Module Guide for Software Engineering

Team #16, The Chill Guys
Hamzah Rawasia
Sarim Zia
Aidan Froggatt
Swesan Pathmanathan
Burhanuddin Kharodawala

January 20, 2026

1 Revision History

Date	Developer	Notes
November 1st	Aidan Froggatt	Completed Section 11
November 12th	Hamzah Rawasia	Completed Sections 6 and 8
November 12th	Swesan Pathmanathan	Completed Sections 5 and 7
November 13th	Burhan Kharodawala	Completed Sections 3, 4, and 10
November 13th	Sarim Zia	Completed Sections 9, 10, and 12

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	**description**
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
Software Engineering	Explanation of program name
UC	Unlikely Change

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (?). We advocate a decomposition based on the principle of information hiding (?). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by ?, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (?). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section **??** lists the anticipated and unlikely changes of the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** gives a detailed description of the modules. Section **??** includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section **??** describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section **??**, and unlikely changes are listed in Section **??**.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Matching Algorithm: The matching algorithm focuses on the goal of finding the best available drivers for the riders. This algorithm takes into account the user's location, timetable, time, and distance. For instance, it matches a rider with a driver that has the same timetable and lives in the same area. The current algorithm will be simply based on these four parameters, and it will have to change in the future to consider more parameters to increase its efficiency and accuracy. Moreover, subtle changes to the algorithm may also be expected for further optimization of its complexity and speed.

AC2: User Interface: As the application proceeds to go through its initial phase of development and use, usability testing is expected to be done to ensure user satisfaction. As the users provide feedback, changes will be made to the user interface in the future phases of development. Moreover, even after that, changes will be made based on accessibility requirements in the future. The layout and navigation are the two main things from the user interaction that would need the most changes as per user's feedback.

AC3: Database Schema: As the app updates with new features, the current method of analytics may change. New parameters would be considered for the matching algorithm, and new features may be added based on future user needs. Therefore, changes may need to be made to the database schemas

AC4: Payment Processing: Additional payment processors and pricing formulas will be used to determine the price estimate for drivers. An addition of parameters to estimate the cost would cause the algorithms to change.

AC5: Development Environment: Changes could be made to the CI/CD configuration to improve the reliability and stability of the system.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Devices: Hitchly is an application that is developed for mobile devices only. A version that is compatible to run on a desktop is an unlikely change as it requires major changes to the software design.

UC2: Language Support: The application is primarily based on the English language. Adding additional languages to the interface would require a major design decision.

UC3: Programming Language and Technologies: The core of this application is supposed to be developed using React Native, Typescript, and SQL. Making changes to how the application is developed is an unlikely change as it requires a change in the entire design of the application.

UC4: Verification Requirement: This app is strictly restricted for McMaster affiliated users as the entire concept of this application is designed for them. It is not subject to removal. Verification of the users will be their McMaster affiliated email.

5 Module Hierarchy

This section provides an overview of the modular design for Hitchly. The hierarchy follows the principle of information hiding, where each module encapsulates a design decision that may change independently. Only the leaf modules shown in Table **??** are implemented.

5.1 Hardware-Hiding Modules There are no hardware-hiding modules in this design. Hitchly relies on high-level framework APIs (Expo, React Native), so no direct hardware abstraction is required.

5.2 Behaviour-Hiding Modules These modules implement the user-visible behaviour of the application. They hide workflow logic, API interactions, and data handling that support Hitchly's core features.

- M1: Authentication & Verification Module
- M2: User Profile Module
- M3: Route & Trip Module
- M4: Matching Module
- M5: Scheduling Module
- M6: Notification Module
- M7: Rating & Feedback Module
- M8: Safety & Reporting Module

- M9: Payment & Cost Estimation Module
- M10: Admin & Moderation Module

5.3 Software-Decision Modules These modules encapsulate internal logic, algorithms, and implementation decisions not visible to users.

- M11: Pricing Module
- M12: Database Module

Level 1	**Level 2**
Hardware-Hiding Module	None
Behaviour-Hiding Module	M1: Authentication & Verification Module M2: User Profile Module M3: Route & Trip Module M4: Matching Module M5: Scheduling Module M6: Notification Module M7: Rating & Feedback Module M8: Safety & Reporting Module M9: Payment & Cost Estimation Module M10: Admin & Moderation Module
Software-Decision Module	M11: Pricing Module M12: Database Module

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table **??**.

7 Module Decomposition

Modules are decomposed according to the principle of information hiding (**?**). Each module's *Secrets* describes the internal design decision that is intentionally hidden from other modules, while the *Services* field specifies *what* the module provides without revealing *how* the service is implemented. The *Frontend UI*, *API Logic*, and *Database Models* indicate how each module is realized across the system's architecture. *Implemented By* identifies the technologies used, and *Type of Module* classifies each module as a Behaviour-Hiding or Software-Decision module according to Section **??**. Only the leaf modules are included.

