Test Document

for

Ride With Us

Version 1.0

Prepared by

Group #: 20 **Group Name: TITANS**

AYUSH YADAV	210251	ayushy21@iitk.ac.in
HARSHIT PATEL	210424	harshitptl21@iitk.ac.in
JATOTH SHASHI VARDAN	230501	shashivj23@iitk.ac.in
KANDULA AMARNADHU	230522	amarkn23@iitk.ac.in
PAL AJAY RAMSAGAR	230725	palajayr23@iitk.ac.in
ROHIT VINOD ATKURKAR	230872	rohitv23@iitk.ac.in
SANGA BADRI	230911	sangabadri23@iitk.ac.in
SUGALI YASHWANTH NAIK	231046	synaik23@iitk.ac.in
SUNANDINI BANSAL	EXY24032	sunandinib@iitk.ac.in
V HARIVANSH	231109	vhari23@iitk.ac.in

Course:	CS253
Mentor TA:	ASHISH SINGH (ashishsg24@iitk.ac.in)
Date:	05/04/2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	AYUSH YADAV HARSHIT PATEL JATOTH SHASHI VARDAN KANDULA AMARNADHU PAL AJAY RAMSAGAR ROHIT VINOD ATKURKAR SANGA BADRI SUGALI YASHWANTH NAIK SUNANDINI BANSAL V HARIVANSH	Initial Tests of the project	04/04/2025

1 Introduction

1.1 Test Strategy

The testing approach employed for this project involved a combination of **manual testing** and **test automation** to ensure comprehensive quality coverage.

- **Manual Testing:** Manual testing was used to validate specific user interactions, UI/UX elements, and edge cases that require human judgment and understanding.
- **Test Automation:** Automation was employed for repetitive and regression testing tasks, helping to quickly validate that the application worked as expected after changes.

1.2 Timing of Testing

Testing activities were conducted both **in parallel with implementation** and **after the implementation was completed**, ensuring continuous validation and high-quality assurance throughout the development lifecycle.

1.3 Testers Involved

- **Developers:** The entire development team was involved in testing, primarily focusing on unit tests, integration tests, and code block testing. Developers were responsible for testing individual features and components to verify correctness and functionality.
- **External Users:** Some external users also contributed to the testing efforts, focusing on user acceptance, usability, and end-to-end system validation.

1.4 Coverage Criteria

The coverage criteria were designed to ensure that all critical parts of the application were thoroughly tested, including the following:

- **Major Code Blocks:** All key functionality and significant code blocks of the system were covered by tests, including both positive and negative scenarios.
- **End-to-End Flows:** Test cases also included major user flows, ensuring that the application performs as expected under normal and edge-case conditions.
- **Regression Testing:** Automated tests were created to ensure that previously tested functionality continued to work correctly after any new changes were implemented.

1.5 Testing Tools Used

- **PHP Testing Tools:** PHPUnit
- **JavaScript Testing Tools:** For the JavaScript components, Jest and Jasmine
- **Performance Testing Tools:** K6

2 Unit Testing

2.1 User Management Unit

Unit Details:

- **Class:** user.php
- **Functions:** encrypt_password(), authenticate_user(), add_user(), get_user(), user_exists(), user_logged_in(), get_logged_in_user(), logout_user()

Test Results:

User Registration and Authentication

- **Test Case 1:** Valid Registration
  - **Input:** Valid user data
  - **Expected:** User created in database
  - **SQL Verification:** SELECT * FROM user WHERE email_address = 'test@example.com';
  - **Result:** PASS
- **Test Case 2:** User Authentication
  - **Input:** Valid credentials
  - **Expected:** User logged in successfully
  - **Verification:** Check session variables
  - **Result:** PASS
- **Test Case 3:** Session Management
  - **Input:** User login/logout actions
  - **Expected:** Session variables set/cleared correctly
  - **Verification:** Check $_SESSION state
  - **Result:** PASS

Additional Comments:

- Password hashing verified
- Session management tested
- Duplicate email checks validated

2.2 Trip Management Unit

Unit Details:

- **Class:** database.php
- **Functions:** add_trip(), get_trip(), get_trips_near_on(), update_spots_taken(), get_drives_for(), get_rides_for(), delete_trip()

