AdventureAudit Release Summary

Instruction: If any of the following sections are well documented, please attach the URL and have a brief description on it. No need to repeat.

Team Members

Name and email	GitHub id	Role of each member and tasks done by each member (brief description)		
Taeho Choi choit1@myumanitoba.ca	Taehoya	Developer	Task	PR
Suhjin Kang kangs2@myumanitoba.ca	Skang9810	Developer	Task	PR
Leonardo Warsito warsitol@myumanitoba.ca	Leonw00	Developer, QA Analyst	Task	PR
Luke Lepa lepal@myumanitoba.ca	Luke-Lepa	Developer	Task	PR

Project Summary

1) Elevator pitch description at a high-level.

AdventureAudit is a travel-focused expense tracking web application that aims to provide users with a quick and easy-to-use tool to document the transactions of their trips. AdventureAudit can document trip expenses individually or with other travellers as a group. (+ more)

2) Highlight the differences between the final version and proposal if applicable

Initially, we had planned to allow users to input transactions in any currency for our Currency Conversion feature. However, we decided to maintain consistency in the database and display, so we modified it to only use the currency of the destination for transactions. As a result, we added a live currency conversion display for the user's home currency, allowing them to reference and understand costs in a more familiar currency. We also considered cases where the destination and the user's home currencies are the same. In this case, the live conversion display is removed for simplicity. Additionally, we accounted for currency accuracy, such as currencies that do not have decimals (according to ISO 4217), among other neat features.

GitHub Repository Link

- https://github.com/Taehoya/Adventure-Audit

Docker Hub Repository Link

The docker image should include everything to run your application. Once the docker image is run, no additional actions are required to launch the application.
1) provide the DockerHub link for your image; 2) provide the instructions to run your docker image(s).

- **Server (261.61MB):**
  1) Docker Hub link
     https://hub.docker.com/repository/docker/taehoya/adventure_audit_server/general
  2) Instruction to run docker images:

```
docker run -p 8000:8000 taehoya/adventure_audit_server:0.0.10
```

For M1 users,

```
docker run -p 8000:8000 --platform linux/amd64
taehoya/adventure_audit_server:0.0.10
```

- **Client (1.45 GB):**
  1) Docker Hub link
     https://hub.docker.com/repository/docker/taehoya/adventure_audit_client/general

  2) Instruction to run docker images:

```
docker run -p 3000:3000 taehoya/adventure_audit_client:0.0.10
```

For M1 users,

```
docker run -p 3000:3000 --platform linux/amd64
taehoya/adventure_audit_client:0.0.10
```

Note: Client image takes about 5-6 minutes to download and install. Please wait patiently until you see this message in the terminal.

```
You can now view client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.17.0.2:3000

Note that the development build is not optimized.
To create a production build, use yarn build.

webpack compiled successfully
Compiling...
Compiled successfully!
webpack compiled successfully
```

List of user stories for each sprint

Make sure the user story is a clickable link to the milestone/issue on GitHub

Sprint 1

- No user story

Sprint 2

- US #1 : As a traveller, I want to be able to view a list of all my trips so that I can keep track of all my trips [Status: Done]
- US #2 : As a traveller, I want to be able to categorize my transactions so that I can get a better understanding of how I'm spending my money [Status: Done]
- US #3 : As a traveller, I want to be able to see how much the entire trip cost so that I can learn about how much it took to travel [Status: Done]
- US #4 :As a traveller, I want to be able to enter my budget for a trip so that I know how much I can spend and not overspend [Status: Done]
- US #5 : As a traveller, I want to be able to view all my transactions of a trip, so that I can view a record of my spending [Status: Done]
- US #6 : As a traveller, I want to be able to add transactions to a trip so that I can record what I have spent during the trip [Status: Done]
- US #7 : As a traveller, I want to be able to create a trip so that I can record my transactions [Status: Done]
- US #8 : As a traveller using AdventureAudit, I want to delete the trip information so that I can remove the trip event that I no longer want to keep a record of. [Status: Done]

