**Schejewel**

**Test Plan and Specification**

**March 20, 2015**

# 

# 

# Table of Contents

[1. Introduction](#h.9fzr2bz8qdzf)

[1.1 Objectives](#h.g3m2mp1tf5tw)

[1.2 Change to Testing Procedure](#h.yk9vib362g6x)

[1.3 Reference Material](#h.l2s1jvbkaemp)

[2. Requirements and Features](#h.6s8c5ybuddpw)

[Backend](#h.pngk9x2lm3yn)

[Frontend](#h.784hdl96rapi)

[3. Features To Be Tested](#h.ey8mskewsiai)

[Front end](#h.tg1un5vpgx5g)

[Backend](#h.y2pncwknjejk)

[4. Features Not To Be Tested](#h.npm1air1bfv)

[5. Approach](#h.ys532os467s3)

[5.1 Component Testing](#h.7x3yipuz98id)

[5.2 Integration Testing](#h.7finkgtsz8ce)

[5.3 Security Testing](#h.lpalq2ref8o3)

[5.4 Regression Testing](#h.tt40fgi67hnt)

[5.5 Beta Testing](#h.m7lecomn61np)

[6. Testing Process](#h.ky5atunuasfi)

[6.1 Test Deliverables](#h.oz3v939da77d)

[6.2 Testing Tasks Back End](#h.37n2raqrwgv8)

[6.3 Testing Tasks Front End](#h.4lkow2jc0rdf)

[6.4 Responsibilities](#h.y40617ha8kcm)

[6.5 Resources](#h.pv0mhlq7r4sf)

[6.6 Schedule](#h.ydl4s72s3tpz)

[7. Environmental Requirements](#h.m7ds1knav6qo)

[7.1 Software](#h.fnu8ap14ihr)

[7.2 Tools](#h.b18a40o24ja8)

[7.3 Risks and Assumptions](#h.miuc8bynoa3k)

# 

# 

# 1. Introduction

## 1.1 Objectives

The objective of this testing plan is provide a guide for submitting and testing code for Schejewel. Testing for each crucial component must be finished in order to push it live. Testing includes all component testing, unit testing, user testing, and integration testing. All software testing is to be done by the software developers that wrote it, and then is verified by the testing team. Our biggest objectives for testing for the Schejewel project are the algorithms for checking schedules with conflicts, checking for available tour times, and user ease of use testing. Reasoning for this is our average customers are not technically friendly people. Being able to have a program that is simple to use is of the utmost importance, or our customer base is gone. Checking schedules is the core functionality of what sets our program apart from every other scheduling program, so if that doesn’t work, we fail.

Right now we have one person on our team who is over all testing, and verifying. He will go through and check to make sure that all tests were written thoroughly in order to minimize bugs that make it through.

## 1.2 Change to Testing Procedure

As a small company, this document will change with approval of the manager as needed. Changes to this document and procedure for testing will be made known to the team through email.

## 1.3 Reference Material

All referenced material can be found on the project wikipedia page.

