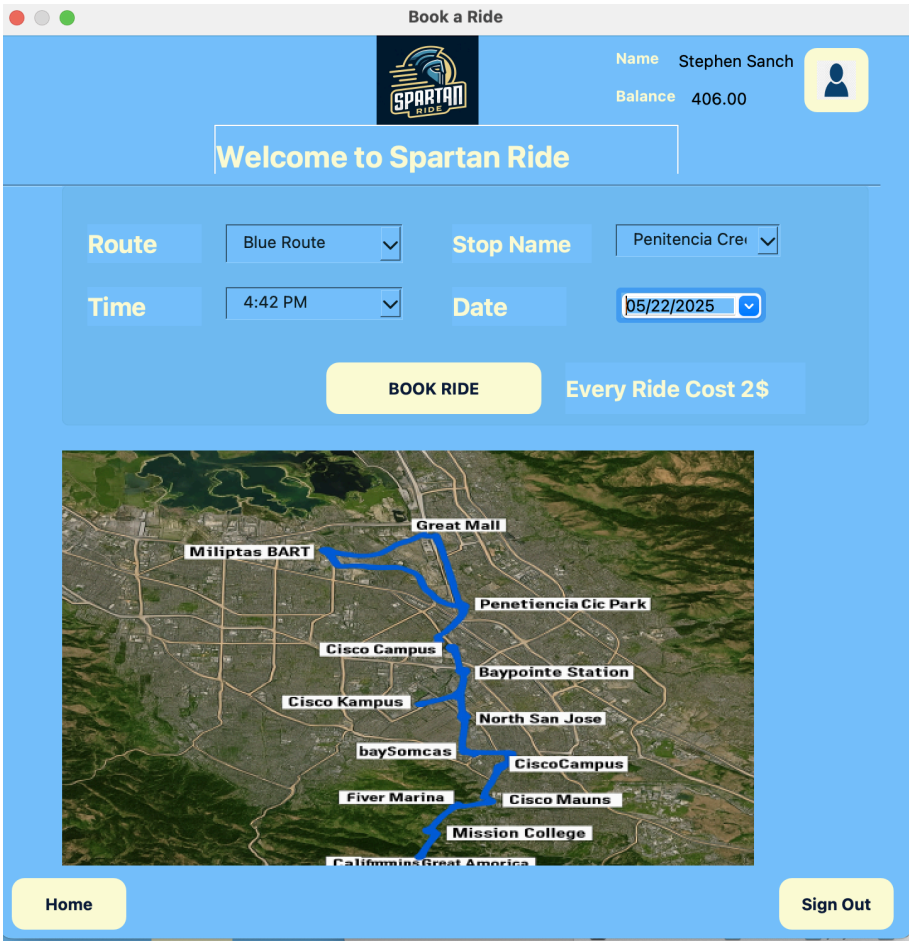


The Student Profile page offers users a clear and structured view of their registered personal information. Upon launching the dialog, the system automatically connects to the backend MySQL database and retrieves the user's data using their unique SJSU ID. The application identifies the user's role by checking the prefix of the ID (e.g., "S" for students), and accordingly queries either the student or faculty table.

The retrieved fields include the first name, email, phone number, department, enrollment year, and course (for students). These details are displayed using non-editable labels in a dialog layout, built with Profile_DialogBox.ui. The data-loading logic is implemented in the Profile_DialogBox.py module, where a dynamic SQL query fetches the correct user profile and populates the interface.

This design ensures users always see accurate, up-to-date information without manually entering any data. The system reads the database configuration from config.ini, connects securely, and handles the entire data fetching process in the background.

- Book Ride

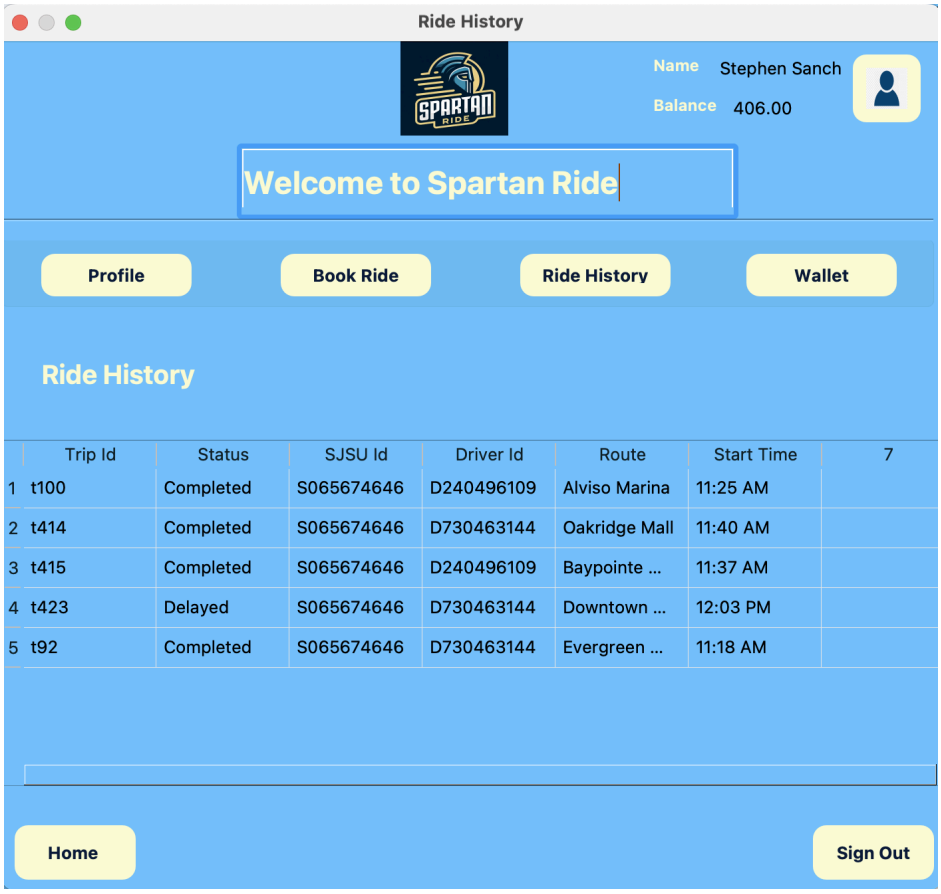


The Book a Ride page enables students to conveniently schedule shuttle rides by selecting their preferred route, stop, time, and date through an intuitive dropdown interface. Upon loading, the system dynamically retrieves the student's name and wallet balance using their SJSU ID by querying the login table via the load_user_info() method defined in the RouteDialog class. This ensures that all ride bookings are tied to the correct user identity and wallet account.