Sprint 3

- US #9: As a traveller, I want to be able to add custom categories so that I can personalize my transactions to have a more detailed record [Status: Done]
- US #10: As a traveller, I want to be able to record transactions in different currencies to make documenting foreign purchases more accurate [Status: Done]
- US #11: As a traveller, I want to be able to invite other users to trips so that I can document transactions with others [Status: Done]
- US #12: As a transaction payer, when I add a transaction, I want to be able to select the other users that I paid for, so that we can divide the cost later. [Status: Done]
- US #13: As a traveller, I want to be able to view my transactions in a pie chart by category so that I can see where I spent the most money. [Status: Done]
- US #14: As a traveller, I want to be able to see my top 5 most expensive transactions so I can see what activities/services cost the most during a trip. [Status: Done]
- US #15: As a group traveller, I want to see how much money I owe to others and who owes me money, so that everyone's debts are documented [Status: Done]

Sprint 4

- No User Stories

User manual

Instructions on how to run the application for each core feature.

1. Log in with your Google account.
2. Your home country that determines the home currency is set to Canada as a default. You can change your home country in the profile setting if you want to see the currencies converted to another currency.

<Core feature 1: Trip Expense / Payment tracking>

3. Add a new trip using the "+ New Trip" button. A new trip button should show up on the left sidebar.
4. Clicking on a trip will open details about the trip, such as its name and the transactions associated with the trip.
5. For each trip, the right sidebar has four functionalities accessed through the four buttons: transaction adder, custom category adder, report, and collaboration tab. It also has an additional functionality serving as a transaction viewer where the details of each transaction are shown when clicking on a transaction.
6. You can edit the trip information (e.g. trip title, date, etc) at any time using the edit trip button.
7. You can also delete a trip with the delete button within the edit trip window.
8. Create a transaction by completing the form in the transaction adder.
9. You can delete transactions by selecting them from the list of transactions and clicking the delete button on the right side of the screen.
10. Create a custom category by completing the form in the custom category adder.
11. You can delete categories in the transaction adder screen by turning on the edit mode switch, which will reveal the delete buttons on each category icon.

<Core feature 2: Group Trip Sharing / Collaboration>

12. Invite another member to the trip in the collaboration tab by typing in the email address of the user. This user must be an existing user of AdventureAudit.
13. The user that gets invited will receive an invitation in their profile window, which they have the option to accept or decline. Each invitation will show you from whom the invitation was from and the title of the trip.
14. Accepting the invitation will allow both users to add transactions and edit properties of the trip.
15. Add a transaction within the shared trip. The newly added transaction should show up on the other member's screen as well.
    a. The transaction adder for a trip with more than one user has two additional fields, which are used to choose who paid (required) and check who was part of the transaction (optional).
16. Delete or leave the trip. If there are multiple members part of a trip, only the creator of the trip can delete the trip. Other members of the trip event can leave the trip if they wish to.

<Core feature 3: Trip Summary Report>

17. Once you have at least one transaction on your trip, the report tab will show you the expense summary of your trip.

18. The pie chart shows how much you have spent on each category.
    a. Clicking on each slice of the pie chart will show the total amount of money spent for that category in the middle of the pie chart.
19. The top 5 transaction list displays the 5 most expensive transactions of the trip in descending order.
20. Debt relationship shows who owes money to you and who you owe money to. This will be blank if you are the only member of the trip.

Overall Arch and Design

Provide the link to the overall arch and class diagram on GitHub.

- **Overall architecture:** https://github.com/Taehoya/Adventure-Audit/blob/main/docs/AdventureAudit_Architecture_lock_diagram.png
- **Class diagram:** https://github.com/Taehoya/Adventure-Audit/blob/main/docs/AdventureAuidt_Class_Diagram.png

Infrastructure

For each library, framework, database, tool, etc
Name and link: 1 paragraph description of why you are using this framework, not other alternatives and why you didn't choose them.