Behaviour-Hiding Modules

M1: Authentication & Verification Module

- **Secrets:** Token/session strategy, email-verification workflow, and credential-validation logic.
- **Services:** Allows users to register, log in, and verify McMaster email accounts.
- **Frontend UI:** Login screens, signup forms, verification code UI.
- **API Logic:** `authRouter` (login, register, verify), session helpers, Zod validation.
- **Database Models:** `User` table (email, password hash, verified flag, session tokens).
- **Implemented By:** React Native (Expo) + tRPC backend with Prisma ORM.
- **Type of Module:** Behaviour-Hiding Module.

M2: User Profile Module

- **Secrets:** Role-assignment logic and profile-preference storage design.
- **Services:** Provides profile editing, preferences, and vehicle data management.
- **Frontend UI:** Profile page, edit forms, vehicle info inputs.
- **API Logic:** `userRouter` for CRUD operations, preference update flow.
- **Database Models:** `User`, `Vehicle`, `Preference`.
- **Implemented By:** React Native UI + tRPC + Prisma.
- **Type of Module:** Behaviour-Hiding Module.

M3: Route & Trip Module

- **Secrets:** Trip-storage schema, route normalization, and time formatting.
- **Services:** Allows users to create, list, and cancel trips.
- **Frontend UI:** Trip creation form, trip list UI, route selection screen.

- **API Logic:** `tripRouter` for trip creation, querying, and deletion.
- **Database Models:** `Trip` table (origin, destination, time, seat count).
- **Implemented By:** React Native + tRPC backend + Prisma.
- **Type of Module:** Behaviour-Hiding Module.

M4: Matching Module

- **Secrets:** Scoring algorithm, distance/time weighting, and Strategy Pattern implementation.
- **Services:** Computes driver–rider compatibility and returns ordered match results.
- **Frontend UI:** Swipe-based card UI, match results list.
- **API Logic:** `matchmakingRouter`, MatchEngine (Strategy Pattern).
- **Database Models:** `Match`, `Swipe`.
- **Implemented By:** Node.js backend (tRPC) + algorithm utilities + Prisma ORM.
- **Type of Module:** Behaviour-Hiding Module.

M5: Scheduling Module

- **Secrets:** Recurring-trip generation algorithm and time-window parsing.
- **Services:** Creates recurring or one-time schedules linked to trips.
- **Frontend UI:** Time picker, recurring toggle, calendar UI.
- **API Logic:** Schedule parser and recurring schedule generator.
- **Database Models:** `Schedule` table.
- **Implemented By:** tRPC backend + Prisma.
- **Type of Module:** Behaviour-Hiding Module.

M6: Notification Module

- **Secrets:** Push-token handling and asynchronous event queue logic.
- **Services:** Sends push notifications for matches, cancellations, and reminders.
- **Frontend UI:** In-app notification center, Expo push integration.
- **API Logic:** Notification service with event emitters.
- **Database Models:** `Notification` table.
- **Implemented By:** Expo Push Service + tRPC backend dispatcher.
- **Type of Module:** Behaviour-Hiding Module.

M7: Rating & Feedback Module

- **Secrets:** Reputation computation and weighting logic.
- **Services:** Enables users to submit ratings and text feedback after trips.
- **Frontend UI:** Post-ride rating screen.
- **API Logic:** `ratingRouter`.
- **Database Models:** `Rating` table (linked to `Match`).
- **Implemented By:** React Native + tRPC + Prisma ORM.
- **Type of Module:** Behaviour-Hiding Module.

M8: Safety & Reporting Module

- **Secrets:** Incident-flagging thresholds and safety-response workflow.
- **Services:** Allows reporting of unsafe behaviour and stores incidents for admin review.
- **Frontend UI:** Report button, safety resources screen.
- **API Logic:** Report submission via `reportRouter`.
- **Database Models:** `Report`.
- **Implemented By:** React Native + tRPC + Prisma.
- **Type of Module:** Behaviour-Hiding Module.

M9: Payment & Cost Estimation Module

- **Secrets:** Fare-calculation logic and mock-payment validation.
- **Services:** Estimates ride cost and records payment confirmations.
- **Frontend UI:** Fare display, confirmation screen.
- **API Logic:** `paymentRouter`.
- **Database Models:** `Payment`, `Transaction`.
- **Implemented By:** tRPC backend + Prisma ORM.
- **Type of Module:** Behaviour-Hiding Module.

M10: Admin & Moderation Module

- **Secrets:** Moderation rules, flagging thresholds, and admin-only access logic.
- **Services:** Allows admins to review reports, ban users, and view platform stats.
- **Frontend UI:** Admin dashboard (optional).
- **API Logic:** `adminRouter`.
- **Database Models:** Admin logs, flagged users.
- **Implemented By:** tRPC backend + Prisma.
- **Type of Module:** Behaviour-Hiding Module.

Software-Decision Modules

M11: Pricing Module

- **Secrets:** Fuel-rate constants, distance multipliers, and fare model.
- **Services:** Computes estimated trip price using distance and conditions.
- **Frontend UI:** Fare estimate displayed in trip preview.
- **API Logic:** Cost-calculation service reused by M9.
- **Database Models:** Pricing configuration constants.
- **Implemented By:** Node.js backend utilities + shared pricing helpers.
- **Type of Module:** Software-Decision Module.