The interface is particularly useful for students who need reliable transportation to and from key campus locations. For example, a student with a 5:00 PM class at Cisco Campus can use this page to select the Blue Route, choose Cisco Campus as the stop, pick 4:42 PM as the departure time, and set the date. With one click on the "BOOK RIDE" button, the system confirms the reservation, deducts the fare from their balance, and logs the trip to the backend.

To assist students with their planning, a route map is provided at the bottom of the page, visually indicating all available shuttle stops. The design ensures that ride scheduling is fast, error-free, and well-integrated with other features like user profiles, ride history, and wallet tracking. The page is implemented using the Student_RouteDetails.ui layout and managed by the Student_Dialog_Route.py script.

- Ride History

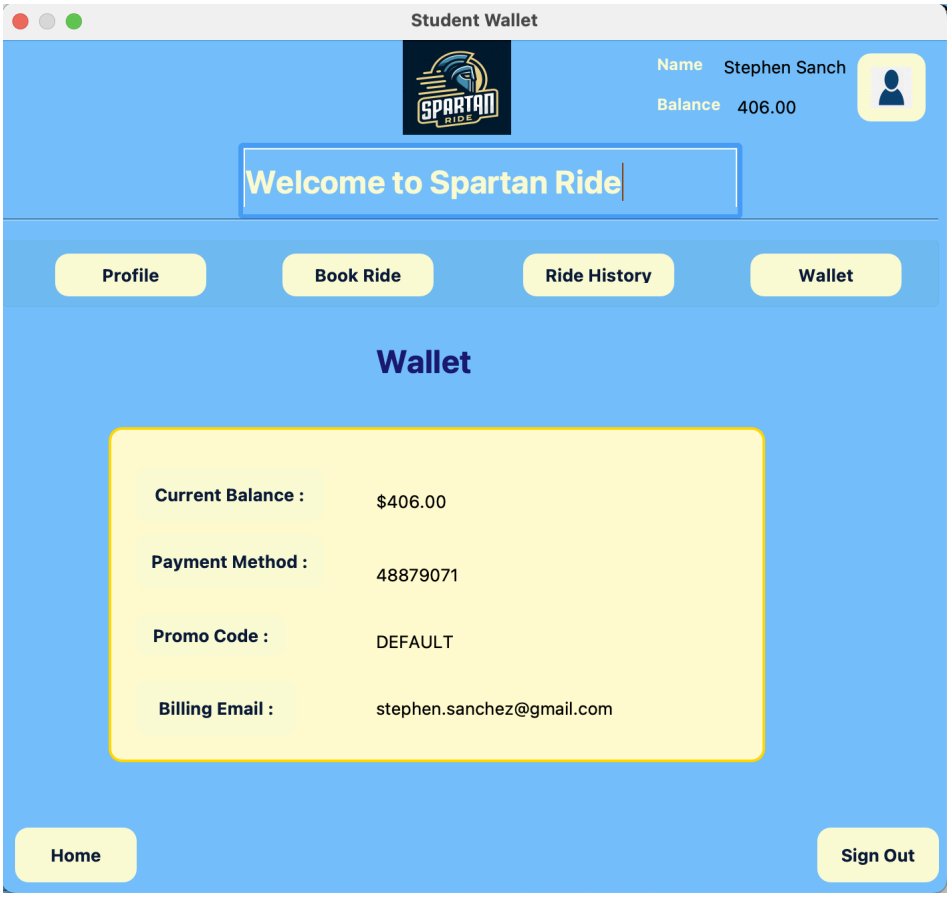


The Ride History page provides students with a detailed summary of their past shuttle rides. When the dialog is launched, the system retrieves the user's trip history from the trip_history table in the backend database using their SJSU ID. Key ride attributes such as trip ID, ride status, driver ID, stop location, and start time are displayed in a sortable table for easy reference.

This page allows students to quickly review their travel records, especially in cases of delays or service complaints. For example, if a student experienced a delay on a recent shuttle trip, they can refer to this history to confirm the trip ID and departure time, which can then be referenced when providing feedback or requesting a refund.

In addition to historical ride data, the interface also displays the user's name and wallet balance, offering quick access to other functions like Profile, Book Ride, and Wallet through dedicated navigation buttons. The page is implemented using the Student_Dialog_RideHistory.ui file and powered by logic defined in Student_Dialog_RideHistory.py.

- Wallet

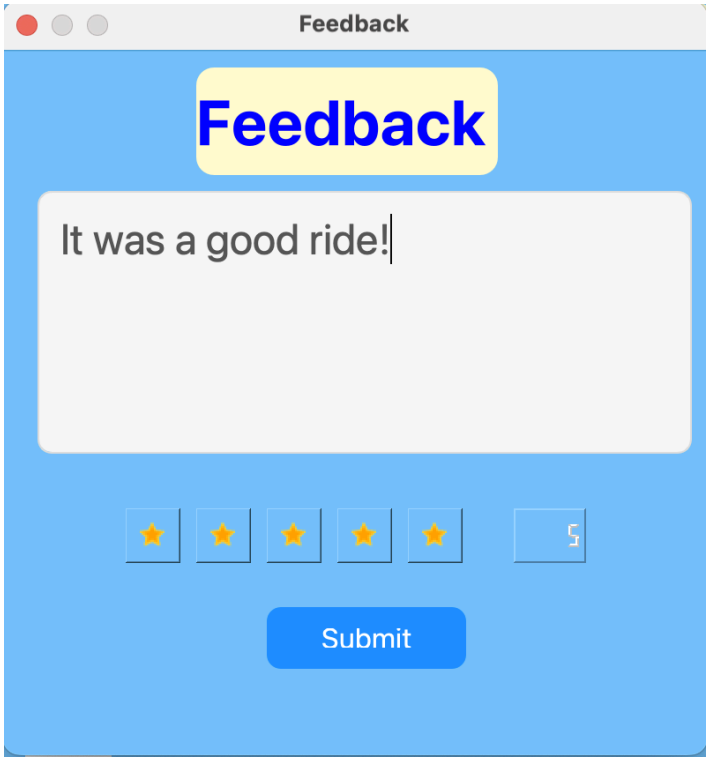


The Wallet page provides users with a snapshot of their financial information within the Spartan Ride system. Upon launching the interface, the application queries the backend wallet table using the student's SJSU ID to retrieve the current balance, payment method, and promo code. Additionally, the user's billing email and full name are fetched from the login table. This process is handled in the WalletDialog class, implemented in Student_Dialog_Wallet.py.

This page is especially useful when students want to verify their available funds before booking a shuttle ride. For example, before selecting a route, a student can quickly check their balance to ensure they have enough credits to cover the ride fare. If needed, they can update their payment method or promo code externally and ensure it is reflected here.

