# Team Contributions: Rev 0
# Software Engineering

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

## 1 Demo Plans

The Revision 0 demonstration will showcase a comprehensive, production-ready version of the Hitchly application with enhanced features beyond the Proof of Concept. Building upon the POC's foundation of McMaster email authentication and basic ride matching, the Rev 0 demo will demonstrate a complete end-to-end user experience with expanded functionality.

### 1.1 Core Features to Demonstrate

- **Complete User Authentication & Verification**
  - McMaster email verification during sign-up
  - Secure login and session management
- **Comprehensive Profile System**
  - User profile creation and management (personal information, preferences)
  - Driver vehicle registration and details
  - Schedule input and timetable management
  - Profile editing and updates
- **Advanced Ride Matching**
  - Multi-passenger matching based on vehicle capacity
  - Schedule and location-based matching algorithm
  - Matching filters (gender preferences, music preferences, vehicle type)
  - Real-time match notifications

- **Route & Trip Management**
  - Trip creation with origin, destination, departure time, and capacity
  - Visual route representation using mapping services
  - Trip lifecycle management (scheduled, active, completed states)
  - Trip cancellation and modification capabilities
  - Seat management and capacity tracking
- **Scheduling System**
  - Recurring schedule creation (weekly patterns, day-of-week selection)
  - Automated trip generation from recurring schedules
  - One-time schedule support
  - Schedule viewing and cancellation
  - Integration with trip creation for seamless ride planning
- **Cost Calculation & Trip Management**
  - Complex fare calculation based on gas costs, number of passengers, and parking fees
  - Per-passenger cost breakdown
  - Trip confirmation workflow
  - Ride check-in system (start and end of trip)
  - Trip summary generation with detailed cost breakdown
- **Safety & Reporting**
  - User reporting system for safety incidents
  - Safety resources and information pages
  - Report submission and tracking
  - Safety flag management
- **Admin & Moderation**
  - Admin dashboard for platform oversight
  - User management and role assignment
  - Report review and resolution workflow
  - User warning and moderation actions
  - Application analytics and statistics
- **User Experience Enhancements**
  - Polished mobile UI using NativeWind and shadcn/ui components
  - Smooth navigation between screens
  - Push notifications for matches and trip updates
  - Rating and review system for completed trips

## 1.2 Demo Workflow

The demonstration will follow a complete user journey:

1. **Driver Onboarding:** Show a driver signing up, verifying their McMaster email, registering their vehicle, and setting their schedule and cost preferences.
2. **Rider Onboarding:** Demonstrate a rider creating their profile, inputting their schedule and location preferences, and setting matching filters.
3. **Recurring Schedule Setup:** Demonstrate creating a recurring weekly schedule (e.g., Monday-Friday morning commute) that automatically generates trips, showing how the Scheduling Module integrates with trip creation.
4. **Trip Creation & Route Visualization:** Show a driver creating a trip with origin and destination, displaying the route on a map, and setting capacity. Demonstrate both one-time trips and trips generated from recurring schedules.
5. **Matching Process:** Show the system matching multiple riders with a driver based on overlapping schedules, compatible locations, and user preferences. Display the matching algorithm's results and cost estimates.
6. **Trip Confirmation:** Demonstrate the confirmation workflow where both driver and riders accept the match, leading to a confirmed trip.
7. **Trip Execution:** Show the ride check-in process at the start and end of the trip, demonstrating real-time trip tracking.
8. **Trip Completion:** Display the generated trip summary with detailed cost breakdown per passenger, and demonstrate the rating/review system.
9. **Safety & Reporting:** Demonstrate the safety reporting feature where a user can submit a report about another user or safety concern, showing the report submission workflow.
10. **Admin Dashboard (Optional):** If time permits, briefly demonstrate the admin dashboard showing user management, report review, and application analytics capabilities.

## 1.3 Technical Demonstration

The demo will also highlight the technical robustness of the system:

- Cross-platform functionality (iOS and Android)
- Responsive UI and smooth user interactions

This comprehensive demonstration will validate that Hitchly has evolved from a proof-of-concept into a functional, user-ready application that addresses the core problem of safe, affordable, and sustainable commuting for the McMaster community.