M12: Database Module

- **Secrets:** ORM mapping, schema decisions, and migration strategy.
- **Services:** Provides access to all persistent data models via a centralized DB client.
- **Frontend UI:** None.
- **API Logic:** Prisma client configuration and model exports.
- **Database Models:** All tables (User, Trip, Match, Schedule, etc.).
- **Implemented By:** Prisma ORM + PostgreSQL.
- **Type of Module:** Software-Decision Module.

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
FR211	M4, M3, M5, M10
FR212	M4, M5, M10
FR213	M4, M2
FR214	M4, M2, M5
NFR211	M4, M10
NFR212	M4, M1, M10
FR221	M5, M3, M10
FR222	M5, M6
FR223	M5, M6, M9
FR224	M6, M5
NFR221	M10, M5
NFR222	M6, M5, M10
FR231	M1, M2
FR232	M1, M10
FR233	M8, M3, M6
FR234	M8, M2, M12
NFR231	M1, M10
NFR232	M1, M10, M12
FR241	M11, M5, M3
FR242	M9
FR243	M9
FR244	M10, M9
NFR241	M9
NFR242	M9
FR251	M2, M1
FR252	M10, M2
FR253	M10, M2, M3
FR254	M2, M4
NFR251	M2, M1
NFR252	M2, M10
FR261	M7, M2
FR262	M8, M12
FR263	M7, M5, M4
FR264	M7, M2
NFR261	M7, M12

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC??	M4, M2, M5
AC??	M1, M2, M3, M4, M5, M6, M7, M8, M9, M11, M12
AC??	M10
AC??	M9, M11
AC??	N/A (External Process)

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

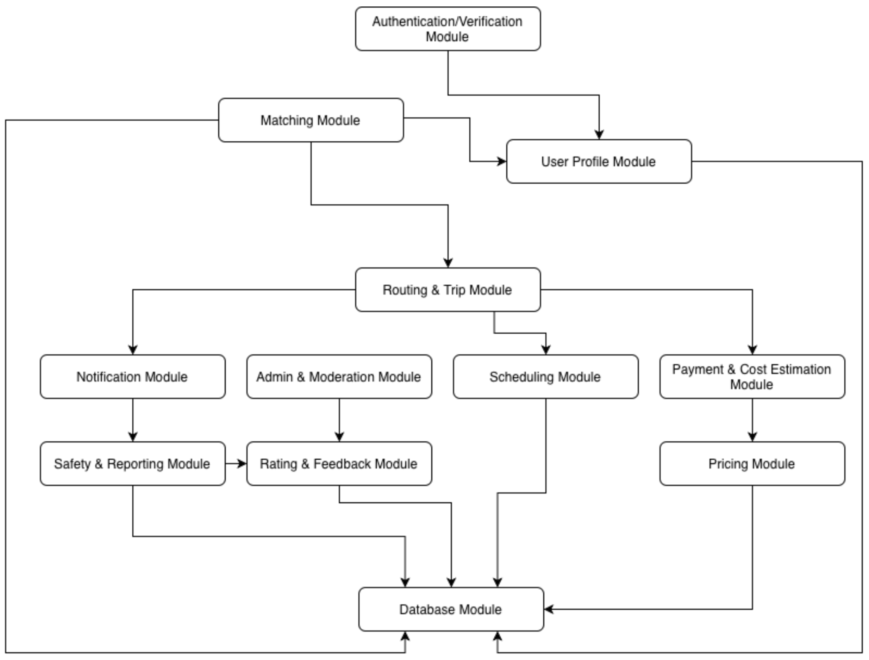


Figure 1: Use hierarchy among modules

10 User Interfaces

This section provides an overview of the important screens of the application.