The interface is simple yet functional, displaying all relevant fields in a structured yellow-highlighted panel. It also integrates with the rest of the platform, offering quick navigation to the Profile, Book Ride, and Ride History pages. Overall, this module enhances transparency and financial control for users managing their campus transportation needs.

- Feedback



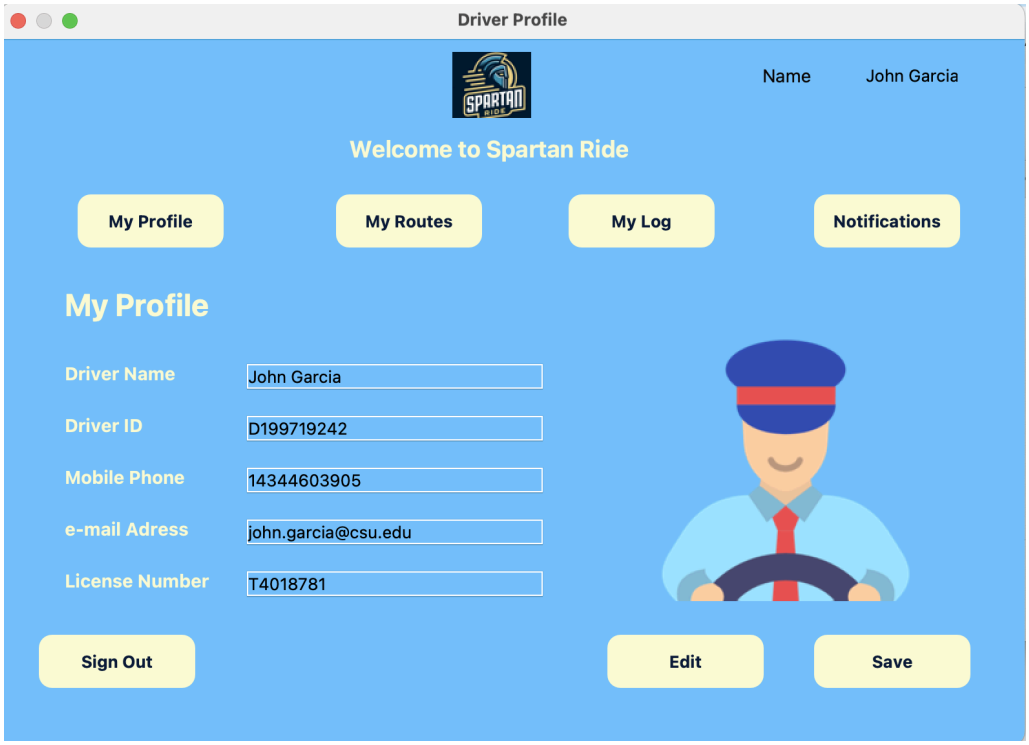
The Feedback page allows students to share their ride experience by submitting a short comment and selecting a star rating from 1 to 5. The interface consists of a text box for open-ended feedback, five clickable star icons implemented using QToolButton, and an LCD display that reflects the selected rating in real time. These elements are managed by the FeedbackDialog class in Feedback_Dialog.py.

When a student completes a ride—whether satisfied or encountering issues—they can access this page to leave input. For instance, after arriving on time and receiving friendly service, a student might write “It was a good ride!” and click the fifth star. The set_rating() method updates the rating display and stores the selected value, while clicking the Submit button triggers on_submit(), saving the feedback to the backend database with the associated user ID.

This feedback system supports continuous service improvement by allowing administrators and drivers to receive direct user input. It also adds a layer of accountability and transparency, helping maintain quality standards across the Spartan Ride platform.

6.3. Driver Page

- Driver Profile



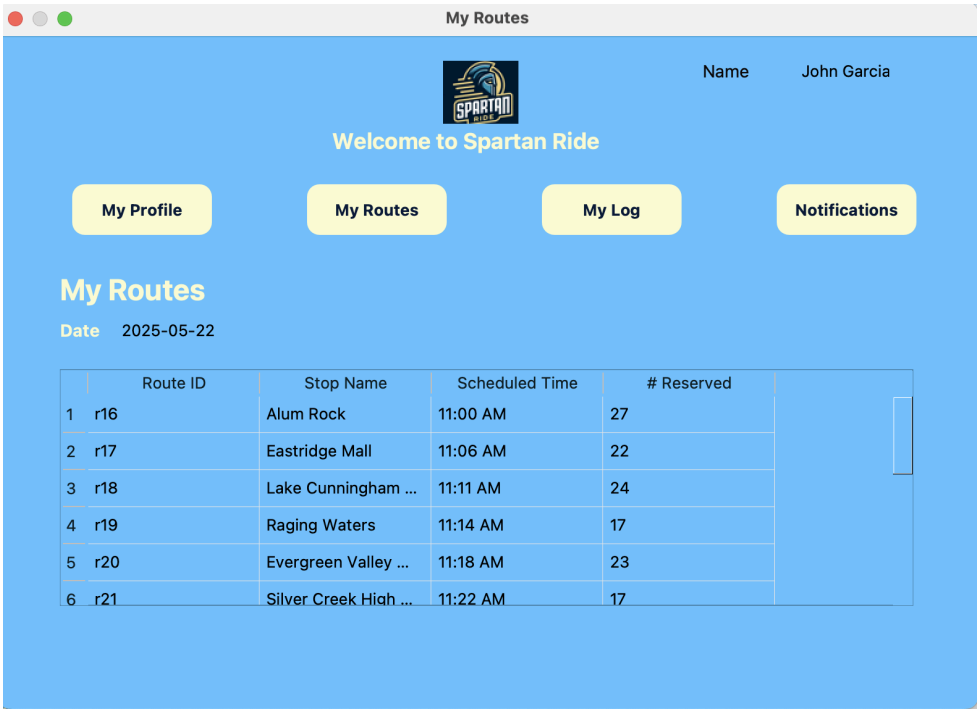
The Driver Profile page provides shuttle drivers with a centralized view of their personal and professional information within the Spartan Ride system. When a driver logs in and accesses this interface, the system automatically queries the driver table using their driver id, and displays fields such as Driver Name, Driver ID, Mobile Phone, Email Address, and License Number.

This page is implemented using the Driver_MyProfile.ui layout, and the data fetching is handled by the get_driver_profile() method defined in backend_driver_ui.py. The layout also includes Edit and Save buttons, allowing drivers to review and update their information directly within the platform—supporting accuracy and operational readiness.

A typical use case might involve a driver noticing that their contact number has changed. They can navigate to the profile page, click Edit, update their phone number, and then save the change. This update ensures that route assignments, payment details, and notifications are properly aligned with the most recent contact info.