## 2 Team Meeting Attendance

| **Student** | **Meetings** |
|---|---|
| Total | 2 |
| Hamzah Rawasia | 2 |
| Sarim Zia | 2 |
| Aidan Froggatt | 2 |
| Swesan Pathmanathan | 2 |
| Burhanuddin Kharodawala | 2 |

## 3 Supervisor/Stakeholder Meeting Attendance

**McMaster Students** [fill in this information]

| **Student** | **Meetings** |
|---|---|
| Total | 2 |
| Hamzah Rawasia | 2 |
| Sarim Zia | 2 |
| Aidan Froggatt | 2 |
| Swesan Pathmanathan | 2 |
| Burhanuddin Kharodawala | 2 |

## 4 Lecture Attendance

| **Student** | **Lectures** |
|---|---|
| Total | 1 |
| Hamzah Rawasia | 1 |
| Sarim Zia | 1 |
| Aidan Froggatt | 1 |
| Swesan Pathmanathan | 1 |
| Burhanuddin Kharodawala | 1 |

# 5 TA Document Discussion Attendance

**TA's Name: Lucas Dutton**

| Student | Lectures |
|---|---|
| Total | 1 |
| Hamzah Rawasia | 1 |
| Sarim Zia | 1 |
| Aidan Froggatt | 1 |
| Swesan Pathmanathan | 1 |
| Burhanuddin Kharodawala | 0 |

Burhanuddin Kharodawala was unable to attend the TA document discussion as he was out of town that day.

# 6 Commits

| Student | Commits | Percent |
|---|---|---|
| Total | 195 | 100% |
| Swesan Pathmanathan | 78 | 40.0% |
| Aidan Froggatt | 52 | 26.7% |
| Hamzah Rawasia | 35 | 17.9% |
| Burhanuddin Kharodawala | 18 | 9.2% |
| Sarim Zia | 12 | 6.2% |

This data is for all the commits post POC and it is taken from the GitHub Repositories Insights Page. Burhanuddin and Sarim have lower number of commits post POC as they had previously worked on the UI/UX design.

# 7 Issue Tracker

| Student | Authored (O+C) | Assigned (C only) |
|---|---|---|
| Swesan Pathmanathan | 12 | 3 |
| Aidan Froggatt | 8 | 7 |
| Hamzah Rawasia | 8 | 6 |
| Burhanuddin Kharodawala | 5 | 3 |
| Sarim Zia | 3 | 2 |

The data is all of the issue opened and closed post POC, and it is taken from the GitHub Repositories Issues Page.

# 8 CICD

Hitchly employs a comprehensive Continuous Integration and Continuous Deployment (CI/CD) pipeline using GitHub Actions to ensure consistent quality, reliability, and maintainability across all project components. The pipeline is designed with intelligent path-based filtering to optimize build times by only running relevant checks when specific file types are modified.

## 8.1 Continuous Integration

The CI pipeline consists of two main workflows: `ci.yml` and `buildtex.yml`. The `ci.yml` workflow implements a change detection system that filters jobs based on whether code files or LaTeX documentation files were modified, ensuring efficient resource usage.

### 8.1.1 Code Quality Checks

For code changes (TypeScript, JavaScript, JSON, Markdown, and configuration files), the following automated checks are executed on every push and pull request:

- **Type Check:** TypeScript type checking ensures type safety across the Turborepo monorepo.
- **Lint:** ESLint runs to enforce code quality standards and catch potential errors.
- **Format Check:** Prettier formatting validation ensures consistent code style. The check fails if files are not properly formatted, prompting developers to run the formatter.
- **Build:** Full project build verification ensures all components compile successfully.
- **Test:** Jest unit tests are executed with a PostgreSQL database service container. The test suite includes database schema migrations via Drizzle ORM before running tests.

All code quality checks must pass before a pull request can be merged into the main branch, guaranteeing that only validated, production-ready code enters the repository.

#### 8.1.2 Documentation Quality Checks

For LaTeX documentation changes, the pipeline performs specialized checks:

- **LaTeX Lint:** ChkTeX analyzes LaTeX files for common errors and style issues.
- **LaTeX Format Check:** `latexindent` validates that LaTeX files follow consistent formatting standards.
- **LaTeX Compilation:** All LaTeX documents are compiled across multiple directories (Checklists, Design documents, SRS, VnV documents, project management documents, etc.) using a matrix strategy. The compilation process handles both simple documents and documents requiring BibTeX bibliography processing.

### 8.2 Continuous Deployment

The `buildtex.yml` workflow handles automated PDF generation for LaTeX documents. When LaTeX files are modified and merged to the main branch, the workflow:

- Detects changed `.tex` files in the `docs/` directory.
- Compiles each changed document using the full TeXLive distribution.
- Automatically commits the generated PDF files back to the repository.

This ensures that PDF documentation is always up-to-date with the latest LaTeX source files without requiring manual compilation.

### 8.3 Design Considerations

The CI/CD pipeline emphasizes speed, consistency, and modularity:

- **Concurrency Control:** Workflows use GitHub's concurrency groups to cancel in-progress runs when new commits are pushed, preventing unnecessary resource consumption.
- **Path-Based Filtering:** Change detection ensures that code checks only run when code files change, and LaTeX checks only run when documentation changes, significantly reducing build times.
- **Matrix Strategy:** LaTeX compilation uses a matrix to compile documents in parallel across multiple directories, improving efficiency.
- **Fail-Fast Strategy:** The pipeline uses a final aggregation job that checks all individual job results, providing clear feedback on which checks failed.

This automated pipeline maintains project stability and code quality as new features and documentation are introduced, while minimizing developer friction through intelligent optimization.