- **Frontend**
    - React (https://react.dev/)
        - We used React mainly because it allows easy code reusability, making our UI development more efficient. In addition, everyone in our team was familiar with JavaScript, which made the process of learning React easier compared to other languages. React's benefits are the Virtual DOM and unidirectional data flow.
    - Chakra-UI (https://chakra-ui.com)
        - We used Chakra-UI because it provided us with multiple pre-built components that are common on modern websites. This allowed us to focus on building our core functional features instead of building these components from scratch, which can take quite some time to do.
- **Backend**
    - Node.js (https://nodejs.org/en)
        - We decided to use Node.js because we all know JavaScript and it allows us to have a unified codebase language for the client (react) and server (node.js). Additionally, Node.js has a large community of developers compared to other server managers, so it was easy to find the resources.
    - Express.js (https://expressjs.com/)
        - We decided to use Express.js because it provides us with lots of handy tools such as handling routes and managing HTTP requests and responses without obscuring Node.js features.
- **Database**
    - MySQL (https://www.mysql.com/)

  - We decided to use MySQL because there is a lot of relation and filtering for our project. We were also all familiar with SQL syntax. We also picked mySQL due to its large community and popularity.

- **Tool**
  - GitHub Actions (https://github.com/features/actions)
    - We selected GitHub actions for CI/CD pipeline because it was easy to integrate with the repository and other GitHub features such as Pull requests, Releases and GitHub secrets.

Name Conventions

List your naming conventions or just provide a link to the standard ones used online.

- Frontend
  - https://github.com/Taehoya/Adventure-Audit/blob/main/docs/Styling%20Guide/frontend.md
- Backend
  - https://github.com/Taehoya/Adventure-Audit/blob/main/docs/Styling%20Guide/backend.md
  - camelCase (Link)
  - eslint-config-airbnb (Link)

Code

Key files: top **5** most important files (full path). We will also be randomly checking the code quality of files. Please let us know if there are parts of the system that are stubs or are a prototype so we grade these accordingly.

File path with a clickable GitHub link	**Purpose (1 line description)**
Main Page (client/src/pages/MainPage.js)	Root component of the frontend React environment. Stores “global variables” and passes them to child components along with functions to update these variables.
Transaction Adder (client/src/components/TransactionAdder.js)	Allows the user to add transactions to a selected trip
Transaction Model (server/src/models/transaction.js)	Facilitates database operations to interact with the transaction table
Trip Model (server/src/models/trip.js)	Facilitates database operations to interact with the trip table
Report Model (server/src/models/report.js)	Contains various functions that generate statistical summaries of a trip

Continuous Integration and deployment (CI/CD)

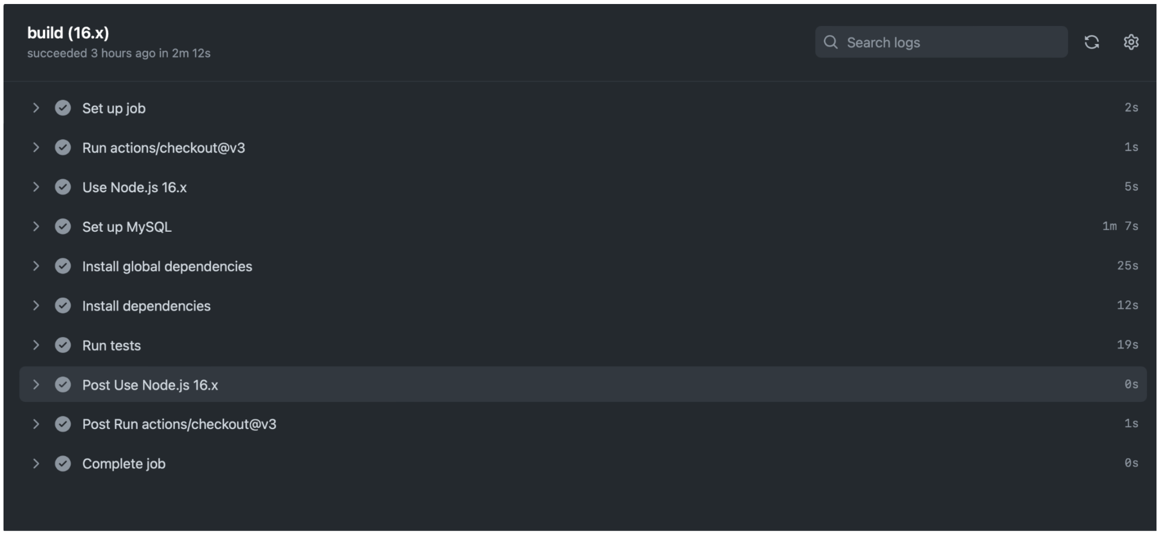
1) Describe your CI/CD environment and the clickable link to your CI/CD pipeline.

