Test Plan - SafeRide

Team: #0000FF Thunder
An Ta
Andy Lee
Frankie Htet
Leon Chen
Orion Tang (Team Leader)

Date Issued:
October 27th, 2021

1. Introduction

1.1 Purpose

This test plan document outlines the strategies to be used to test the initial full release of the SafeRide web application.

Section 2, test strategy, details the strategies to be employed while testing. This section is meant to give testers the requisite knowledge to successfully carry out software tests.

Section 3, execution strategy, details the specific actions the team will take to carry out testing.

Section 4, test scenarios, lays out the high-level scenarios to be tested and provides a framework for testers to create test cases when features are being developed and tested during sprints.

1.2 Project Overview

SafeRide is a web application designed for micro-mobility commuters to help them safely navigate between destinations. Out of all available routes, it will provide the user with the safest route based on factors like live traffic density, accident history along said route, availability of bike lanes, and other key factors. It will also provide to the user an analysis of the provided route with graphical overview of key statistics. SafeRide will additionally allow users to report hazards as they see them, promoting route transparency and allowing riders to prepare for these hazards ahead of time.

Software and hardware specifications: SafeRide is to run on the latest stable release of the Chrome web browser upon its release. Currently this is version 95.0.4638.54 for Windows, Linux, and macOS, and 95.0.4638.50 for Android and iOS. Any new stable versions of Chrome that are released during the development cycle will be supported for final release. SafeRide is intended to work on any device, desktop or mobile, that is capable of running this most recent Chrome release.

2. Test Strategy

2.1 Test Objectives

Testing is carried out to verify that the final release of SafeRide adheres to all specified requirements. Additionally, test scripts that are created for automated testing are to be reusable for future testing. Test cases will be written, tests will be run and automated where appropriate, and defects will be logged and rated to aid in future fixes.

Through testing, SafeRide will be a production-ready web application with test documentation and scripts to aid in maintaining the software.

2.2 Test principles

- Tests will be centered on meeting the functional and non-functional requirements of each feature
- Procedures will be consistent across all tests done on the system
- Testing activity is to be measurable and repeatable
- Test environment will emulate a production environment
- Unit testing is to be conducted for every completed feature before integration
- Regression testing is to be conducted for the entire system after new functionality is added to ensure the system still works as intended
- Testing will be flexible to account for changes in the project during the development process

2.3 Scope

Work listed below is to be performed by the team in order to complete the aforementioned test objectives.

- Testing on all SafeRide functionalities listed in this document and the Business Requirements document
- Documentation on tests which include the area to be tested, test name, the test steps, and the expected results
- Reports will be generated for each test performed based on the test type

2.4 Testing Types

2.4.1 Exploratory Testing

Purpose: The purpose of exploratory testing is to uncover defects that are not considered through more explicit testing methods.

Method: Exploratory testing is carried out without documentation or scripts. It is more reliant on tester intuition and tends to point out defects in complex workflows or UI. If the tester thinks something can go wrong that is not covered by a test case, they can perform an exploratory test.

Test Deliverables: Because exploratory testing does not rely on premade test cases, the tester will only document the scenario they tested that led to the issue.

2.4.2 Unit Testing

Purpose: Unit testing ensures each individual feature functions as intended, e.g. creating an account properly saves the account in the database.

Method: Test scenarios are detailed in this document. Unit testing will follow each test scenario and be automated where possible to ensure regression testing is consistent and reduce duplication of effort. Each case will test that the feature under examination meets both functional and non-functional requirements.

2.4.3 Integration Testing

Purpose: Integration testing ensures microservices work together as intended as they are introduced. They test that data is being transferred correctly or databases are being properly updated.

Method: If there are dependencies in the system, i.e. a microservice depends on another to function, integration testing will take place to make sure they interact with each other properly. Integration test cases are outlined in this document.

2.4.4 Regression Testing

Purpose: Regression testing ensures that introduction of new features does not introduce new defects.

Method: Tests previously run will be run again. Scripting of tests will make regression testing more consistent and efficient.