# 9 Team Charter Trigger Items

## 9.1 Summary of Quantified Triggers

The team charter, as documented in the Development Plan, establishes the following quantified metrics to ensure accountability and consistent contribution from all team members:

- **Attendance:** Members are expected to attend at least 90% of scheduled team meetings, supervisor/stakeholder meetings, and lectures.
- **Commits:** Each member should make regular commits reflecting steady progress, with a minimum of 2 commits per week per person.
- **Issues:** All members must contribute to opening, discussing, and closing GitHub Issues, with at least 1–2 issues closed per week per person.
- **Documentation & Reviews:** Members are expected to contribute to documentation and review teammates' work at least once per deliverable.
- **Unexcused Absences:** No more than 2 unexcused absences per term are permitted.

These triggers are designed to ensure balanced workload distribution, consistent progress, and high-quality deliverables throughout the project lifecycle.

## 9.2 Violations

During the Rev 0 period, the team has not experienced any violations of the established triggers. All team members have:

- Maintained attendance above the 90% threshold for all meeting types (team meetings, supervisor meetings, lectures, and TA document discussions).
- Made regular commits to the repository, contributing to the project's steady progress.
- Actively participated in issue creation, assignment, and resolution.
- Engaged in code and documentation reviews as part of the pull request process.

Any absences that occurred were communicated to the team in advance with valid reasons, ensuring transparency and maintaining team cohesion. For example, Burhanuddin Kharodawala was unable to attend one TA document discussion session due to being out of town, and this absence was communicated to the team beforehand.

### 9.3 Plan to Address Future Violations

The team charter outlines a progressive consequence structure for addressing violations:

- **Initial Response:** Members falling short of expectations will first receive a reminder and the opportunity to catch up on missed contributions.
- **Escalation:** If underperformance persists, the issue will be discussed with the TA or instructor and reflected in peer evaluations.
- **Minor Infractions:** For minor misses (e.g., attendance or commits), light-hearted consequences may be applied, such as bringing coffee or snacks to the next meeting, maintaining a positive team culture while reinforcing expectations.

Throughout the project, the team will continue to assess the triggers and modify them as appropriate if they are found to be too weak, too strong, or ambiguous. The goal is to maintain realistic expectations that promote consistent contribution while allowing for flexibility when team members face legitimate challenges.

## 10 Additional Productivity Metrics

Beyond the standard metrics tracked in previous sections, the team employs several additional productivity indicators to assess overall project health and team performance.

### 10.1 Pull Request Review Process

The team maintains a structured code review process where:

- All pull requests require review by at least one team member (the designated Reviewer) before merging to the main branch.
- Pull requests must include clear titles and descriptions explaining the changes.
- Code reviews are conducted via GitHub's pull request interface, ensuring standardized review procedures.
- All CI/CD checks must pass before a pull request can be merged, enforcing code quality standards.

This process ensures that code quality is maintained through peer review and that all team members stay informed about project changes.

## 10.2 CI/CD Pipeline Health

The team tracks the health of the CI/CD pipeline as an indicator of code quality and integration stability:

- **Build Success Rate:** The pipeline's ability to successfully compile and test all components reflects the stability of the codebase.
- **Automated Quality Checks:** The consistent passing of type checks, linting, formatting, and tests indicates adherence to project standards.
- **Documentation Compilation:** Successful LaTeX compilation across all documentation directories ensures documentation quality and completeness.

During Rev 0, the CI/CD pipeline has maintained high reliability, with automated checks catching issues early and preventing problematic code from entering the main branch.

## 10.3 Code Quality Metrics

The team uses automated tools to maintain code quality:

- **Type Safety:** TypeScript's type system provides compile-time error detection, reducing runtime bugs.
- **Static Analysis:** ESLint and Prettier enforce consistent coding standards and catch potential issues.
- **Test Coverage:** Unit tests verify the correctness of critical business logic and components.

These metrics help ensure that the codebase remains maintainable and reliable as the project scales.

## 10.4 Collaboration and Communication

The team tracks collaboration effectiveness through:

- **Issue Management:** Active use of GitHub Issues for task tracking, discussion, and progress monitoring.
- **Meeting Documentation:** Regular team meetings documented through GitHub Issues ensure accountability and knowledge sharing.
- **Documentation Contributions:** Team members contribute to both code documentation and project documentation (LaTeX documents), ensuring comprehensive project knowledge.

These metrics reflect the team's commitment to transparent communication and collaborative development practices.

## 10.5 Project Organization

The team uses GitHub Projects as a central tool for organizing tasks and tracking progress:

- Issues are organized on project boards with appropriate labels (documentation, task, bug, feature).
- Tasks are broken down into manageable issues assigned to specific team members.
- Progress is tracked through issue status and board organization.

This organizational approach ensures that work is well-planned, assigned, and tracked throughout each deliverable period.