Test Results:
Trip Operations

- **Test Case 1:** Trip Creation
    - **Input:** Valid trip data
    - **Expected:** Trip created in database
    - **SQL Verification:** SELECT * FROM trip WHERE driver_id = [test_id];
    - **Result:** PASS
- **Test Case 2:** Trip Listing
    - **Input:** User ID
    - **Expected:** User's trips returned
    - **SQL Verification:** SELECT * FROM trip WHERE driver_id = [test_id] OR id IN (SELECT trip_id FROM trip_request WHERE user_id = [test_id]);
    - **Result:** PASS
- **Test Case 3:** Trip Deletion
    - **Input:** Trip ID
    - **Expected:** Trip and associated data deleted
    - **SQL Verification:** SELECT COUNT(*) FROM trip WHERE id = [test_id];
    - **Result:** PASS
- **Test Case 4:** Spots Management
    - **Input:** Updated spots count
    - **Expected:** Spots taken updated correctly
    - **SQL Verification:** SELECT spots, spots_taken FROM trip WHERE id = [test_id];
    - **Result:** PASS

Additional Comments:

- Location validation verified
- Time format checked
- Cascade deletion tested
- Spots constraint verified (spots_taken <= spots)

2.3 Search and Share Unit

Unit Details:

- **Class:** search_ajax.php, share_post.php
- **Functions:** search_get(), request_post(), share_post()

Test Results:
Search Operations

- **Test Case 1:** Basic Search
    - **Input:** Route and time parameters

    - **Expected:** Matching trips returned
    - **Verification:** Check returned trip data
    - **Result:** PASS
- **Test Case 2:** Women-Only Search
    - **Input:** women_only flag
    - **Expected:** Filtered trips based on gender
    - **Verification:** Check trip filtering
    - **Result:** PASS
- **Test Case 3:** Route Matching
    - **Input:** Route coordinates
    - **Expected:** Nearby trips returned
    - **Verification:** Check coordinates calculations
    - **Result:** PASS

Share Operations

- **Test Case 1:** Trip Sharing
    - **Input:** Trip details
    - **Expected:** Trip created with places
    - **SQL Verification:** SELECT * FROM trip WHERE driver_id = [test_id];
    - **Result:** PASS
- **Test Case 2:** Place Management
    - **Input:** Origin and destination
    - **Expected:** Places created and linked
    - **SQL Verification:** SELECT * FROM place WHERE id IN (SELECT origin_id, destination_id FROM trip WHERE id = [test_id]);
    - **Result:** PASS
- **Test Case 3:** Women-Only Trips
    - **Input:** Trip with women_only flag
    - **Expected:** Trip created with women_only set
    - **SQL Verification:** SELECT women_only FROM trip WHERE id = [test_id];
    - **Result:** PASS

Additional Comments:

- Search filters verified
- Place management tested
- Trip creation verified
- Route matching validated
- Women-only flag tested

2.4 Ride Request Unit

Unit Details:

- **Class:** database.php
- **Functions:** request_ride(), update_ride_request_status(), get_requests_for_trip()

Test Results:

Request Management

- **Test Case 1:** Request Creation
    - **Input:** Valid request data
    - **Expected:** Request created
    - **SQL Verification:** SELECT * FROM trip_request WHERE trip_id = [test_id];
    - **Result:** PASS
- **Test Case 2:** Request Status Update
    - **Input:** New status
    - **Expected:** Status updated
    - **SQL Verification:** SELECT status FROM trip_request WHERE trip_id = [test_id];
    - **Result:** PASS
- **Test Case 3:** Request Retrieval
    - **Input:** Trip ID
    - **Expected:** All requests returned
    - **SQL Verification:** SELECT COUNT(*) FROM trip_request WHERE trip_id = [test_id];
    - **Result:** PASS

Additional Comments:

- Status transitions verified
- Spot management checked
- Request validation confirmed

2.5 Utility Functions Unit

Unit Details:

- **Class:** functions.php
- **Functions:** sanitize_string(), html_respond(), json_respond()

Test Results:

Core Utilities

- **Test Case 1:** Input Sanitization
  - **Input:** Regular text
  - **Expected:** Cleaned text
  - **Verification:** Check output string
  - **Result:** PASS
- **Test Case 2:** Response Formatting
  - **Input:** Status and data
  - **Expected:** Properly formatted response
  - **Verification:** Check HTML/JSON output
  - **Result:** PASS

Additional Comments:

- Input validation verified
- Response formats checked

3 Integration Testing

3.1 Integration Testing Strategy

Testing Approach

1. **Core Module Testing**
   - Test user authentication flow
   - Test trip and request management
   - Test search functionality
   - Test session handling
   - Test database constraints
   - Test error handling
   - Test real-time updates
2. **Integration Points**
   - User login → Trip operations
   - Trip creation → Request handling
   - Search → Request creation
   - Session → Feature access
   - Database → Business logic
   - Error handling → User feedback
   - Status updates → UI updates
3. **Test Environment**
   - Local development setup
   - Test database with sample data
   - Basic test cases