We used GitHub action for our CI/CD pipeline. All the files are found here .github/workflows in the directory of the repository.

- CI Pipeline
  - https://github.com/Taehoya/Adventure-Audit/blob/main/.github/workflows/node.js.yml
  - https://github.com/Taehoya/Adventure-Audit/blob/main/.github/workflows/codeql.yml
- CD Pipeline
  - https://github.com/Taehoya/Adventure-Audit/blob/main/.github/workflows/docker-deploy-client.yml
  - https://github.com/Taehoya/Adventure-Audit/blob/main/.github/workflows/docker-deploy-server.yml
- Security Scanning (CodeQL)
  - https://github.com/Taehoya/Adventure-Audit/blob/main/.github/workflows/codeql.yml

2) Snapshots of the CI/CD execution. Provide one for CI and one for CD to demo your have successfully set up the environment.

- CI Pipeline

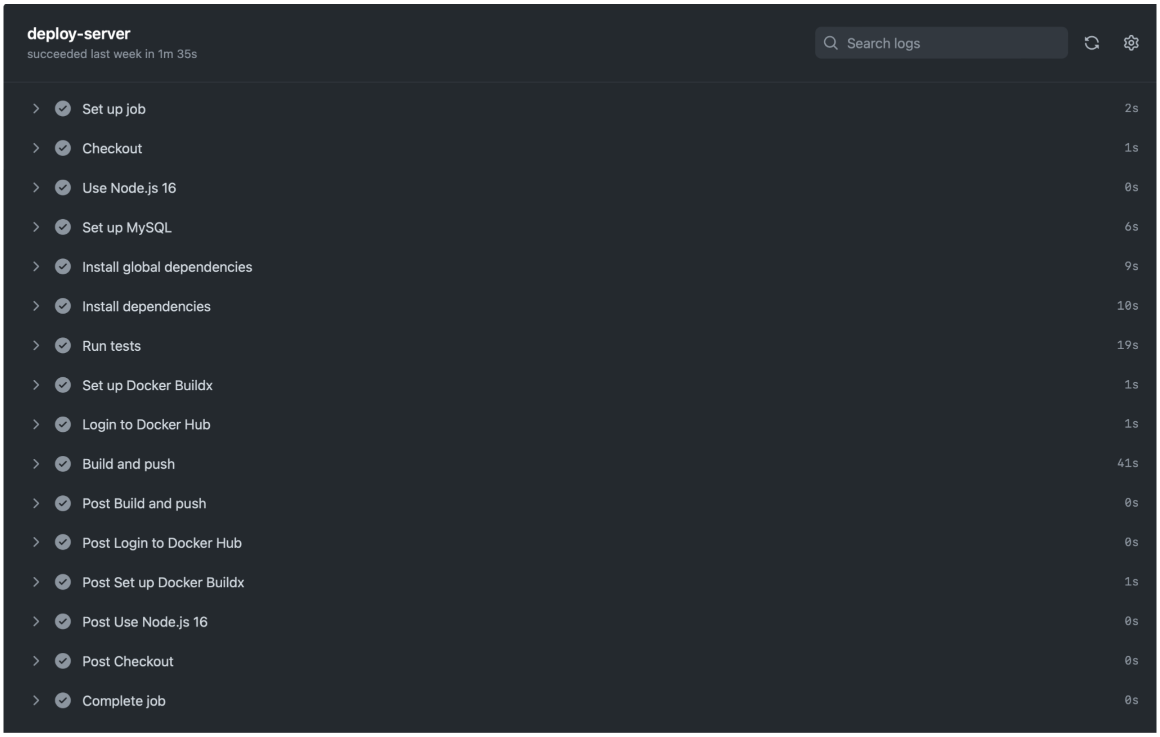


build (16.x)

succeeded 3 hours ago in 2m 12s

```
Set up job                          2s
Run actions/checkout@v3             1s
Use Node.js 16.x                    5s
Set up MySQL                        1m 7s
Install global dependencies         25s
Install dependencies                12s
Run tests                           19s
Post Use Node.js 16.x               0s
Post Run actions/checkout@v3        1s
Complete job                        0s
```