The profile interface is also integrated with other key driver features such as My Routes, My Log, and Notifications, enabling seamless transitions and improving the efficiency of day-to-day operations.

- My Routes

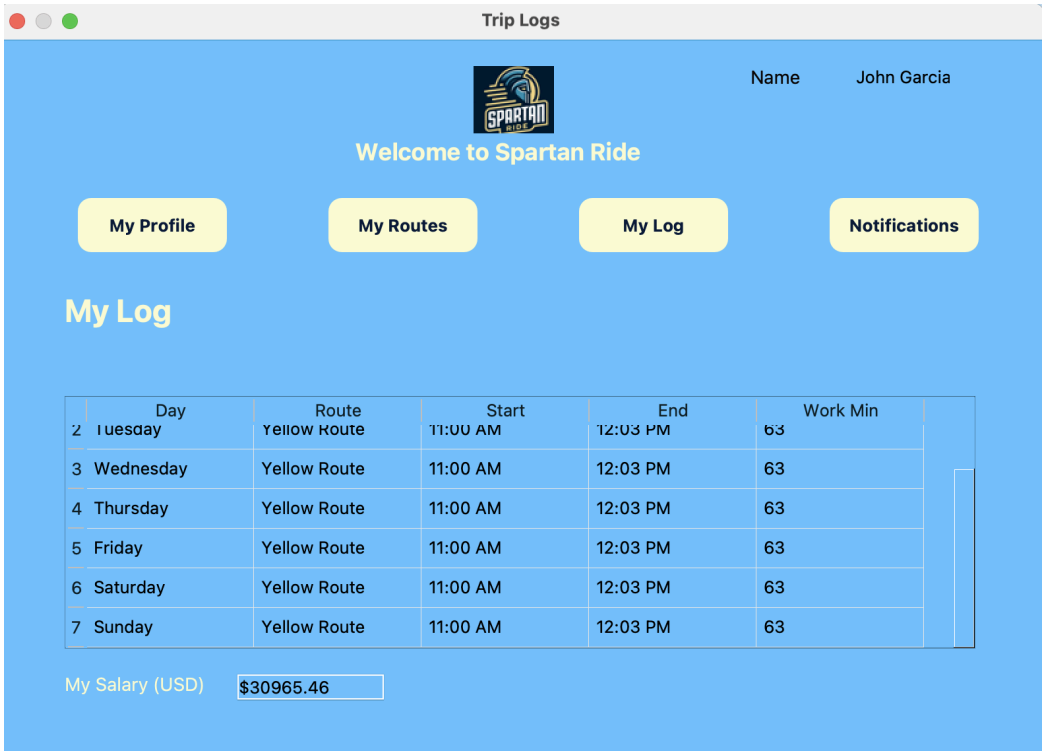


The My Routes page helps shuttle drivers efficiently manage their daily operations by presenting a detailed schedule of their assigned stops and reservation counts. Upon selecting a date, the system retrieves the driver's assigned route and shift from the driver table using their driver_id. It then queries the route table to fetch all relevant stops and estimated times, joining with the booking table to calculate how many passengers have reserved each stop.

This logic is implemented in the get_routes_by_date() function within backend_driver_ui.py, while the visual layout is built from Driver_MyRoutes.ui. For instance, if a driver is assigned the Blue Route for the morning shift on May 22, they will see a table listing stops like Alum Rock, Lake Cunningham, and Silver Creek High, along with exact times (e.g., 11:00 AM, 11:11 AM) and the number of passengers per stop.

This feature ensures that drivers can plan their timing, anticipate passenger volume, and deliver service efficiently. It also improves reliability by helping drivers quickly identify which stops may need more attention due to higher reservation numbers.

- Trip Logs



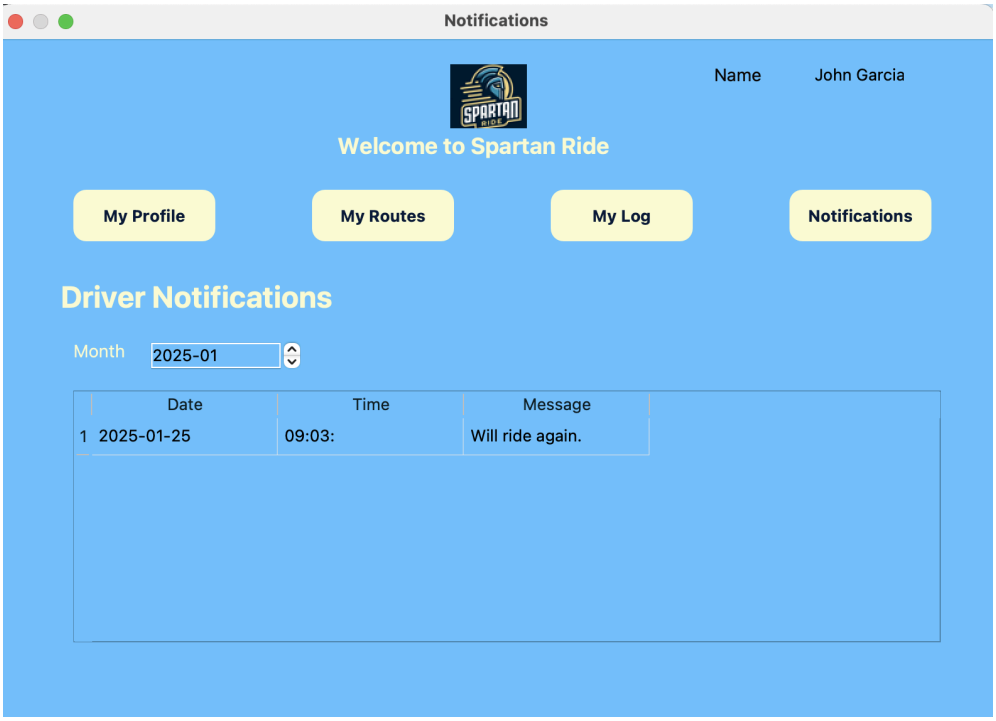
The Trip Logs page provides drivers with a detailed summary of their weekly driving schedule and associated compensation. Upon accessing this interface, the system uses the driver's ID to fetch their assigned route and shift from the driver table. It then queries the route table to determine the earliest departure and latest arrival times across that route and shift.

These times are parsed and processed to calculate the daily working minutes, which are repeated across the week to generate a complete 7-day log. This process is handled in the get_trip_logs() function in backend_driver_ui.py. The UI layout is managed by Driver_MyLog.ui, presenting columns for Day, Route, Start Time, End Time, and Work Minutes.

For example, if a driver's route starts at 11:00 AM and ends at 12:03 PM, the system calculates a 63-minute work duration and populates that value for each day from Monday to Sunday. The total estimated salary is then shown below the table, reflecting cumulative work.