3.2 User Authentication Integration

Module Details

- **Classes:** user.php, functions.php
- **Integration Points:** Login → Session → Feature access

Test Results

Test Case 1: Basic Login Flow

- Login with valid credentials
- Check session variables
- Access protected features
- **Result:** PASS

Test Case 2: Session Persistence

- Login and perform actions
- Check session maintained
- Logout and verify session cleared
- **Result:** PASS

Test Case 3: Access Control

- Try accessing without login
- Verify access denied
- Login and retry
- **Result:** PASS

Additional Comments

- Basic session handling working
- Some edge cases need testing
- Security checks implemented

3.3 Trip and Request Integration

Module Details

- **Classes:** database.php
- **Integration Points:** Trip → Request → Spots management

Test Results

Test Case 1: Basic Trip Flow

- Create trip
- Submit request
- Check spots updated
- **Result:** PASS

Test Case 2: Request Management

- Update request status
- Verify spots count
- Check trip availability
- **Result:** PASS

Test Case 3: Women-Only Trips

- Create women-only trip
- Submit request (male user)
- Verify request rejected
- **Result:** PASS

Additional Comments

- Basic trip flow working
- Spots management needs improvement
- Gender restrictions implemented

3.4 Search and Request Integration

Module Details

- **Classes:** search_ajax.php, database.php
- **Integration Points:** Search → Request → Status updates

Test Results

Test Case 1: Search to Request

- Search for trips
- Select trip
- Submit request
- **Result:** PASS

Test Case 2: Filter Integration

- Apply women-only filter
- Check request handling
- **Result:** PASS

Test Case 3: Status Updates

- Update request status
- Check search results
- **Result:** PASS

Additional Comments

- Basic search working
- Filter implementation needs review
- Real-time updates could be improved

3.5 Database Constraints Integration

Module Details

- **Classes:** database.php
- **Integration Points:** Database → Business logic → User interface

Test Results

Test Case 1: Spots Management

- Create trip with 4 spots
- Submit 5 requests

- Verify only 4 accepted
- **Result:** PASS

Test Case 2: Request Status Flow

- Submit request (pending)
- Accept request (accepted)
- Complete trip (completed)
- Verify status transitions
- **Result:** PASS

Test Case 3: User-Trip Relationships

- Create trip as driver
- Submit request as rider
- Verify correct relationships
- **Result:** PASS

Additional Comments

- Spots constraint enforced
- Status transitions validated
- Relationships maintained

3.6 Error Handling Integration

Module Details

- **Classes:** database.php, functions.php
- **Integration Points:** Error handling → User feedback → Recovery

Test Results

Test Case 1: Invalid Input Handling

- Submit invalid trip data
- Verify error message
- Check database state
- **Result:** PASS

Test Case 2: Database Error Recovery

- Simulate database error
- Verify graceful failure
- Check error logging
- **Result:** PASS

Additional Comments

- Error messages clear
- Recovery mechanisms working

3.7 Real-Time Updates Integration

Module Details

- **Classes:** database.php, search_ajax.php
- **Integration Points:** Status changes → UI updates

Test Results

Test Case 1: Trip Status Updates

- Update trip status
- Verify search results
- **Result:** PASS

Test Case 2: Request Status Updates

- Update request status
- Verify spots count
- **Result:** PASS

Test Case 3: Spots Availability

- Update spots taken
- Verify spots remaining
- Check real-time updates
- **Result:** PASS

Additional Comments

- Status updates working
- Real-time sync verified

4 System Testing

4.1 System Testing Strategy

The system testing focuses on testing the complete web application, ensuring all PHP scripts and MySQL database operations work together correctly.

Testing Approach

1. **End-to-End Testing**
    - Complete user journeys through the web interface.
    - Database operations and transactions.
    - Form submissions and validations.
    - Session management across pages.
    - Negative testing for invalid inputs.
    - Data integrity testing.
2. **Test Environment**
    - Local XAMPP/WAMP server setup.
    - MySQL database with test data.
    - Web server (Apache) configuration.
3. **Testing Tools**
    - Browser developer tools.
    - MySQL Workbench for database monitoring.
    - Network tab for AJAX requests.
    - PHP error logging.

4.2 User Journey Testing

Testing complete user journeys through web interface

Test Cases

Test Case 1: Driver Journey

1. **User Registration**
    - Fill registration form in register.php.
    - Submit to register_post.php.
    - Verify database entry.
    - **Result:** PASS
2. **Trip Creation**
    - Login through login.php.
    - Access share.php.
    - Submit trip form.
    - Verify trip in database.
    - **Result:** PASS
3. **Request Management**
    - View requests in trips.php.
    - Accept/reject through trips_ajax.php.