https://github.com/2russellsmith/Schejewel

# 2. Requirements and Features

## Backend

1. Log you in and out securely

1.1 Compare provided credentials to the database stored ones

1.2 Return a token if they successfully validate

1.3 Validate the token for successful login

1.4 Expire token after a fixed time

2. Add read only accounts to your account

2.1 Form allows accounts to be made from an existing account

2.2 New accounts made default to read only

2.3 Read only accounts cannot modify data in any way

2.4 Secure endpoints based on user roles

3. Easily create customized tour schedules

3.1 Form for creating new tours of a specific tour type

3.2 Form for creating new tour types

3.3 Tool that calculates where a tour of a given tour type could be inserted into a day

4. Have those tour schedules checked for conflicts

4.1 Auto check for conflicts/resolutions when tours are updated

4.2 Set flag in database on tours that are conflicted after they are detected

4.3 Remove flag once they are no longer in conflict

5. Report those conflicts with suggestions on how to fix them

6. Generate customized reports on tour's schedules, numbers, and costs

6.1 API tours crud operations

6.2 API resources crud operations

7. Communicate with other companies to coordinate sharing resources

7.1 API companies crud operations

8. Automatically update changes in cruise line

8.1 API cruise line crud operations

9. Report changes instantly in your schedule to corresponding cruise lines

10. Be able to add employees shifts to the schedules based on tours

10.1 Calculate an employee’s start and end time

10.2 Employee can view their schedule from tool, including what they will be doing at a detailed level

10.3 Privileged accounts can view all employee schedules through Schejewel

10.4 API user crud operations

10.5 API privilege crud operations

11. Individual reports for employees based on their duties

11.1 Accounts have different view permissions settings

11.2 Accounts with different view permissions have access to different data

## Frontend

1. Log in and out

12.1 Users will be directed to login page if tokens do not exist or are invalid

12.2 Users will be redirected to main page if they have valid tokens

1. Read Only Users

13.1 Read only users cannot see buttons that allow adding or changing data

1. Resources

14.1 Resources get fetched correctly from backend

14.2 Correctly display list of all resources to user

14.3 Should be able to filter resources by owner and tour

14.4 Can assign resources to tour and tour group

1. Tours

15.1 Tours get pulled fetched from backend

15.2 Correctly display list of all tours to user

15.3 Should be able to filter tours by owner and tour

15.4 User can edit tours and tour groups

1. Report those conflicts with suggestions on how to fix them Generate customized reports on tour's schedules, numbers, and costs
2. Communicate with other companies to coordinate sharing resources

17.1 Refreshing will grab the most up to date data from other companies

1. Automatically update changes in cruise line (nothing for front end)
2. Report changes instantly in your schedule to corresponding cruise lines (nothing in front end)
3. Be able to add employees shifts to the schedules based on tours

20.1 Button click from modal leads to generation of employee shift view

20.2 New shift views are visible to all employees

1. Individual reports for employees based on their duties

# 3. Features To Be Tested

|  |  |
| --- | --- |
| Front end | Backend |
| * *Users directed to login page (12.1)* * *Users directed to main page (12.2)* * *Read only buttons have limited access (13.1)* * *Resource data pulled from back end is displayed correctly (14.1, 14.2)* * *Resource filtering buttons (14.3)* * *Resource types can be added to tour types correctly (14.4)* * *Generate employee shift with button (22.1)* * *Shifts list is updated for all employees and manager (22.2)* * *Confirm that each different employee has different reports available (23)* | * *Verify login credentials (1.1)* * *Pass back validated token (1.2)* * *Validate token for successful login (1.3)* * *Expire token after fixed time (1.4)* * *Secure endpoints based on user roles (2.4)* * *Auto check for tour conflicts/resolutions (4.1)* * *Persist tour conflicts with flag in database (4.2)* * *Update persistent tour conflicts when resolved (4.3)* * *API tour crud operations (6.1)* * *API resource crud operations (6.2)* * *API company crud operations (7.1)* * *API cruise line crud operations (8.1)* * *Secure privileged accounts (10.3)* * *API user crud operations (10.4)* * *API privilege crud operations (10.5)* * *Limit account data based on permissions (11.1, 11.2)* |

# 

# 

# 

# 

# 

# 

# 

# 4. Features Not To Be Tested

Models and loading them with data from JSON objects or from the database. These are data classes with getters and setters and we are using well tested tools for handling them.

# 

# 

# 

# 

# 

# 

# 5. Approach

## 5.1 Component Testing

The approach for our component testing is pretty straight forward. Testing our methods that deal with the schedule checking are the highest priority. The speed and efficiency of these algorithms is what the component testing will be focused on for Schejewel. Checking all other functions that contain logic or some other processing will be next. These tests will also include any complicated SQL statements that are run on the database.

## 5.2 Integration Testing

Testing that will be done while integrating our front and backend components. Biggest potential problems with integration testing are some of the conversions of data (mainly time and grouping for the tours) into the database format and vice versa. Other potential problems are testing the AWS connections between the different databases and servers. We had quite a bit of trouble setting them up, so we plan on writing tests to make sure we know if they are causing problems again.

## 5.3 Security Testing

Testing that will be done to validate the security of our system. These tests will focus mainly on our API which processes all of our customers sensitive data. We will perform multiple tests to make sure our system is secure. Sensitive data includes tours schedules, business figures (such as how many people went on each tour), and keeping each company’s information private from other companies that use Schejewel. We will test this by using known protocols and procedures that are secure. This is also where code reviews will play a big role as it will allow another set of eyes to spot any security flaws.

## 5.4 Regression Testing

Testing that will occur throughout the development of our product. All tests must pass if a test that was previously passing fails we know that we have broken the code. Changing old test cases will need to be approved by the test team in order to make sure that they are in reality no longer valid tests. Pretty straight forward. Moving on.