2.5 Test Tools

- Selenium (to be confirmed with tech review)
    - For automated testing of web app
- Google Sheets
    - For creating test reports
- Visual Studio 2019
    - Provides performance and load testing capabilities to ensure non-functional requirements are met

2.6 Test Environment

All members of the team are required to set up and test in the following environment to ensure consistency across tests.
Web server: IIS 10+/Node.js (to be confirmed with tech review)
Database: SQL Server 2019
OS: Windows 10 64bit
Minimum Hardware:

Processor: Intel i5-7200U or equivalent
Memory: 8GB RAM
Storage: 16GB available

Browser: Google Chrome 95.0.4638.54+ (desktop) / 95.0.4638.50+ (mobile)

2.7 Test Data

For features that require existing user data in order to be tested, this data will be created and pre-loaded specifically for testing purposes.

3. Execution Strategy

3.1 Test Team

The SafeRide team does not have dedicated testers. Instead, because of the small size of the team, each person will partake in some sort of testing. The team leader will oversee testing and ensure that team members understand the tasks they are carrying out.

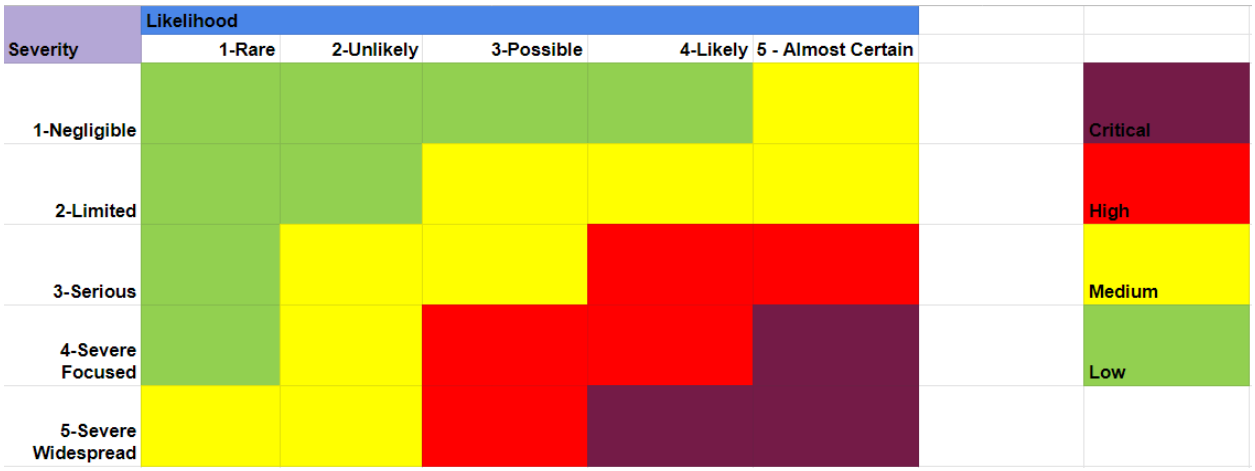
3.2 Exit Criteria

In order to consider testing for each sprint iteration complete, the following exit criteria must be met. If an item cannot be met whatsoever, the risk of leaving this item incomplete will be assessed and used for future decision making.

- 100% of created test cases are run
- At least 95% pass rate for test scripts
- No critical severity defects remain
- Uncovered defects are documented
- Newly created tests are saved for regression testing
- At least 95% of medium and high severity defects have been fixed or mitigated

3.3 Defect Risk Analysis

The following matrix is to be used in determining the risk of an uncovered defect. The tester will use their best discretion when it comes to rating a defect.



Critical: The defect is likely to occur and can cause data loss, system crashes, or other damage that takes significant resources to recover from.
High: The defect can occur with reasonable frequency and can cause loss of vital functionality of the software.
Medium: The defect can lead to loss of quality, but not loss of vital functionality.
Low: The defect has little impact on software use, e.g. a cosmetic defect.

3.4 Test Schedule

Since the team is developing using an Agile process, testing will take place during every sprint. Developers will test the features that they are tasked with developing during each sprint. Testing is expected to be completed before the end of each sprint and reports will be included in the sprint review. See project plan document for more details on sprint scheduling.

There will be one last comprehensive test phase once all software functionality, as defined in the BRD, is complete for SafeRide. This will ensure the software to be moved into the production environment fully functions as intended for end-users and all requirements are met.

Covering each test scenario for each feature is intended to take up to a full day of work (8 hours) or less. This is to ensure testing is feasibly completable by the team within a sprint.