    - Check database updates.
    - **Result:** PASS
4. **Trip Completion**
    - Update status through trips_ajax.php.
    - Verify request statuses.
    - Check history in trips.php.
    - **Result:** PASS

Test Case 2: Rider Journey

1. **User Registration**
    - Complete registration form in register.php.
    - Verify database entry.
    - **Result:** PASS
2. **Trip Search**
    - Use search.php interface.
    - Submit search parameters.
    - Verify results display.
    - **Result:** PASS
3. **Request Process**
    - Submit request through trips_ajax.php.
    - Track status in trips.php.
    - Check database updates.
    - **Result:** PASS
4. **Trip Completion**
    - Verify status updates.
    - Check history in trips.php.
    - **Result:** PASS

Additional Comments

- All PHP scripts working correctly.
- Database operations verified.
- Session persistence confirmed.
- Form submissions successful.

4.3 Performance Testing

Testing web application performance

Test Cases

Test Case 1: Page Load Performance

- Homepage (index.php) load time.
- Search results (search.php) loading.
- Trip details (trips.php) page.
- Response time < 1s.
- **Result:** PASS

Test Case 2: Database Query Performance

- Complex search queries (search_ajax.php).
- Multiple table joins (database.php).

- Large result sets.
- Query time < 500ms.
- **Result:** PASS

Test Case 3: Form Submission

- Registration form (register_post.php).
- Trip creation form (share_post.php).
- Request submission (trips_ajax.php).
- Response time < 1s.
- **Result:** PASS

Additional Comments

- PHP execution time acceptable.
- Database queries optimized.
- Form handling is efficient.

4.4 Security Testing

Testing web application security

Test Cases

Test Case 1: Authentication

- Login form security (login.php).
- Session management (user.php).
- Password hashing (user.php).
- **Result:** PASS

Test Case 2: Authorization

- Page access control (functions.php).
- Database permissions (database.php).
- API endpoint security.
- **Result:** PASS

Test Case 3: Input Validation

- Form data sanitization (functions.php).
- SQL injection prevention (database.php).
- **Result:** PASS

Additional Comments

- Security headers set.
- Input validation working.
- Database security verified.

5 Conclusion

5.1 Effectiveness and Exhaustiveness of Testing

The testing process was largely **effective** and **exhaustive**, with the following outcomes:

- **Comprehensive Coverage:** All major code blocks, including user management, trip management, search, and request handling, were adequately tested.
- **Performance Testing:** The system's performance was validated, ensuring pages loaded quickly and queries ran efficiently.
- **Security Testing:** Key security concerns, such as input validation, session management, and database permissions, were rigorously tested.

However, some areas could benefit from further in-depth testing:

- **Edge Cases:** Some edge cases and rare user scenarios may not have been fully explored.
- **Real-Time Updates:** While real-time updates were tested, further performance validation could be done to ensure the system scales well under heavy user loads.

5.2 Components Not Adequately Tested

While the majority of the components were thoroughly tested, there were some areas that could be improved:

- **User Roles and Permissions:** Although user authentication and basic session handling were thoroughly tested, more attention could be paid to complex user roles and permissions, especially in terms of access control for different features.
- **Stress Testing:** While performance was tested for general functionality, stress testing under high loads and concurrent users could have been more exhaustive.
- **Mobile Responsiveness:** The testing strategy did not include extensive checks for mobile responsiveness and behavior on different devices.

5.3 Difficulties Faced During Testing

Several challenges were encountered during the testing process:

- **Integration Complexity:** Since the project involved multiple modules interacting with each other, integration testing sometimes revealed unexpected dependencies, making debugging more complex.
- **Test Data Management:** Maintaining consistent and meaningful test data in the test environment was occasionally challenging.

5.4 Improvements for the Testing Process

Several improvements could be made to enhance the overall testing process:

- **Enhanced Edge Case Testing:** Focus more on testing unusual or unexpected user behavior and edge cases, particularly in integrations and real-time updates.
- **Stress and Load Testing:** Implement dedicated stress and load testing to evaluate how the system handles high traffic and concurrent users. Tools like Apache JMeter could be useful for this purpose.
- **Cross-Device Testing:** Ensure that the application works flawlessly across different devices and browsers. Testing for mobile responsiveness and browser compatibility could be more extensive.
- **Automated UI Testing:** While functional tests were automated, adding UI testing automation (e.g., using Selenium) would help ensure the visual aspects of the application remain consistent.

Appendix A - Group Log

<*Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist in determining the effort put forth to produce this document*>