Testing Strategy Outline

DriveBC

Code Challenge

November 30, 2022

Document version: 1.0

For:	Prepared by:
Evaluation Team Tahna Neilson Tahna.Neilson@gov.bc.ca	**Sophie Repussard** QA Manager sophie@oxd.com +1 604 694 0554
DriveBC	**OXD** 210–12 Water Street Vancouver, BC, V6B 1A5 Canada **oxd.com**

Testing Strategy

Background

This Code Challenge asked us to build an Open Geo-spatial Consortium (OGC) compliant web map application that displays road events, webcam images, a travel advisory message, and allows for email notifications for the users.

The business case is that DriveBC is the provincial traveler information system that is used to inform travelers of events that may impact their travel along a provincial highway. Travelers will need to view current and geo-spatially accurate information on a map, create routes and receive notifications of any new events along that route.

The web map-based application should allow map navigation and routing (using listed third-party APIs) as well as allowing users to view, save, and delete routes they create and receive email notifications of events along that route.

In the context of this time-limited Code Challenge, this document's purpose is to demonstrate how our team would test its solution if more time was available.

Scope

Approach

Our team will test end-to-end the web map application and its functionalities, including testing on mobile devices.

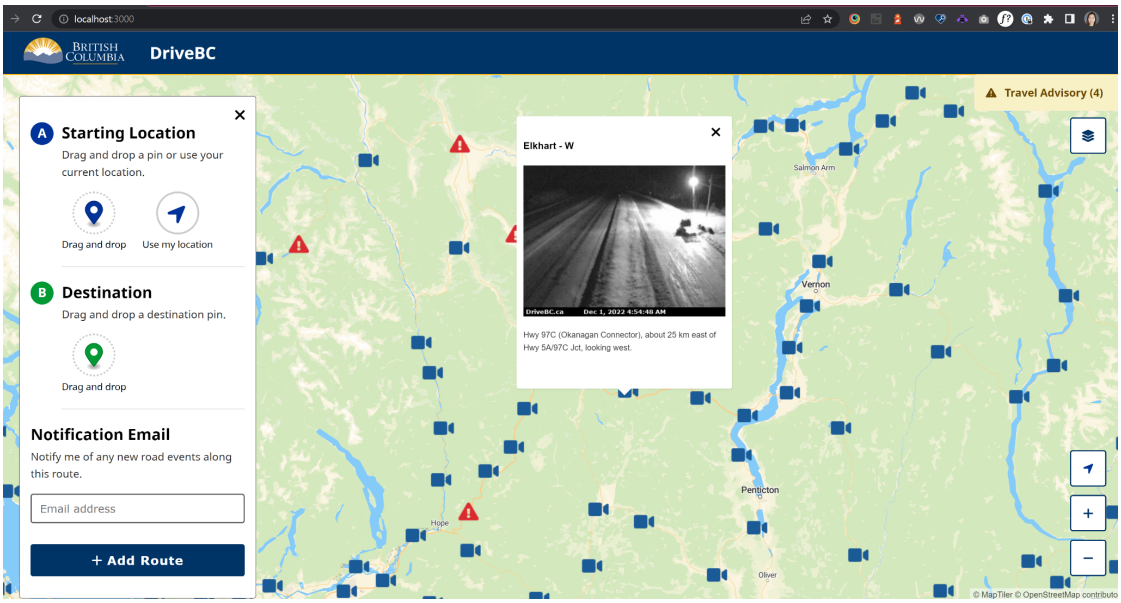
Activities and outcomes

Task 1: Unit testing and code reviews

- Using test-driven development, our developers will add unit tests in the code
- Those unit tests can be part of a CI/CD pipeline, so every code commit will trigger running the full test suite
- Peer code reviews will be conducted for every pull request via GitHub where tracking and history is available to adhere to working in an open-source environment
- **Outcome: Ensures correctness of code by introducing a faster code refactoring process**
- **For Code Challenge:** Some unit tests have been added to the code but due to the time constraint, we made the decision to bypass test-driven development processes