3.5 Testing Risks and Mitigations

Risk: Testing may fall behind schedule and affect the progress of development and future testing.
Risk level: Medium. It is likely that certain testing items will fail to be completed due to the nature of the team's schedule.
Mitigation: Each sprint is scheduled to account for the time it will take to test and each feature is written to take approximately a full work day to test. If a testing item could not be completed, the risk of leaving that item incomplete will be assessed and if the risk is medium to critical, the testing will have to be done in a different sprint.

Risk: A defect is found late into the development process
Risk level: Medium.
Mitigation: Test scenarios are written to cover as many scenarios as possible. Additionally, the team can be proactive in adding scenarios as needed due to the flexibility of testing and it is likely new scenarios will be uncovered as development takes place.

Risk: Natural disasters
Risk level: Low. Natural disasters are possible but rare.
Mitigation: The team is working mostly virtually and has remote development capability, so in the case of a natural disaster, project resources will largely remain unaffected.

Risk: Project requirements continuously change and leads to significant extra work
RIsk level: High. Since the team is using agile, it is likely requirements can change and impact the amount of work to be done.
Mitigation: The scope laid out in the test plan and BRD are intended to prevent the project from growing to unmanageable levels. Any change to the project should be within scope to prevent large scope creep.

4. Test Scenarios

The following test scenarios reference the use cases for SafeRide as defined in the business requirements document. They are to be referenced when creating the actual test cases during a sprint.

Test cases will be written by the tester during the sprint which focuses on completing the use case. They are a low-level representation of the test scenario and are more granular in order to ensure consistency and reproducibility in the tests. They will adhere to the following format, as an example:

Test Scenario ID:
4.1.3

Test Scenario:
Verify that user must enter password adhering to security standards

Test type:
Unit

Test case name:
Check password below 8 characters

Test Steps:

1. Go to https://www.saferideapp.com
2. Select “Register account”
3. Enter valid email
4. Enter password below 8 characters

Expected outcome:
User is told their password is invalid and what a valid password consists of

Tests run:
10

Tests passed:
0

Tests failed:
10

Actual outcome:
0% of tests as expected
100% of tests, user is able to create account

Severity of actual outcome (if not expected):
Severe Widespread & Almost Certain: Critical

4.1 Register Account

References section 3.1 of BRD

4.1.1 Verify new user can create an account with new credentials

- Requires database connection and test data

4.1.2 Verify existing user cannot create duplicate account

- Requires database connection and test data

4.1.3 Verify user must enter password adhering to security standards

- Passwords should have at least 8 characters up to a maximum of 128 characters. Must contain a special character
- Passwords entered directly into our database (not using google login) will be hashed and salted for added security

4.1.4 Verify password storage adheres to security standards

- Requires database connection and test data
- Passwords are salted and hashed before being stored in the database

4.1.4 Verify Google login functionality

4.1.5 Verify register account functionality meets performance standards

- Account information will be logged in the database within 1 second

4.2 Log in

References section 3.2 of BRD

4.2.1 Verify user can log in

- Requires database connection and test data

4.2.2 Verify login functionality meets performance standards

- User shall be logged in within 1 second of entering valid credentials

4.3 Search for route and get directions

References section 3.3 of BRD

4.3.1 Verify risk rating algorithm outputs reasonable, consistent scores

- Requires database connection

4.3.2 Verify search functionality

- Locations are suggested in real time based on what user is typing in search box

4.3.3 Verify the route display functionality

4.3.4 Verify valid directions are returned to the user

4.3.5 Verify route search, directions, and display functionality meet performance standards

- User should still be able to interact with the map even if route is calculating (async)
- SafeRide will take no longer than 3 seconds to return route directions and display the route to the user from when the user requests direction

4.4 Add waypoint to route

References section 3.4 of BRD

4.4.1 Verify user can add waypoints correctly

- Up to a max of 4 waypoints can be added
- If maximum waypoints have been added, will tell the user such

4.4.2 Verify user can remove waypoints correctly

4.4.3 Verify directions and display are updated after waypoint is added

4.4.4 Verify waypoint functionality meets performance standards

- SafeRide will take no longer than 3 seconds to return route directions and display the route to the user from when the user updates the route

4.5 View route history

References section 3.5 of BRD

4.5.1 Verify past routes are displayed to user correctly

- Requires database connection and test data
- Up to the past 30 routes are displayed
- Each entry in route history displays start and end locations, number of waypoints, and date calculated
- If no routes are available, will tell the user such