**5.5 Environment Testing**

These tests will encompass testing on multiple browser environments and operating systems to ensure that our product is a comprehensive solution. Because Schejewel is a web application and the companies that we have been in contact thus far are very technologically unsound, we will need to make sure our webapp is supported for older browsers as well. This will be done mainly by hand at first, until we can make a more automated version for testing.

## 5.6 Beta Testing

Testing that will be done by the customer using a pre-release version of our product. This will help us validate and verify that our product meets the business needs or our customer. Our beta testing group will be the companies that are already interested in using Schejewel such as Experience Alaska Tours, Alaska Lodge Adventures, and Taquan Air.

# 

# 6. Testing Process

## 6.1 Test Deliverables

After our tests run we should have a postman collection of all api endpoint calls. This allows us to regression test to make sure that changes to the API have not broken other parts of the API. We should also have a Maven report for our J-Unit tests after all the tests are complete.

## 6.2 Testing Tasks Back End

**Verify login credentials (1.1)**

Test Objective:

Verify that the API checks login credentials and returns a token generated for that specific user. If invalid credentials an error is returned.

Technique:

Using postman we will hit the login endpoint with a username and password. We will then check the response to see if a token was returned in the X-AUTH-TOKEN response header.

Completion Criteria:

A token is returned and decoded to verify that it has the proper user credentials. If the login was invalid an error will be returned.

Special Considerations:

The API must be attached to a database with relevant test data. The token is returned in a base64 encoded format and needs to be decoded to verify.

**Pass back validated token (1.2)**

Test Objective:

Verify that a token was successfully created for a valid user.

Technique:

Using J-Unit we will create a test with a valid and invalid user. We will pass these users to the algorithm that creates tokens. We will then verify that the token was created correctly.

Completion Criteria:

A valid token is created for the valid user. An exception is thrown for the invalid user and no token is generated.

Special Considerations:

Tokens are base64 encoded and need to be decoded to verify their properties.

**Validate token for successful login (1.3)**

Test Objective:

After a token is issued if a user hits the endpoint again with the token the token should be validated and then data returned.

Technique:

Use J-Unit to pass a valid token the the validator and check if it returns valid. Use J-Unit to pass an invalid token to the validator and check if it returns invalid. Use postman to hit the endpoint with a valid and invalid token to try and get data back. If its valid data is returned else an exception is thrown.

Completion Criteria:

A valid or invalid response from the validate function. Data returned for a valid token and an exception for an invalid token.

Special Considerations:

The API must be attached to a database with relevant test data.

**Expire token after fixed time (1.4)**

Test Objective:

Check to see if a token expires after a set period of time after being issued.

Technique:

Use postman to log in a valid user and retrieve their token. After waiting the expiration time try to access data from the API with the token.

Completion Criteria:

The user should not be able to access data if the token has expired.

Special Considerations:

The API must be attached to a database with relevant test data.

**Secure endpoints based on user roles (2.4)**

Test Objective:

Check to make sure that restricted endpoints are not accessible by standard users.

Technique:

Use postman to log in as a valid user with no privileges. Then try to access a restricted endpoint. Repeat the process as a user with privileges.

Completion Criteria:

The user with no privileges should not be allowed to access data. The user with privileges should be allowed to access data.

Special Considerations:

The API must be attached to a database with relevant test data.

**Auto check for tour conflicts/resolutions (4.1)**

Test Objective:

Test to see if tour conflicts and resolutions are automatically flagged and reported to the user.

Technique:

Schedule two tours at the same time and see if the system is able to detect the conflict. Also update cruise ship times to cause conflicts and check if the system is able to detect the conflict.

Completion Criteria:

Checking the database we should see that the conflicting tours are flagged indicating that the system was able to detect the conflict. Resolutions to the conflict should also be present in the database.

Special Considerations:

The API must be attached to a database with relevant test data. A tester will have to manually check the database to make sure the test executed properly.

**Persist tour conflicts with flag in database (4.2)**

Test Objective:

Verify that our system is able to flag and persist conflicts in the database.

Technique:

Using J-Unit create a test that schedules a conflict and then saves the conflict in the database.

Completion Criteria:

The scheduling conflict should be flagged in the database indicating that the conflicts are persisting.

Special Considerations:

The API must be attached to a database with relevant test data.