This feature enables drivers to monitor their weekly engagement, verify compensation, and raise concerns if discrepancies are observed. It also promotes transparency and time accountability within the system.

- Notifications



The Notifications page serves as a feedback inbox for drivers, allowing them to view comments submitted by riders. Upon loading the page, the system retrieves all feedback messages related to the driver from the feedback table, filtered by their driver_id. Each feedback entry contains a timestamp (submitted_at) and a comment message.

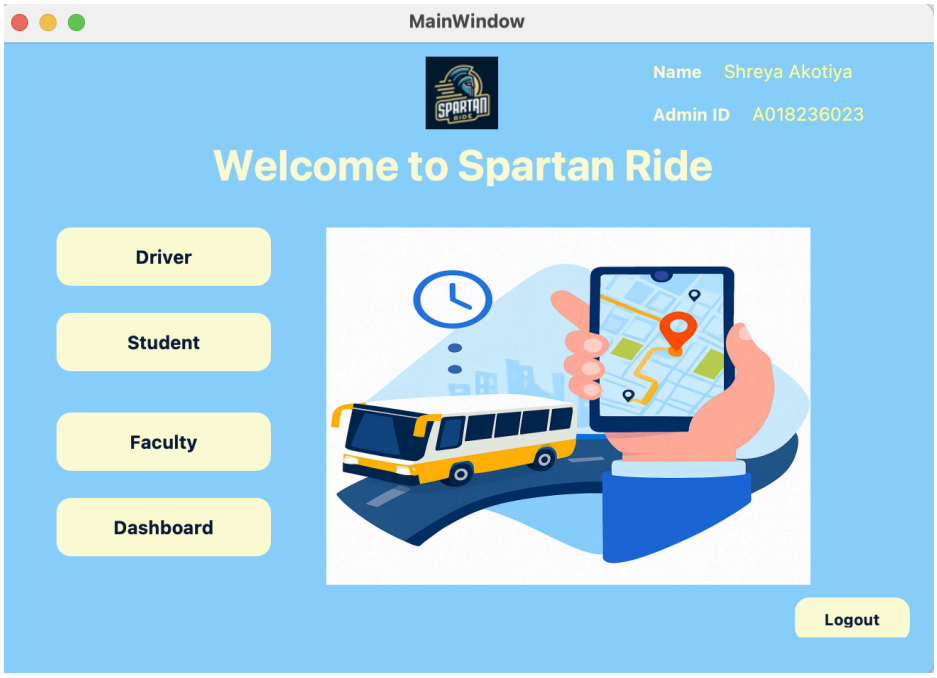
These entries are processed by the get_notifications() function in backend_driver_ui.py, which formats each message into date, time, and content fields. The UI (Driver_Notification.ui) displays the messages in a simple, scrollable table, with a dropdown control that enables drivers to filter notifications by month.

For example, if a rider leaves the comment “Will ride again” on January 25, 2025 at 09:03, this entry appears in the table. This feedback loop allows drivers to gain direct insight into service satisfaction, improve their performance, and track patterns over time.

By combining transparency and simplicity, the Notifications module contributes to service quality assurance and reinforces communication between drivers and riders within the Spartan Ride ecosystem.

6.4. Admin Page

- Admin Main Window



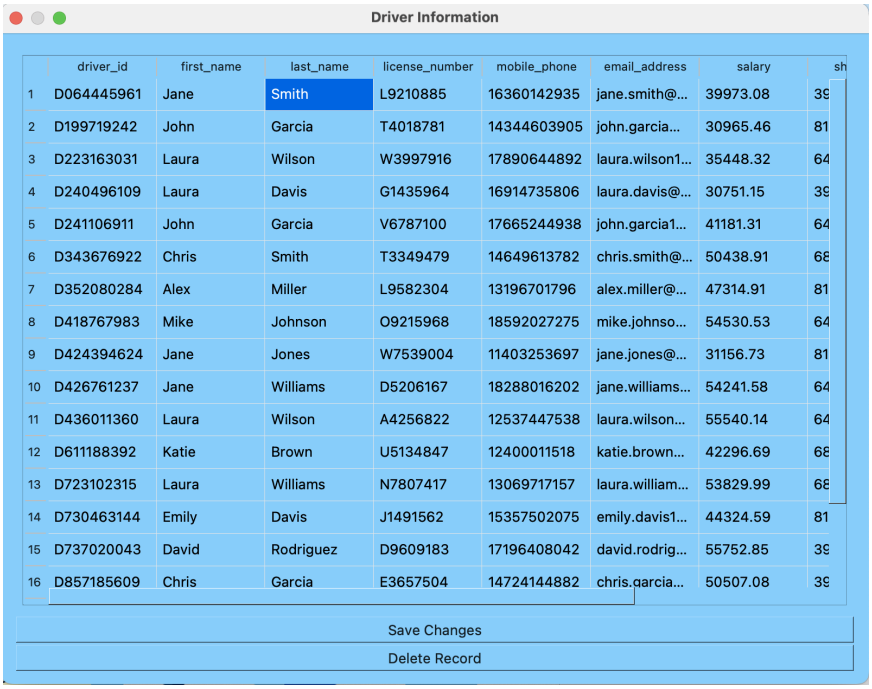
The Admin Main Window acts as the central control panel for system administrators. From this interface, admins can manage and access user information across all roles—Drivers, Students, and Faculty—as well as analyze overall system metrics via the Dashboard module.

The page layout includes a welcome message personalized with the admin's name and Admin ID, along with navigation buttons for each functional section. When the Driver, Student, or Faculty button is clicked, the corresponding user interface (e.g., DriverDialog, StudentDialog) is launched in a modal window. These modules allow the admin to view and potentially update information for that user group.

For example, if an admin receives a complaint about driver behavior, they can click Driver, open the respective dialog, and verify the driver's route assignments and feedback. Similarly, the Dashboard button opens a performance overview window, where ride statistics, user engagement trends, and booking activity may be analyzed.

This main window, implemented in the Ui_Admin class of admin_window.py, is built on the admin.ui layout and connects seamlessly to the broader administrative functions. It provides a clean, role-based entry point for managing the Spartan Ride platform.