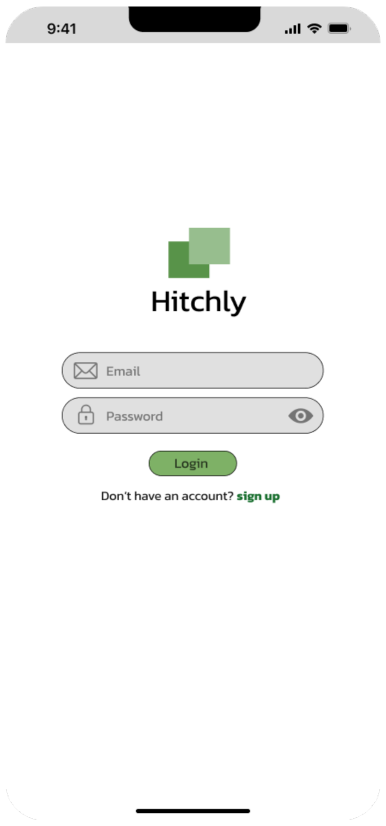


Figure 2: Main Login Screen

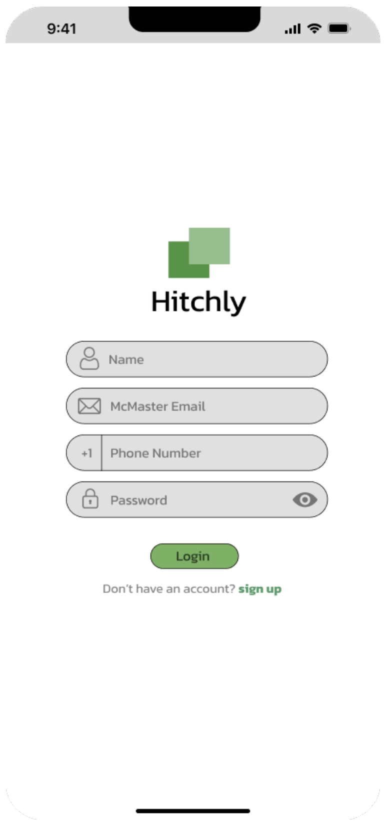
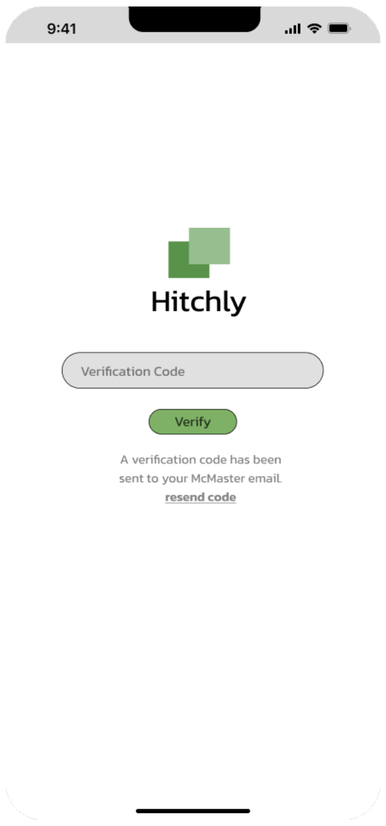


Figure 3: Register Screen



A verification code has been sent to your McMaster email.