- CD Pipeline



```
deploy-server
succeeded last week in 1m 35s

Set up job                      2s
Checkout                        1s
Use Node.js 16                  0s
Set up MySQL                    6s
Install global dependencies     9s
Install dependencies            10s
Run tests                       19s
Set up Docker Buildx            1s
Login to Docker Hub             1s
Build and push                  41s
Post Build and push             0s
Post Login to Docker Hub        0s
Post Set up Docker Buildx       1s
Post Use Node.js 16             0s
Post Checkout                   0s
Complete job                    0s
```

Testing

Link to testing plan

- https://github.com/Taehoya/Adventure-Audit/blob/main/docs/AdventureAudit_Test_Plan.docx.pdf

Unit/integration/acceptance test

Each story needs a test before it is complete. In other words, the code coverage (in terms of statements) should be 100%. If some classes/methods are missing unit tests, please describe why and how you are checking their quality. Please describe any unusual/unique aspects of your testing approach.

List the **10** most important unit tests with links below (if there are more than one unit tests in one test file, indicate clearly).

Test File path with clickable GitHub link	What is it testing (1 line description)
Report Model tests (server/__test__/model/report.test.js)	Test report model function such as 1. Retrieve the total expense of a trip 2. Retrieve the top 5 transactions 3. Retrieve much was spent on each category 4. Retrieve how much money the user

	owes to others, and who owes the user money. Check whether the user gets the expected data from the database.
Trip Model tests (server/__test__/model/trip.test.js)	Test trip model function (SQL query). 1. Retrieve trip by start date, end date 2. Insert trip 3. Update trip 4. Delete trip Check whether the user gets the expected data from the database.
Category Model tests (server/__test__/model/category.test.js)	Test category model function (SQL query). 1. Retrieve category 2. Insert category 3. Delete category Check whether the user gets the expected data from the database.
Transaction Model tests (server/__test__/model/transaction.test.js)	Test transaction model function (SQL query). 1. Retrieve transaction 2. Insert transaction 3. Delete transaction Check whether the user gets the expected data from the database.
User Model tests (server/__test__/model/user.test.js)	Test user model function (SQL query). 1. Retrieve user 2. Insert user 3. Update user Check whether the user gets the expected data from the database.
Trip Controller tests (server/__test__/controllers/unit/trip.test.js)	Test trip endpoint API using stub data 1. Retrieve trip 2. Create trip 3. Update trip 4. Delete trip and check whether the user gets the expected response.
Category Controller tests (server/__test__/controllers/unit/category.test.js)	Test category endpoint API using stub data 1. Retrieve category 2. Create category

	and check whether the user gets the expected response.
Transaction Controller tests (server/__test__/controllers/unit/transaction.test.js)	Test transaction endpoint API using stub data 1. Retrieve transaction 2. Create transaction 3. Delete transaction and check whether the user gets the expected response.
Invite Controller tests (server/__test__/controllers/unit/invite.test.js)	Test invite endpoint API using stub data 1. Create invite 2. Delete invite and check whether the user gets the expected response.
Group Controller tests (server/__test__/controllers/unit/group.test.js)	Test group endpoint API using stub data 1. Create group 2. Delete group and check whether the user gets the expected response.

List the **5** most important integration tests with links below (if there are more than one unit tests in one test file, indicate clearly).

Test File path with clickable GitHub link	**What is it testing (1 line description)**
Trip tests (server/__test__/controllers/integration/trip.test.js)	Test trip endpoint API such as 1. Get trip 2. Create trip 3. Update trip 4. Delete trip and check whether the user gets the expected response and data from the database.
Transaction tests (server/__test__/controllers/integration/transaction.test.js)	Test transaction endpoint API such as 1. Get transaction 2. Create transaction 3. Update transaction 4. Delete transaction and check whether the user gets the expected response and data from the database.

Category tests (server/__test__/controllers/integration/category.test.js)	Test category endpoint API such as 1. Get category 2. Create category 3. Delete category and check whether the user gets the expected response and data from the database.
User tests (server/__test__/controllers/integration/user.test.js)	Test user endpoint API such as 1. Get user 2. Create user 3. Update user and check whether the user gets the expected response and data from the database.
Report tests (server/__test__/controllers/integration/report.test.js)	Test report endpoint API such as 1. Total expense 2. Top 5 transactions 3. How much user spends for each category 4. Debt relation and check whether the user gets the expected response and data from the database.