- Driver Information



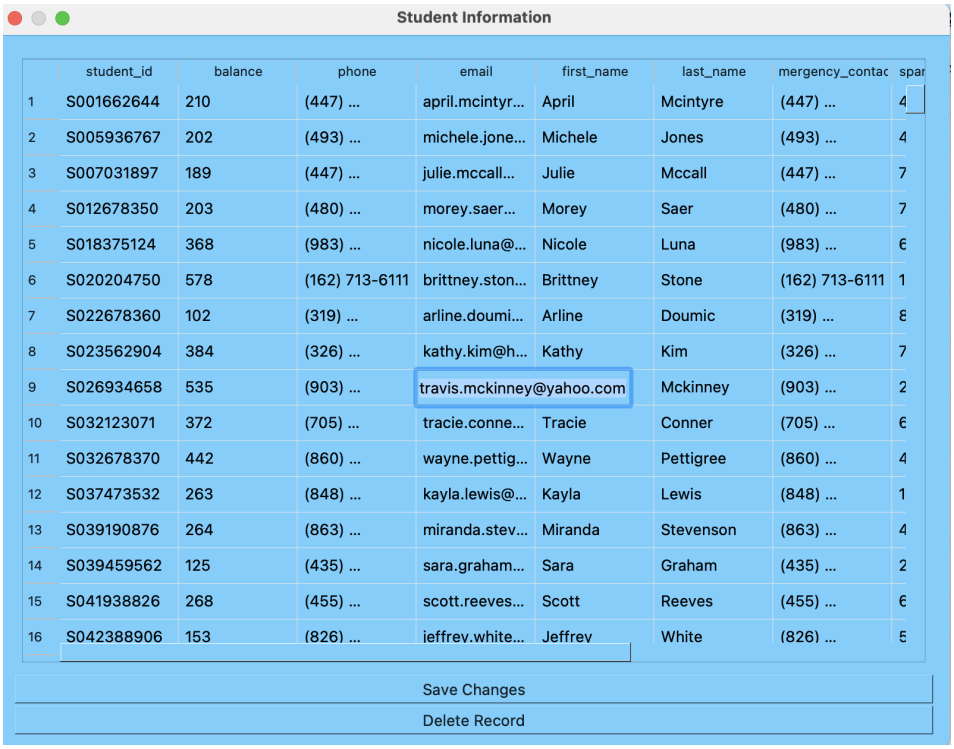
The Driver Information page allows system administrators to access, edit, and manage all registered driver data. Upon loading the interface, the system connects to the backend database and retrieves all records from the driver table using a SELECT * FROM driver SQL query. The result is displayed in a scrollable, editable QTableWidget layout.

Each row corresponds to a registered driver, with columns such as Driver ID, First and Last Name, License Number, Phone Number, Email, Salary, and Shift Assignment. The admin can directly modify any cell—for example, to update a phone number or correct a spelling error—and then click Save Changes to commit the edits to the database. This functionality is handled by the save_changes() method.

In a use case where a driver's license number was recently updated or their salary changed due to an internal policy, the admin can make those edits directly in the table and save them in real-time. Similarly, if a driver has left the organization, the admin can select the corresponding row and click Delete Record to permanently remove it from the system, via the delete_selected_rows() function.

This screen, implemented within the DriverDialog class in Admin_drivery.py, combines full visibility with simple edit-and-save interactivity, streamlining administrative control over the Spartan Ride workforce.

- Student Information



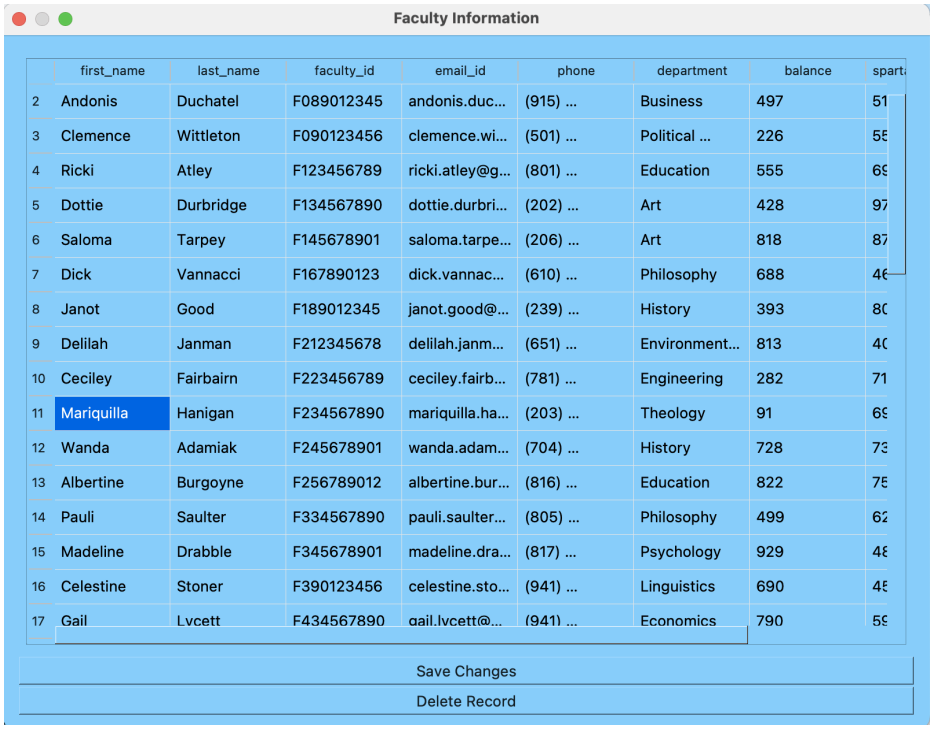
The Student Information page provides administrators with full visibility into all registered student records. Upon launching the interface, the system executes a SELECT * FROM student query and retrieves the data from the backend database. The results are displayed in a QTableWidget, where each row corresponds to one student.

Displayed fields include Student ID, Wallet Balance, Phone Number, Email, First and Last Name, Emergency Contact, and Spartan Points. The administrator can directly edit any of these values—for instance, updating contact information or correcting a misspelled name—and save the changes using the Save Changes button, which triggers the save_changes() function to commit updates back to the database.

In a typical use case, a student might request that their emergency contact be updated. The admin can find the student's record, edit the field inline, and click Save Changes to immediately update the system. Similarly, the Delete Record button allows the administrator to remove a student from the database entirely, ensuring that records remain accurate and up to date.

The feature is implemented within the StudentDialog class in Admin_student.py, offering a direct, spreadsheet-like editing interface for fast and effective student management.

- Faculty Information



The Faculty Information page provides administrators with direct access to all registered faculty member records within the Spartan Ride system. The page loads all data from the faculty table using a SELECT * FROM faculty query and displays the result in an editable table interface powered by QTableWidget.

Displayed fields include Faculty ID, First and Last Name, Email, Phone Number, Department, Wallet Balance, and Spartan Points. This table layout allows admins to directly edit any field—for example, correcting a faculty member's email or updating their department—and click Save Changes to push the updates back to the database.