Figure 4: Verification Screen

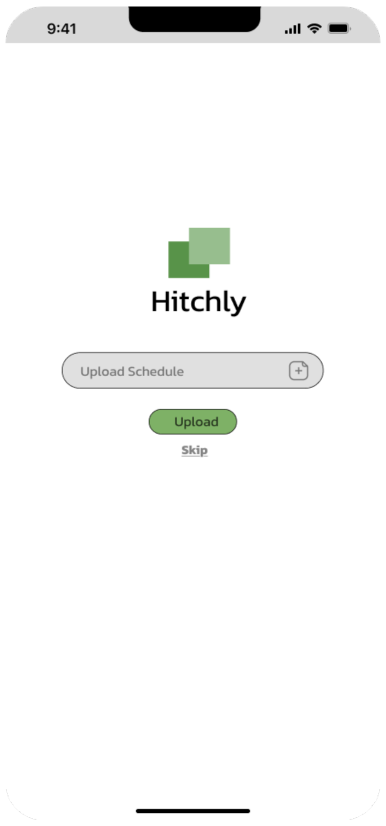


Figure 5: Upload Schedule Screen

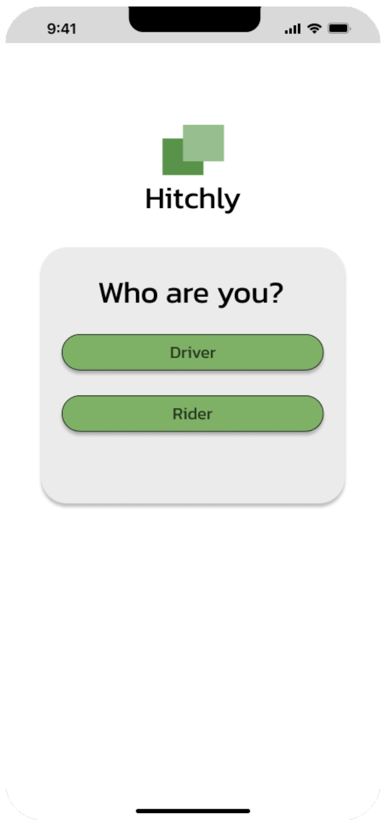


Figure 6: Select User Type Screen

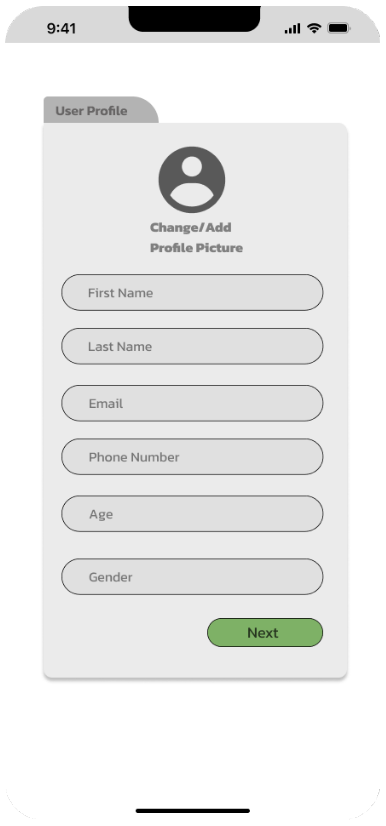


Figure 7: User Prompted to Add Profile Information

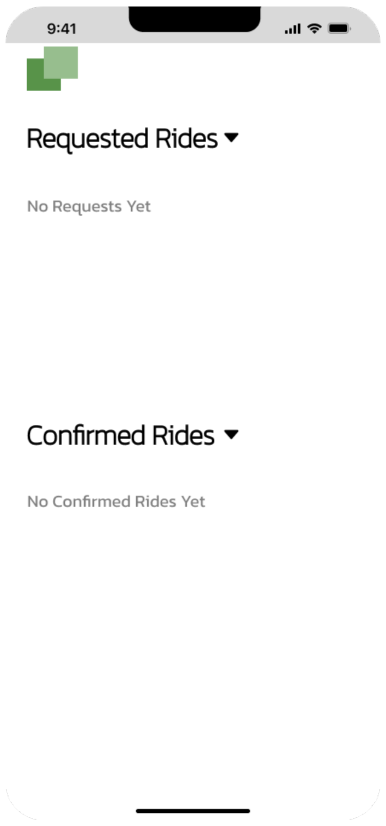


Figure 8: Home Screen

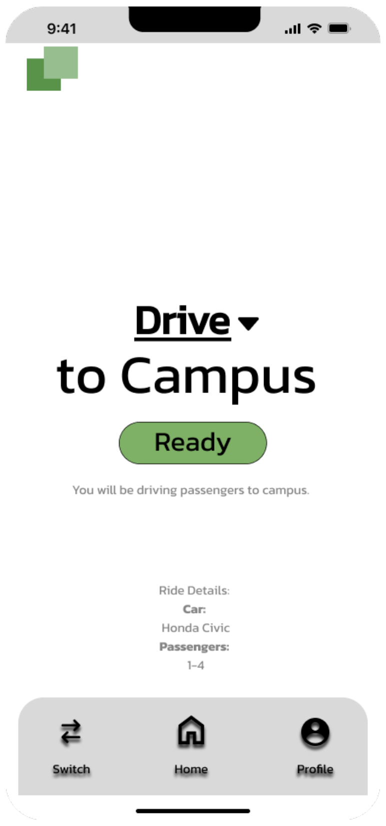
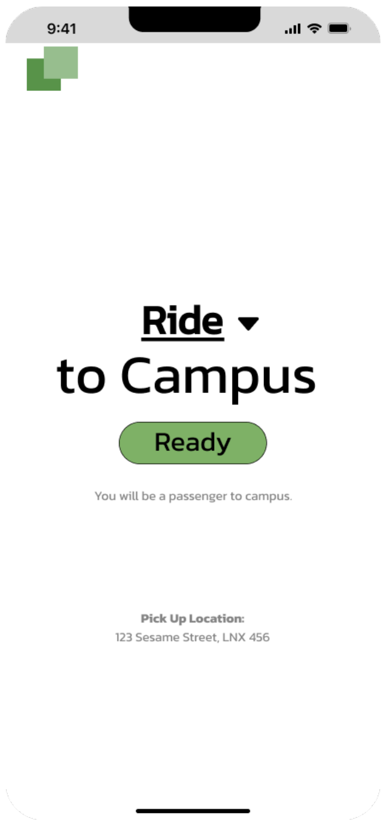


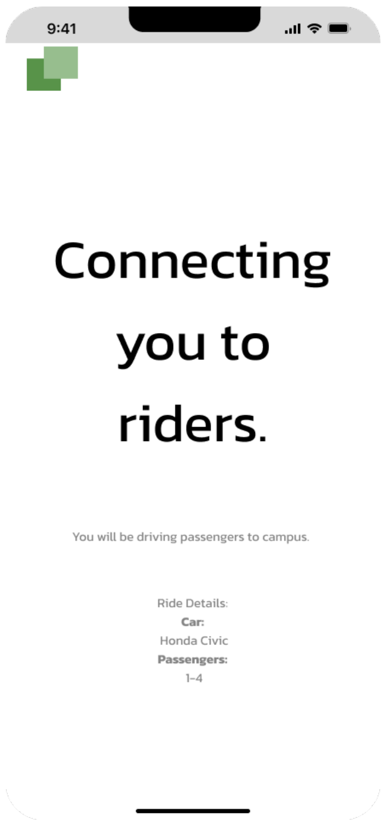
Figure 9: Switch to Driver Screen



You will be a passenger to campus.

Pick Up Location:
123 Sesame Street, LNX 456

Figure 10: Switch to Rider Screen

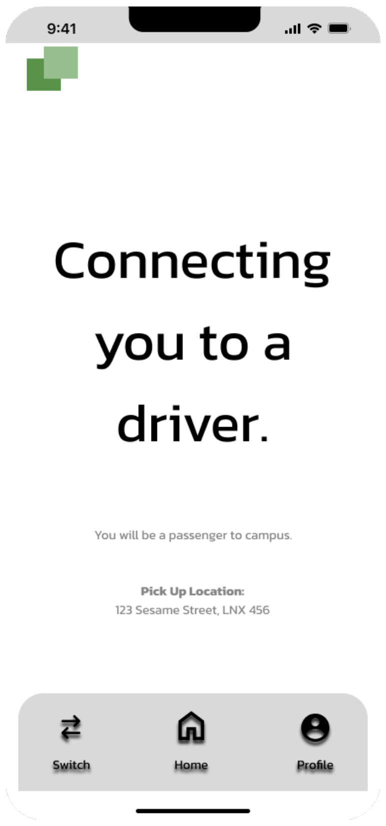


Connecting you to riders.