List the **5** most important acceptance tests with links below. If your acceptance tests are done manually, you should have detailed steps on how to run the test and the expected outcome for the test.

Test File path (if you automated the test) or as comments in GitHub issues (if it's with clickable GitHub link	**Which user story is it testing**
Steps: 04-acceptance-test.md Video: User-Story 4	User story 4: As a traveller, I want to be able to add transactions to a trip so that I can record what I have spent during the trip.
Steps: 09-acceptance-test.md Video: User-Story 9	User story 9: As a traveller, I want to be able to record transactions in different currencies to make documenting foreign purchases more accurate
Steps: 10-acceptance-test.md Video: User-Story 10	User story 10: As a traveller, I want to be able to add custom categories so that I can personalize my transactions to have a more detailed record

Steps: 11-acceptance-test.md Video: User-Story 11	User story 11: As a traveller, I want to be able to invite other users to trips so that I can document transactions with others
Steps: 12-acceptance-test.md Video: User-Story 12	User story 12: As a transaction payer, when I add a transaction, I want to be able to select the other users that I paid for, so that we can divide the cost later

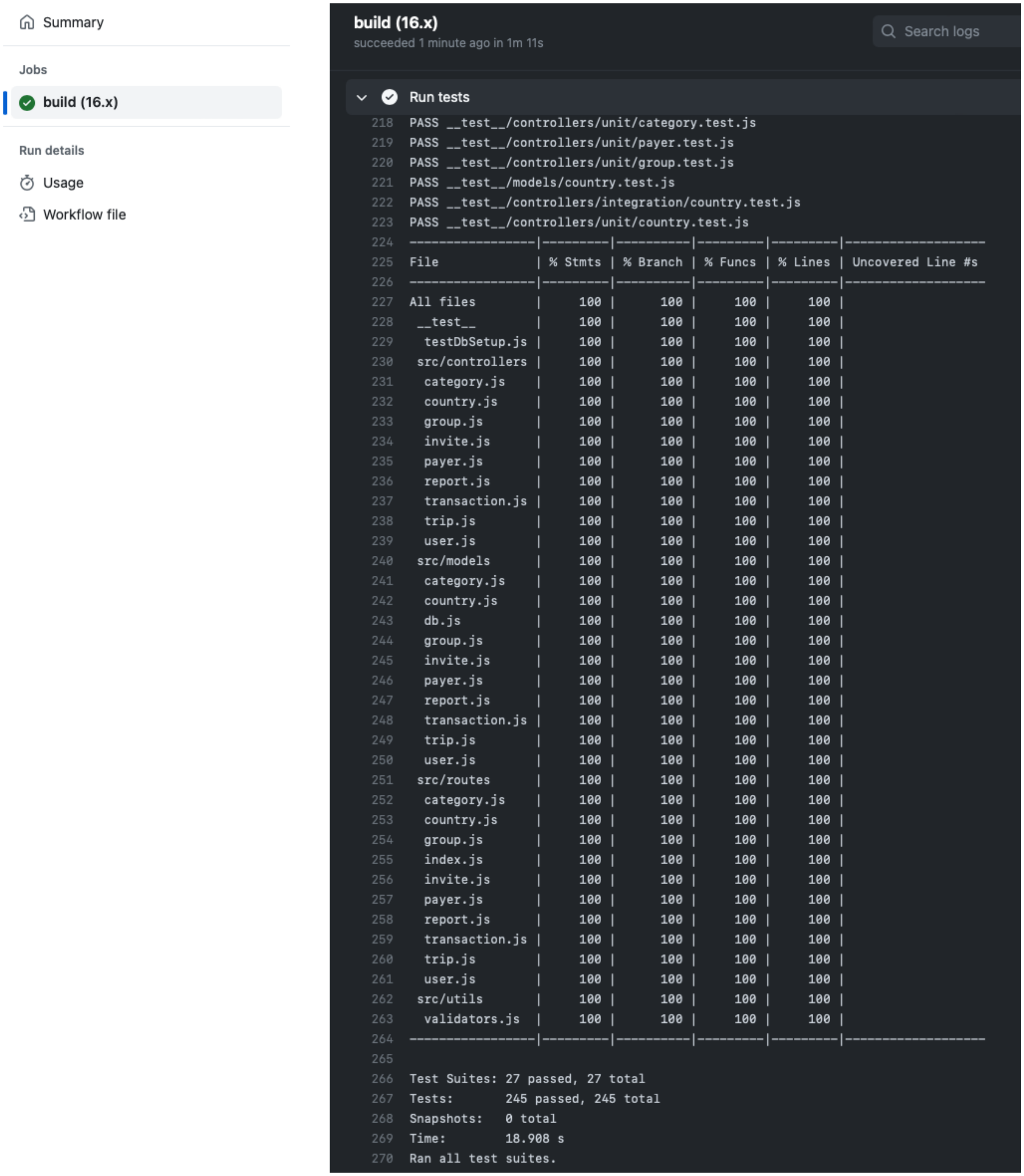
Regression testing

1) Describe how you run the regression testing (e.g., which tests are executed for regression testing and which tool is used?). 2) Provide the link to the regression testing script and provide the last snapshot of the execution and results of regression testing.

1) We run regression testing by using the CI pipeline (GitHub Action). We executed unit testing, and integration testing for our regression tests.

2) Regression test scripts:
https://github.com/Taehoya/Adventure-Audit/blob/main/.github/workflows/node.js.yml

   Below is the last snapshot of the execution. (Link)



```
PASS __test__/controllers/unit/category.test.js
PASS __test__/controllers/unit/payer.test.js
PASS __test__/controllers/unit/group.test.js
PASS __test__/models/country.test.js
PASS __test__/controllers/integration/country.test.js
PASS __test__/controllers/unit/country.test.js
-----------------|---------|----------|---------|---------|-------------------
File             | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----------------|---------|----------|---------|---------|-------------------
All files        |     100 |      100 |     100 |     100 |