In a practical scenario, suppose a faculty member's department affiliation changes from Political Science to Public Policy. The administrator can locate the record, update the "department" cell, and save the changes in real time. If a faculty member no longer uses the system, the admin can use the Delete Record button to remove that user completely.

This interface is implemented in the FacultyDialog class in Admin_facultyy.py and offers a streamlined solution for maintaining data quality, consistency, and administrative control over faculty records.

- Dashboard Page

The Dashboard page offers administrators a comprehensive overview of the Spartan Ride system's operational and user metrics through a series of visualizations. Implemented in the DashboardDialog class within admin_dashboard.py, the interface connects to the database using a secure configuration and dynamically loads data into multiple plots using pandas and matplotlib. The layout consists of a combination of vertical and grid-based components to accommodate various charts in a clean and organized way.

At the top of the dashboard, a pie chart displays the proportion of trips categorized as Completed, Delayed, or Cancelled, offering a quick view of overall service reliability. Next to it, a line plot visualizes average driver ratings, allowing the administrator to identify consistently high- or low-performing drivers based on user feedback. Additional bar charts illustrate booking patterns across different hours of the day and days of the week, highlighting peak travel times and potential capacity issues. A donut chart compares the number of passengers across different shuttle routes (Blue, Yellow, Orange, Green), helping administrators identify the most utilized routes. Other panels compare shuttle usage between faculty and students and rank drivers based on the number of trips they have completed.

These insights help administrators monitor performance, allocate resources more effectively, and improve the overall quality of service. For instance, if the dashboard shows a significant number of delayed trips on a particular route, the admin can further investigate route congestion or driver performance. Similarly, if the booking trends chart reveals underutilization during specific hours, schedules can be adjusted accordingly.

Overall, the dashboard serves as a centralized and data-driven tool that enables quick diagnostics and informed decision-making for Spartan Ride operations.

7. Working of the Analytical Module

Operational data is extracted from the live database and loaded into an analytical data warehouse using scheduled ETL pipelines. The data is transformed to match the STAR schema format, with a central fact table (e.g., trips or bookings) and supporting dimension tables (e.g., time, route, user). This structure allows for flexible slicing and dicing of the data. For instance, administrators can analyze total rides by hour, week, or region, and correlate feedback trends with specific drivers or time windows. These metrics are used for service optimization and planning.

8. Specifications and Usability of the Analytical Module

The analytical dashboard presents multiple key performance indicators (KPIs) for system administrators. These include:

- Trips by Status: Proportions of Completed, Delayed, and Cancelled rides
- Driver Ratings: Average ratings across drivers
- Route Popularity: Number of passengers per route
- Temporal Trends: Bookings by hour, day, and week
- User Segmentation: Ride preference breakdown by students vs. faculty
  These metrics are visualized using pie charts, line plots, bar charts, and donut charts. Admins can filter the views by time or role to support informed decision-making.

9. Use Cases

Please use the below login information when you test. Note that you have to select the ‘right’ role when you sign-in.

Role	ID	PW
Student	S065674646	Welcome@259
Faculty	F834567890	Welcome@441
Driver	D199719242	Welcome@460
Admin	A018236023	Welcome@999

9.1 Student and Faculty Use Cases

Students and faculty members use the platform primarily to book campus shuttle rides and manage their ride-related information. Upon logging in, they are directed to their role-specific dashboard, which provides access to features such as Book Ride, Ride History, Wallet, Profile, and Feedback.

- **Booking a Ride:** A student planning to attend a class at Cisco Campus at 5:00 PM opens the **Book Ride** page, selects the Blue Route, picks "Cisco Campus" as the stop, and chooses 4:42 PM as the departure time. Upon clicking “Book Ride,” the system deducts the fare from their wallet and confirms the reservation.

- **Reviewing Past Trips:** After encountering a delay, the student accesses the Ride History page to confirm the exact departure time and ride ID, which can later be referenced in feedback.

- **Submitting Feedback:** After a smooth ride, the student navigates to the Feedback page, selects a five-star rating, and enters a brief comment to recognize the driver’s good performance.

- **Wallet Management:** Before booking another ride, the user checks the Wallet page to ensure sufficient balance. Payment method and promo code information are also reviewed.

- **Viewing Profile:** The student accesses the Profile page to verify that their contact information and department are up to date.

These workflows are identical for faculty members, who interact with the same UI and database fields.

9.2 Driver Use Cases

Drivers use the system to manage their routes, monitor schedules, and track their work history and feedback.

- **Viewing Assigned Routes:** At the start of the day, a driver opens the My Routes page, selects today's date, and views a schedule that includes stop names, estimated times, and passenger reservations for each stop.

- **Tracking Work History:** At the end of the week, the driver visits the Trip Logs page to review their daily working hours and estimated salary. If discrepancies are noticed, they can report them to the admin.

- **Reviewing Feedback:** The driver opens the Notifications page to read comments submitted by recent passengers, using them to reflect on their service quality.

- **Managing Personal Info:** On the Profile page, the driver updates a recently changed phone number and saves the new data to the database.

These features help drivers stay organized, respond to rider feedback, and maintain up-to-date records.

9.3 Administrator Use Cases

Administrators use the system to manage user data, monitor system usage, and make informed operational decisions.

- **Managing User Records:** When a student requests an update to their emergency contact, the admin opens the Student Information page, finds the record, edits the relevant field, and clicks Save Changes to commit it.

- **Editing Driver Data:** After receiving an HR update, the admin opens the Driver Information page, updates a driver's salary, and deletes the record of a driver who no longer works for the university.

- **Viewing System Metrics:** To prepare a monthly report, the admin accesses the **Dashboard**, examines booking trends by day and hour, and reviews the pie chart summarizing trip completion rates.

- **Analyzing Route Popularity:** The admin compares shuttle usage across the Blue, Yellow, Orange, and Green routes using the donut chart in the Dashboard, identifying which routes require capacity adjustments.

These actions ensure the platform runs efficiently, stays up to date, and evolves based on data-driven insights.

10. Technical Aspects

The Spartan Ride System was developed using Python and PyQt for the front-end, MySQL for data storage, and Matplotlib and Seaborn for visualization. The UI design was prototyped using Qt Designer. SQL queries were used for database interaction, and the system relies on structured schema design principles with strict normalization. Data was simulated using Mockaroo and GPT-assisted generation. ETL processes and dashboards were tested in Jupyter Notebooks to ensure analytical reliability. GitHub was used for version control and team collaboration.