**Update persistent tour conflicts when resolved (4.3)**

Test Objective:

Verify that our system is able to remove flags from conflicts marking them as resolved.

Technique:

Using J-Unit create a test that creates a conflict and then resolves the conflict saving its state in the database.

Completion Criteria:

The conflict that has been resolved should be flagged as resolved in the database.

Special Considerations:

The API must be attached to a database with relevant test data.

**API tour crud operations (6.1)**

Test Objective:

Verify that the API returns the correct information for crud operations for tours or errors out on bad information.

Technique:

Using postman, a browser plugin, which helps with making HTTP requests we will set the proper headers to reach the tour endpoint in our API. We will then check the response to see if the data was manipulated as expected. We will also send malformed data and expect exceptions to be returned.

Completion Criteria:

A tour JSON object is created, retrieved, updated, or deleted and returned as part of a HTTP response.

Special Considerations:

The API must be attached to a database with relevant test data.

**API resource crud operations (6.2)**

Test Objective:

Verify that the API returns the correct information for crud operations for resources or errors out on bad information.

Technique:

Using postman, a browser plugin, which helps with making HTTP requests we will set the proper headers to reach the resource endpoint in our API. We will then check the response to see if the data was manipulated as expected. We will also send malformed data and expect exceptions to be returned.

Completion Criteria:

A resource JSON object is created, retrieved, updated, or deleted and returned as part of a HTTP response.

Special Considerations:

The API must be attached to a database with relevant test data.

**API company crud operations (7.1)**

Test Objective:

Verify that the API returns the correct information for crud operations for companies or errors out on bad information.

Technique:

Using postman, a browser plugin, which helps with making HTTP requests we will set the proper headers to reach the company endpoint in our API. We will then check the response to see if the data was manipulated as expected. We will also send malformed data and expect exceptions to be returned.

Completion Criteria:

A company JSON object is created, retrieved, updated, or deleted and returned as part of a HTTP response.

Special Considerations:

The API must be attached to a database with relevant test data.

**API cruise line crud operations (8.1)**

Test Objective:

Verify that the API returns the correct information for crud operations for cruise lines or errors out on bad information.

Technique:

Using postman, a browser plugin, which helps with making HTTP requests we will set the proper headers to reach the cruise line endpoint in our API. We will then check the response to see if the data was manipulated as expected. We will also send malformed data and expect exceptions to be returned.

Completion Criteria:

A cruise line JSON object is created, retrieved, updated, or deleted and returned as part of a HTTP response.

Special Considerations:

The API must be attached to a database with relevant test data.

**Secure privileged accounts (10.3)**

Test Objective:

Check to make sure that accounts are only by their owners.

Technique:

Log in as a user, using postman try to access data from another users account.

Completion Criteria:

No data is returned to the user and an exception is thrown.

Special Considerations:

The API must be attached to a database with relevant test data.

**API user crud operations (10.4)**

Test Objective:

Verify that the API returns the correct information for crud operations for users or errors out on bad information.

Technique:

Using postman, a browser plugin, which helps with making HTTP requests we will set the proper headers to reach the user endpoint in our API. We will then check the response to see if the data was manipulated as expected. We will also send malformed data and expect exceptions to be returned.

Completion Criteria:

A user JSON object is created, retrieved, updated, or deleted and returned as part of a HTTP response.

Special Considerations:

The API must be attached to a database with relevant test data.

**API privilege crud operations (10.5)**

Test Objective:

Verify that the API returns the correct information for crud operations for privileges or errors out on bad information.

Technique:

Using postman, a browser plugin, which helps with making HTTP requests we will set the proper headers to reach the privilege endpoint in our API. We will then check the response to see if the data was manipulated as expected. We will also send malformed data and expect exceptions to be returned.

Completion Criteria:

A privilege JSON object is created, retrieved, updated, or deleted and returned as part of a HTTP response.

Special Considerations:

The API must be attached to a database with relevant test data.

**Limit account data based on permissions (11.1, 11.2)**

Test Objective:

Check to make sure that we do not allow users without permissions to access protected endpoints on our API.

Technique:

Log in as a user without permissions. Using postman try to access an endpoint that requires permissions.

Completion Criteria:

The user is not returned data and an exception is throw.

Special Considerations:

The API must be attached to a database with relevant test data.