4.5.2 Verify past routes are deleted correctly

- Requires database connection and test data
- Routes are automatically deleted in first-in/first-out order if list is full
- Routes are automatically deleted after 30 days

4.5.3 Verify route history is only available to logged in users

4.5.4 Verify route history functionality meets performance standards

- Routes will be displayed within 1 second of request

4.6 Save a route from route history

References section 3.6 of BRD

4.6.1 Verify route is saved correctly after saving it from route history

- Maximum of 10 routes can be saved
- If maximum has been reached, will tell the user such
- Requires database connection and test data

4.6.2 Verify all saved routes are displayed correctly

- Requires database connection and test data
- If no routes have been saved, tell the user such

4.6.3 Verify save route functionality meets performance standards

- Route will be saved within 1 second of request

4.7 Share a saved route

References section 3.7 of BRD

4.7.1 Verify shareable link is functional

- Link can be pasted into browser search bar and will automatically display route and return directions

4.7.2 Verify shareable link expires after 60 minutes

- Requires database connection and

4.7.3 Verify expired links, when used, take user to map view with empty search parameters

- Tell the user the link is not useable

4.8 Delete a saved route

References section 3.8 of BRD

4.8.1 Verify route is deleted correctly

- Requires database connection and test data
- Deletion shall require confirmation on behalf of the user

4.8.2 Verify shareable links associated with route expire upon route deletion

- Requires database connection and test data

4.8.3 Verify route deletion meets performance standards

- Deletion shall occur within 1 second of request

4.9 Use a saved route to get directions

References section 3.9 of BRD

4.9.1 Verify user selecting to use a saved route is functional

- Requires database connection and test data
- Automatically take user to map view, display route and return directions

4.9.2 Verify using a saved route meets performance standards

- User is taken back to map view within 1 second of request
- Route calculation will still take under 3 seconds after user has reached the map view

4.10 Use map overlays

References section 3.10 of BRD

4.10.1 Verify map overlays are functional

- A route must be selected in order for an overlay to be displayed
- One overlay can be selected at a time; selecting another overlay will disable the previous one

4.10.2 Verify map overlays are visually distinct and informative

- Overlays will be color coded based on intensity of specified metric
- Red - severe, yellow - medium, no color - low/no severity

4.10.3 Verify map overlays meet performance standards

- Overlay will be loaded onto the map asynchronously
- Overlay will be displayed within 3 seconds of request

4.11 Report hazards

References section 3.11 of BRD

4.11.1 Verify user can add hazard

- Requires database connection

4.11.2 Verify user can cancel hazard placement

4.11.3 Verify existing hazards are displayed correctly

- Requires database connection and test data
- If the same hazard types are reported within 10 meters of each other, these will be treated as the same hazard
- Hazard labels will display a counter of how many reports the hazard has gotten

4.11.4 Verify hazards persist and expire correctly

- Requires database connection and test data
- Hazards will last for 1 hour. This timer is reset is the hazard is reported again
- The maximum amount of hazards is 99. Oldest hazards will be deleted first if the limit is to be exceeded

4.11.5 Verify hazard report functionality meets performance standards

- Hazard will be saved in database within 1 second of reporting it

4.12 View safety analysis for a route

References section 3.12 of BRD

4.12.1 Verify safety analysis displays correctly

- Requires database connection
- Ensure graphs are correct to the data

4.12.2 Verify safety analysis functionality meets performance standards

- Route analysis is generated and displayed within 3 seconds of request

4.13 Change password

References section 3.13 of BRD

4.13.1 Verify user can change password

- Requires database connection and test data

4.13.2 Verify user cannot change password without correct credentials

- Requires database connection and test data

4.13.3 Verify new password adheres to security standards

- Passwords should have at least 8 characters up to a maximum of 128 characters. Must contain a special character
- Passwords entered directly into our database (not using google login) will be hashed and salted for added security

4.13.4 Verify password storage adheres to security standards

- Requires database connection and test data
- Passwords are salted and hashed before being stored in the database

4.13.5 Verify password change meets performance standards

- Password will be changed within 1 second of request

4.14 Delete account

References section 3.14 of BRD

4.14.1 Verify account deletion is functional

- Requires database connection and test data
- Requires confirmation on behalf of user

4.14.2 Verify account deletion meets performance standards

- Account shall be deleted within 1 second of request