S.No.	Folder	File Name	Type	Page Directly Linked to
1	(root)	README.md	Other	
2	(root)	Sign_Up_Dialog.py	Python	
3	(root)	Spartan-Ride-SJSU.pptx	Other	
4	.idea	.gitignore	Other	
5	SourceCode	Admin_dashboard.py	Python	Admin Dashboard Page
6	SourceCode	Admin_drivery.py	Python	Admin - Driver Info Page
7	SourceCode	Admin_facultyy.py	Python	Admin - Faculty Info Page
8	SourceCode	Admin_student.py	Python	Admin - Student Info Page
9	SourceCode	Admin_window.py	Python	Admin Main Window
10	SourceCode	Feedback_Dialog.py	Python	Feedback Page
11	SourceCode	Login_Page.py	Python	Login/Main Window
12	SourceCode	Profile_DialogBox.py	Python	Profile Page
13	SourceCode	Sign_up_Dialog.py	Python	
14	SourceCode	Student_Dialog.py	Python	
15	SourceCode	Student_Dialog_RideHistory.py	Python	Ride History Page
16	SourceCode	Student_Dialog_Route.py	Python	Book Ride / Route Page
17	SourceCode	Student_Dialog_Wallet.py	Python	Wallet Page
18	SourceCode	__init__.py	Python	
19	SourceCode	backend_driver_ui.py	Python	
20	SourceCode	data201.py	Python	
21	SourceCode	db_util.py	Python	
22	SourceCode	main.py	Python	
23	SourceCode	ui_helper.py	Python	
24	UI_Files	Admin_Dialog.ui	Qt5 UI File	
25	UI_Files	Driver_Image.png	Other	
26	UI_Files	Driver_MyLog.ui	Qt5 UI File	Trip Log Page
27	UI_Files	Driver_MyProfile.ui	Qt5 UI File	Profile Page
28	UI_Files	Driver_MyRoutes.ui	Qt5 UI File	Book Ride / Route Page

29	UI_Files	Driver_Notification.ui	Qt5 UI File	Driver Notifications Page
30	UI_Files	Feedback_Dialog.ui	Qt5 UI File	Feedback Page
31	UI_Files	Login_Page.ui	Qt5 UI File	Login/Main Window
32	UI_Files	Profile_DialogBox.ui	Qt5 UI File	Profile Page
33	UI_Files	Sign_Up_Dialog.ui	Qt5 UI File	
34	UI_Files	Student_Dialog.ui	Qt5 UI File	
35	UI_Files	Student_Dialog_RideHistory.ui	Qt5 UI File	Ride History Page
36	UI_Files	Student_Dialog_Wallet.ui	Qt5 UI File	Wallet Page
37	UI_Files	Student_RouteDetails.ui	Qt5 UI File	Book Ride / Route Page
38	UI_Files	admin.ui	Qt5 UI File	
39	UI_Files	baskbashers.ini	Config File	Database Config
40	UI_Files	feedback.ui	Qt5 UI File	Feedback Page
41	UI_Files	qt_temp.K51742	Other	
42	UI_Python	Driver_MyLog.py	Python	Trip Log Page
43	UI_Python	Driver_MyProfile.py	Python	Profile Page
44	UI_Python	Driver_MyProfile_ui.py	Python	Profile Page
45	UI_Python	Driver_MyRoutes.py	Python	Book Ride / Route Page
46	UI_Python	Driver_Notification.py	Python	Driver Notifications Page
47	UI_Python	Login_Page_ui.py	Python	Login/Main Window
48	UI_Python	Profile_DialogBox_ui.py	Python	Profile Page
49	UI_Python	Sign_Up_Dialog_ui.py	Python	
50	UI_Python	Student_Dialog_RideHistory_ui.py	Python	Ride History Page
51	UI_Python	Student_Dialog_Wallet_ui.py	Python	Wallet Page
52	UI_Python	Student_Dialog_py.py	Python	
53	UI_Python	Student_Dialog_ui.py	Python	
54	UI_Python	Student_RouteDetails_ui.py	Python	Book Ride / Route Page
55	UI_Python	__init__.py	Python	
56	UI_Python	admin.py	Python	
57	UI_Python	db_util.py	Python	
58	config	config.ini	Config File	Database Config
59	config	config_wh.ini	Config File	Database Config
60	data	.DS_Store	Other	
61	data	baskbashers_db.sql	SQL Dump	Database Dump
62	data	booking.csv	Other	
63	data	driver.csv	Other	
64	data	faculty.csv	Other	
65	data	feedback.csv	Other	Feedback Page
66	data	login.csv	Other	Login/Main Window
67	data	route.csv	Other	Book Ride / Route Page
68	data	shuttle.csv	Other	
69	data	student.csv	Other	
70	data	trip_history.csv	Other	

71	data	wallet.csv	Other	Wallet Page
72	Images	ChatGPT Image Apr 21, 2025, 10_35_04 PM.png	Other	
73	Images	ChatGPT Image Apr 21, 2025, 12_04_51 AM.png	Other	
74	Images	ChatGPT Image Apr 21, 2025, 12_09_12 AM.png	Other	
75	Images	ChatGPT Image May 8, 2025, 09_34_18 PM.png	Other	
76	Images	ChatGPT Image May 8, 2025, 09_34_26 PM.png	Other	
77	Images	Driver_Image.png	Other	
78	Images	SpartanLogo.png	Other	
79	Images	SpartanLogo_small.png	Other	
80	Images	abc.png	Other	
81	Images	calender.png	Other	
82	Images	city.png	Other	
83	Images	fd.png	Other	
84	Images	icons8-star-48.png	Other	
85	Images	im.png	Other	
86	Images	images.png	Other	
87	Images	is.png	Other	
88	Images	mail.png	Other	
89	Images	male.png	Other	
90	Images	maxresdefault.jpg	Other	
91	Images	person.png	Other	
92	Images	photo.png	Other	
93	Images	picture.png	Other	
94	Images	spartan.png	Other	
95	Images	spartan_logo.png.jpeg	Other	
96	Images	spartan_singup.png	Other	
97	Images	spartanlogo.jpeg	Other	
98	Images	star.png	Other	
99	Images	student_dialog.png	Other	
100	Images	student_imager.png	Other	
101	Images	Bay.png	Other	
102	Images	Screenshot 2025-05-05 at 10.35.30,ÄØPM.png	Other	
103	Images	blue route.png	Other	
104	Images	green route.png	Other	
105	Images	orange route.png	Other	
106	Images	yellow route.png	Other	

Useful Links:

- Github: https://github.com/sakotiya/SPARTANRIDESJSU

How to Operate the App:

1. Connect to the Remote Server
2. User terminal to go to project folder
3. Run following command

    ````SPARTANRIDESJSU % python -m SourceCode.Login_Page````

- Login Information

Role	ID	PW
Student	S065674646	Welcome@259
Faculty	F834567890	Welcome@441
Driver	D199719242	Welcome@460
Admin	A018236023	Welcome@999
    ````