You will be driving passengers to campus.

Ride Details:
Car:
Honda Civic
Passengers:
1-4

Figure 11: Driver Connection Screen



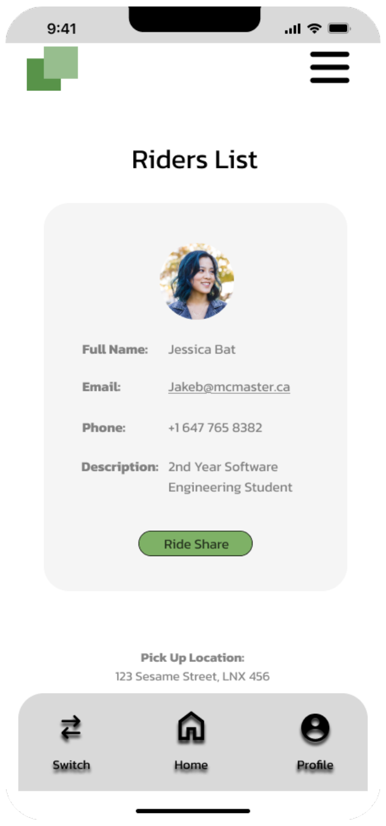
9:41

Connecting you to a driver.

You will be a passenger to campus.

Pick Up Location:
123 Sesame Street, LNX 456

Figure 12: Rider Connection Screen



Riders List

Full Name: Jessica Bat

Email: Jakeb@mcmaster.ca

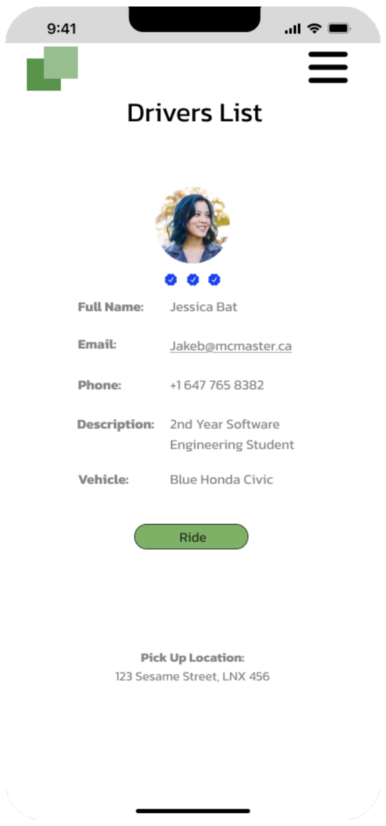
Phone: +1 647 765 8382

Description: 2nd Year Software Engineering Student

Ride Share

Pick Up Location:
123 Sesame Street, LNX 456

Figure 13: Matched Rider List Screen



9:41

Drivers List

Full Name:	Jessica Bat
Email:	Jakeb@mcmaster.ca
Phone:	+1 647 765 8382
Description:	2nd Year Software Engineering Student
Vehicle:	Blue Honda Civic