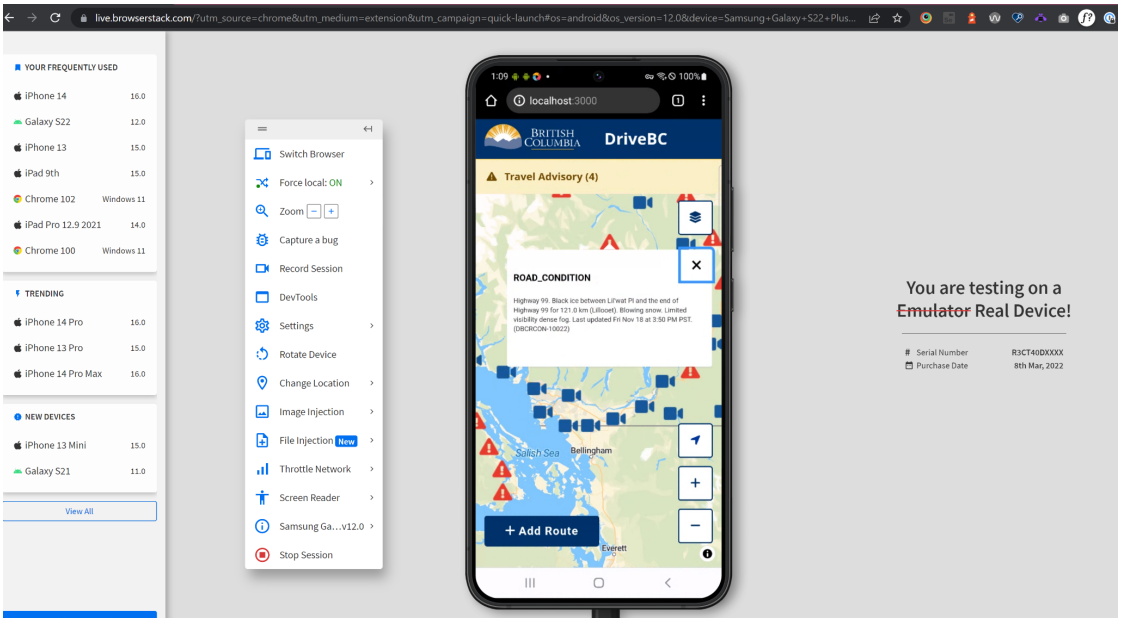
Task 2: Manual functional and exploratory testing (QA)

- Using Agile testing processes, tester(s) will manually test the functionalities against the documented requirements. This involves creating test plans and test cases and executing them via a Test Management Tool such as TestRail
- Cross-browser testing will be conducted based on a browser matrix created based on analyzing Google Analytics data of actual usage by current users of DriveBC
- API testing will be performed using Postman
- Regression testing will also be performed along the way
- **Outcome: Ensures the quality of the deliverable by ensuring the requirements are met and identifying issues along the development process**
- **For Code Challenge:** Exploratory testing has been performed using Chrome browser to test the requirements from the Instructions file and also based on wireframes designed by our UX designer (included in documentation submitted). We internally used a Jira board to track tasks and bug tickets to help team collaboration



Task 3: Mobile testing

- Using a third-party service called BrowserStack that allows testing on a large range of real devices, our team will test the application across different mobile devices and browsers
- Testing of mobile-specific gestures, especially around the map interface, will be conducted
- **Outcome: Ensures interacting with the application on a mobile device is seamless**
- **For Code Challenge:** We used BrowserStack and Chrome Inspector to verify that the site is responsive and functional on small screens



Task 4: User acceptance testing (UAT)

- As part of the release process, user acceptance testing is usually performed by a different team than the development team (usually by real world intended audience)
- Those testers are given the opportunity to interact with the product before its official release to see if any features have been overlooked or if it contains any bugs
- **Outcome: Ensure the application can handle real-world tasks and perform up to business specifications**
- **For Code Challenge:** Out of scope of this Code Challenge

Task 5: Performance and load testing

- Test server load and performance using tools such as Jmeter
- Test server load balancing and scalability for country/region
- Test server data expiration and allow monitoring and alerting
- Test server storage
- Test server data ingestion
- Test server logging and telemetry
    - Ensure adherence to applicable privacy laws and policies
    - Log success and failure at stages of critical user journeys
- **Outcome: Ensures the application can perform well under real life based load conditions**
- **For Code Challenge:** Out of scope of this Code Challenge

Task 6: Automation testing

- CI/CD pipelines are put in place to automate the delivery process by automatically building code, running tests and deploying the application to an environment. This can be done in GitHub Actions
- API testing can be automated using Postman collections and its Collection Runner
- UI automation test suites can be built with frameworks such as Cypress or Selenium
- **Outcome: Automating as much as possible will decrease the likelihood of regression bugs introduced by new code and will also reduce the risk of manual errors and provide standardized feedback loops to developers, and enable fast product iterations**
- **For Code Challenge:** Out of scope of this Code Challenge

Task 7: Security testing

- Following the OWASP testing recommendations and using tools such as Burp Suite, the team would perform:
    - security analysis,
    - static and dynamic testing in whitebox and blackbox environments,
    - as well as penetration testing and vulnerability scanning
- **Outcome: Identify the threats in the application and measure its potential vulnerabilities**
- **For Code Challenge:** Vulnerability scans are run on both frontend and backend projects (see README file)