## 6.3 Testing Tasks Front End

**Main page navigates correctly based on token validity (12.1, 12.2)**

Test Objective: Check to ensure that if a user accesses the web site without valid tokens that they are directed to the login page, and the splash page if they access the website with valid tokens.

Technique: Manually test the web app to ensure it passes the completion criteria.

Completion Criteria: Both cases are redirected correctly.

**Read only accounts view a prompt when accessing modifiers (13.1)**

Test Objective: With a read only account, try to access buttons that lead to our data modifying modal windows. The read only accounts should not be able to find the buttons in the DOM.

Technique: Manually test the web app to ensure it passes the completion criteria.

Completion Criteria: Read only accounts cannot access DOM elements that are used to modify data.

**Resource data pulled from back end is displayed correctly (14.1, 14.2)**

Test Objective: Confirm that all data that should be visible to users is in the DOM correctly for resource view.

Technique: Manually test the web app to ensure it passes the completion criteria.

Completion Criteria: Using a specific date from the database with known resource values, check that all pieces of information from the database entry that should be visible for the resource view is visible.

**Resource filtering buttons(14.3)**

Test Objective: Confirm that DOM elements that aren’t part of the selected tour or owner are hidden or not highlighted.

Technique: Manually test the web app to ensure it passes the completion criteria.

Completion Criteria: Using a specific date from the database with known resource values, owners and tours, click on at least one specific owner and tour. Check the DOM to ensure that elements that are known to not be of that owner or tour are not visible or are hidden.

**Resource types can be added to tour types correctly (14.4)**

Test Objective: When a resource needs to be added to an existing tour type, the end user can add this resource through a modal window, and that change is saved and perpetuated from then on.

Technique: Manually test the web app to ensure it passes the completion criteria.

Completion Criteria: Use DOM manipulation to access modal for adding resource to tour type, hit button, and then refresh resource view. Check incoming database information to ensure that change in database is returning correctly.

**Generate employee shift with button (22.1)**

Test Objective: Confirm that by clicking on the generate employee button, that a new shift appears for that employee.

Technique: Manually test the web app to ensure it passes the completion criteria.

Completion Criteria: After clicking generate shift button, navigate to employee shifts, check DOM to ensure that a shift div exists for the name of the employee added. If not, then reject this test.

**Shifts list is updated for all employees and manager (22.2)**

Test Objective: Confirm that once a new shift is added to the employee shifts, everyone can see the new shifts immediately.

Technique: Manually test the web app to ensure it passes the completion criteria.

Completion Criteria: After the generation of a new shift is complete (22.1), access the shifts page with the manager, the employee who was added, and a different employee. All three cases will confirm that the DOM has the new shift added. If not, reject this test.

**Confirm that each different employee has different reports available (23)**

Test Objective: Confirm that different privilege level users see different reports.

Technique: Manually test the web app to ensure it passes the completion criteria.

Completion Criteria: View the dashboards page with a manager and a regular employee. Compare DOM values to confirm that managers have access to manager reports, while regular employees have access to employee reports. If the manager doesn’t have access to the manager reports, or the employee has access to manager reports, reject this test case.

## 6.4 Responsibilities

Guy Warner the testing manager is responsible for all the tests concerning our product. Each developer is responsible for testing the code they write to verify its correctness. Each developer should report to Guy after the initial testing of their code.

## 6.5 Resources

Maven: The maven build tool is being used to build our project and run all unit tests. If a test fails then the build is rejected. This ensures that new code added has not broken other parts of our project.

Postman: A tool used to send HTTP requests in order to test API’s.

J-Unit: A tool used for unit testing code.

User: Users will be used to test frontend functionality.

## 6.6 Schedule

3/20/2015 Unit tests (Database, API)

4/1/2015 Integration tests (Frontend, API, Database)

4/14/2015 Usability testing

5/1/2015 Alpha testing

7/1/2015 Beta testing

# 

# 

# 7. Environmental Requirements

## 7.1 Software

We will need to run usability tests all all major browsers. (Chrome, Firefox, and Explorer). This will insure that our program runs in any environment given to it.

## 7.2 Tools

Maven: The maven build tool is being used to build our project and run all unit tests. If a test fails then the build is rejected. This ensures that new code added has not broken other parts of our project.

Postman: A tool used to send HTTP requests in order to test API’s.

J-Unit: A tool used for unit testing code.

## 7.3 Risks and Assumptions

We assume that there is valid data in our test database allowing us to run our tests.