Pick Up Location:
123 Sesame Street, LNX 456

Figure 14: Matched Driver List Screen

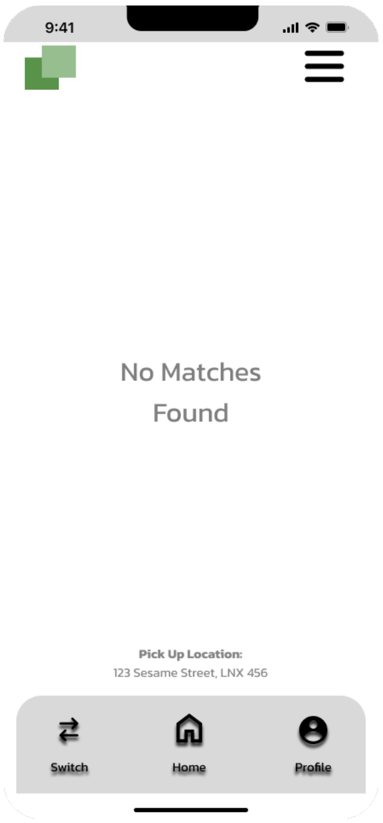


Figure 15: No Matches Found Screen

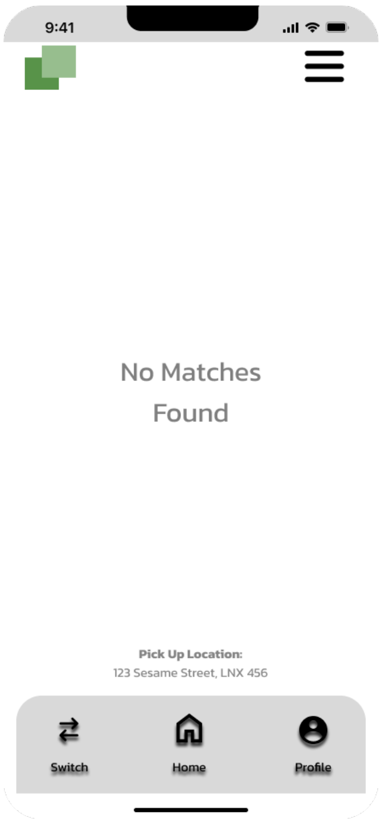


Figure 16: No Matches Found Screen

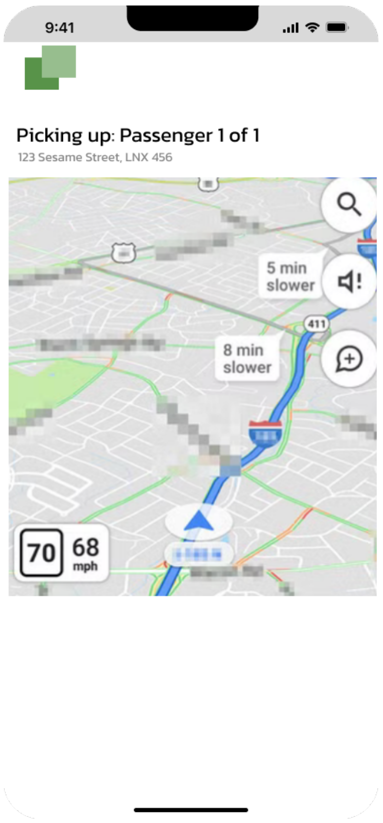
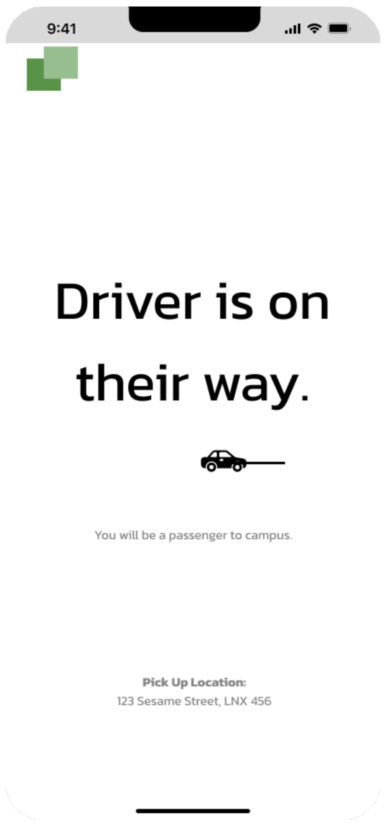


Figure 17: Driver Navigation Screen

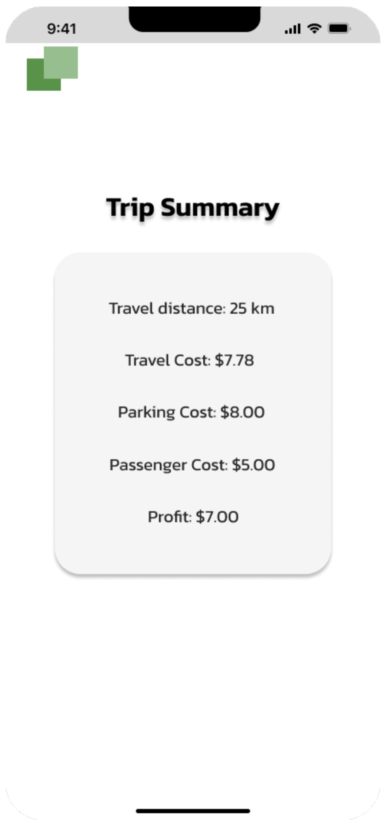

9:41

Driver is on their way.

You will be a passenger to campus.

Pick Up Location:
123 Sesame Street, LNX 456

Figure 18: Rider Wait Screen



Travel distance: 25 km

Travel Cost: $7.78

Parking Cost: $8.00

Passenger Cost: $5.00

Profit: $7.00

Figure 19: Cost Estimation Screen

11 Design of Communication Protocols

This section describes the design of the communication protocols implemented in Hitchly. Communication occurs primarily between the client-side mobile application (developed with React Native / Expo) and the backend server (implemented with Node.js, Express, and tRPC). The system also integrates third-party APIs for geolocation and notifications.

11.1 11.1 Overall Architecture

All communication between the mobile client and the backend follows a request–response model over HTTPS, ensuring confidentiality and integrity. Data is serialized in JSON format and transferred through REST and type-safe tRPC endpoints. The architecture adopts a modular service-based design to support scalability and maintainability:

- **Frontend (Client):** React Native app that sends authenticated requests to backend endpoints via HTTPS using the tRPC client.
- **Backend (Server):** Express server exposing routes through the tRPC API layer. It validates and processes incoming requests, interacts with the PostgreSQL database

through Drizzle ORM, and returns structured JSON responses.

- **External Services:** Mapping and geolocation (Google Maps / Mapbox API), Expo Push Notification Service for alerts, and optional OAuth verification via McMaster SSO (for email validation).