Task 8: Accessibility testing

- Following the official guidelines from W3C, the team will perform accessibility testing on the application
- Testing involves manual testing, and using tools such as Wave, or screen reader software such as NVDA
- **Outcome:** The application should meet the WCAG 2.1 level AA criteria to meet industry standards for accessibility and inclusion
- **For Code Challenge:** Out of scope of this Code Challenge

Task 9: Usability testing

- We will conduct initial user research, including interviews directly with the public and exploratory usability testing to provide insight into how the public uses the current DriveBC site
- Then we will take a collaborative approach to co-creating initial designs with the public and finally create clickable wireframes to review with the team. This will result in a prototype for usability testing with citizens
- Ideally usability testing of opportunities or features should be done a few sprints ahead of planned development to ensure sufficient time for design and validation. At this stage changes and feedback can be incorporated into the designs more efficiently than later when changes must be made to code
- **Outcome: Engaging users for feedback throughout the project to uncover more opportunities for improvements which can be inserted into our Agile team's backlog**
- **For Code Challenge:** Out of scope of this Code Challenge

Quality Assurance Testing

Following the Agile testing methodology, features developed are tested to determine if the desired functionality was achieved and defect reports are logged for the developers, which will be fixed then retested again. A tracking tool such as Jira will be used to track those reports and facilitate team collaboration. All features are tested prior to being sent for UAT.

QA testers will run the first round of testing before Product Manager, Business Analysts, and end users test in a production-like setting.

Prior to release, the QA team should execute a suite of regression tests. Tests of this sort generally fall into two categories: smoke testing, which is a quick check to make sure that recent code changes have not had a serious impact on the product; and deeper regression testing, which aims to cover off as large a portion of the product's functionality as possible.

Smoke testing can be run on an ad hoc basis, or as a regular part of the build process. Smoke tests are lightweight and provide fast feedback on a product's status after changes are made. As a guideline, they should take under half an hour to perform manually. The ideal situation for tests such as these is that they are fully automated, and running as a task in the build pipeline. If desired, they can be used as a gatekeeping mechanism; if the smoke test fails, the build process is abandoned until a fix has been made to make the smoke test pass again.

A more extensive suite of regression tests should be executed before product release. These tests are normally executed manually, although some degree of automation may also be employed. In designing these tests, several factors need to be considered:

- Tests should cover at least the most common user flows
- Coverage should be as broad as possible, to exercise full system functionality
- Tests should have clear pass/fail criteria
- Tests should be reviewed with stakeholders (e.g. BAs, Dev, Support)
- The regression test suite should be periodically updated for relevance and to cover new functionality

Periodically testing application performance is recommended. The approach should be to determine if recent changes to the code have had an impact, positive or negative, on application performance. This can be made a task in a CI pipeline, or can be executed on an ad hoc basis. The recommended approach here is to first amass a set of performance baseline data, which can be used for comparison with ongoing performance tests. Should a code change result in an improvement in performance, baseline data should be updated.

Defects organization and prioritization

Defects found affecting the acceptance criteria of existing requirements are prioritized over developing new features. All defects are first assessed for priority/urgency to fix and assigned a priority by the tester. Following Agile methodologies and ceremonies, adjustments may be made during the backlog refinement process with the team.

Priority definitions for defects

Priority	Definition
Blocker	Feature does not function, blocks user progress - Must be fixed immediately. All other items can wait. Release cannot occur with a known blocker. Blockers should not be in a release sent for UAT.
Critical	Feature does not function as intended, and can in some cases block progress. Workarounds are available, but the feature cannot be released with this defect. Can be reproduced 100% of the time.
High	Feature does not function properly. Visibility to encounter the issue is high.
Medium	Feature may function incorrectly under specific circumstances. It can be challenging to reproduce. Issue does not occur on a common user path. Issue is not highly visible to the end user. Acceptable to release, but should exist in the backlog to be fixed in a future release.
Low	Would be nice to fix. Does not affect functionality in a meaningful way. Is not visible to the end user. Acceptable to release, but should exist in the backlog to be fixed in a future release.

Release management

Most of the time in small Agile teams, the task of managing releases is shared among the team members. Oftentimes, QA testers play a key role in the release management process since they already are very aware of the product's overall quality.

Tasks may include:

- Schedule UAT and Prod releases and maintain a release plan
- Controlling deployments to environments (such as Test, or Production)
- Setting up and maintaining a clear UAT process with UAT testers
- Handling communications to UAT testers, stakeholders and development team
- Assessing potential risks to releases and mitigating them