11.2 11.2 Communication Flow

A typical data exchange involves the following sequence:

1. The user performs an action in the mobile application (e.g., requests a ride, updates schedule, or rates a driver).
2. The frontend constructs a tRPC call encapsulating the action data and attaches the user's JWT authentication token in the header.
3. The server authenticates the request, validates input schemas, and executes the corresponding service logic.
4. The backend queries or updates PostgreSQL through Drizzle ORM and returns a success or error payload to the client.
5. The client updates its UI state using React Query based on the received data.

11.3 11.3 API Endpoints and Message Types

Hitchly's backend exposes the following logical API groups:

- **Auth API:** Handles user registration, login, McMaster email verification, and token refresh. Exchanges encrypted credentials and JWT tokens.
- **User API:** Manages user profiles, schedules, and preferences.
- **Matching API:** Accepts commute data, performs matching algorithm, and returns ranked ride offers.
- **Trip API:** Records completed trips, generates summaries, and calculates cost-sharing.
- **Rating API:** Sends and retrieves ratings and reviews.
- **Notification API:** Uses Expo Push Notification service to deliver trip confirmations, cancellations, and updates in real time.

All messages follow the JSON structure:

```
{
  "status": "success",
  "data": { ... },
  "timestamp": "2025-01-03T12:00:00Z"
}
```

11.4 11.4 Security and Reliability

Security and reliability are ensured through:

- **Transport Security:** HTTPS with TLS 1.3 and HSTS enforcement.
- **Authentication:** JWT Bearer tokens included in headers of all authorized requests.
- **Input Validation:** All incoming payloads validated using Zod schemas in tRPC to prevent injection or malformed data.
- **Error Handling:** Standardized error codes with human-readable messages returned to the client.
- **Rate Limiting:** Express middleware limits excessive requests to prevent abuse.
- **Retry Mechanism:** Client-side React Query retries transient network failures automatically.

11.5 11.5 Design Considerations

The communication design of Hitchly emphasizes the following principles:

- **Consistency:** Shared TypeScript types across frontend and backend guarantee end-to-end type safety through tRPC.
- **Extensibility:** Additional endpoints (e.g., payment API or advanced analytics) can be introduced without modifying existing client modules.
- **Scalability:** The stateless REST/tRPC design supports horizontal scaling of the Express server behind a load balancer.
- **Low Latency:** JSON over HTTPS is lightweight; caching of common responses and asynchronous notification delivery minimize perceived delay.

This design ensures secure, efficient, and maintainable communication between Hitchly's mobile application, backend, and external services, supporting its goals of safety, reliability, and sustainability for McMaster commuters.

12 Timeline

The following timeline details our implementation for Rev 0. Each task is broken down to the module level. Each tasks has roles and responsibilities defined along with estimated development duration.

Table 4: Phase 1

Task	**Module(s)**	**Status**	**Responsible**
Creating Hitchly Wireframes for UI	M2	Done	Burhan, Sarim
Make initial version of matching algorithm	M4	Done	Hamzah, Sarim
Set up authentication and application bones	M1	Done	Aidan

Table 5: Phase 2: Front End and User Roles

Task	**Module(s)**	**Status**	**Responsible**
Develop login, registration, and session flows	M1	Weeks 14–16	Aidan
Build user profile screens and preferences interface	M2	Weeks 14–16	Swesan
Create upload, settings, and account management pages	M2	Weeks 14–16	Burhan
Build navigation, layouts, and key frontend components	M2	Weeks 14–16	Hamzah
Integrate basic user-to-database connections for profile initialization	M12	Weeks 15–17	Sarim

Table 6: Phase 3: Core Ride-Share Logic and Trip Flow

Task	**Module(s)**	**Status**	**Responsible**
Implement matching algorithm and rider–driver compatibility logic	M4	Weeks 18–21	Sarim, Hamzah
Build Routing & Trip creation (origin, destination, seat count)	M3	Weeks 18–21	Burhan
Implement Scheduling module (recurring rides, time windows)	M5	Weeks 19–22	Aidan
Integrate Payment & Cost estimation (distance, time, pricing model)	M9	Weeks 20–22	Swesan
Build Pricing module (fare calculations, dynamic pricing rules)	M11	Weeks 20–23	Swesan
Add Rating & Feedback system for both riders and drivers	M7	Weeks 21–23	Sarim
Implement Admin & Moderation tools (flagging, dispute handling)	M10	Weeks 22–24	Burhan, Aidan
Build Notification module (push/in-app updates for trips)	M6	Weeks 21–24	Hamzah

Table 7: Phase 4: Safety, Database Integration, and Finalization

Task	**Module(s)**	**Status**	**Responsible**
Implement Safety & Reporting module (incident reports, flagging)	M8	Weeks 24–26	Burhan
Complete full Database Module integration across all features	M12	Weeks 24–27	Aidan, Sarim
Perform end-to-end testing of all ride flows	M1–M12	Weeks 25–28	All
Optimize performance, resolve bugs, and finalize UI polish	M1–M12	Weeks 26–29	All
Prepare release build, documentation, and deployment setup	M1–M12	Weeks 28–30	All

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.