 __test__        |     100 |      100 |     100 |     100 |
  testDbSetup.js |     100 |      100 |     100 |     100 |
 src/controllers |     100 |      100 |     100 |     100 |
  category.js    |     100 |      100 |     100 |     100 |
  country.js     |     100 |      100 |     100 |     100 |
  group.js       |     100 |      100 |     100 |     100 |
  invite.js      |     100 |      100 |     100 |     100 |
  payer.js       |     100 |      100 |     100 |     100 |
  report.js      |     100 |      100 |     100 |     100 |
  transaction.js |     100 |      100 |     100 |     100 |
  trip.js        |     100 |      100 |     100 |     100 |
  user.js        |     100 |      100 |     100 |     100 |
 src/models      |     100 |      100 |     100 |     100 |
  category.js    |     100 |      100 |     100 |     100 |
  country.js     |     100 |      100 |     100 |     100 |
  db.js          |     100 |      100 |     100 |     100 |
  group.js       |     100 |      100 |     100 |     100 |
  invite.js      |     100 |      100 |     100 |     100 |
  payer.js       |     100 |      100 |     100 |     100 |
  report.js      |     100 |      100 |     100 |     100 |
  transaction.js |     100 |      100 |     100 |     100 |
  trip.js        |     100 |      100 |     100 |     100 |
  user.js        |     100 |      100 |     100 |     100 |
 src/routes      |     100 |      100 |     100 |     100 |
  category.js    |     100 |      100 |     100 |     100 |
  country.js     |     100 |      100 |     100 |     100 |
  group.js       |     100 |      100 |     100 |     100 |
  index.js       |     100 |      100 |     100 |     100 |
  invite.js      |     100 |      100 |     100 |     100 |
  payer.js       |     100 |      100 |     100 |     100 |
  report.js      |     100 |      100 |     100 |     100 |
  transaction.js |     100 |      100 |     100 |     100 |
  trip.js        |     100 |      100 |     100 |     100 |
  user.js        |     100 |      100 |     100 |     100 |
 src/utils       |     100 |      100 |     100 |     100 |
  validators.js  |     100 |      100 |     100 |     100 |
-----------------|---------|----------|---------|---------|-------------------

Test Suites: 27 passed, 27 total
Tests:       245 passed, 245 total
Snapshots:   0 total
Time:        18.908 s
Ran all test suites.
```

Load testing

1) Describe the environment for load testing, such as tools, load test cases.

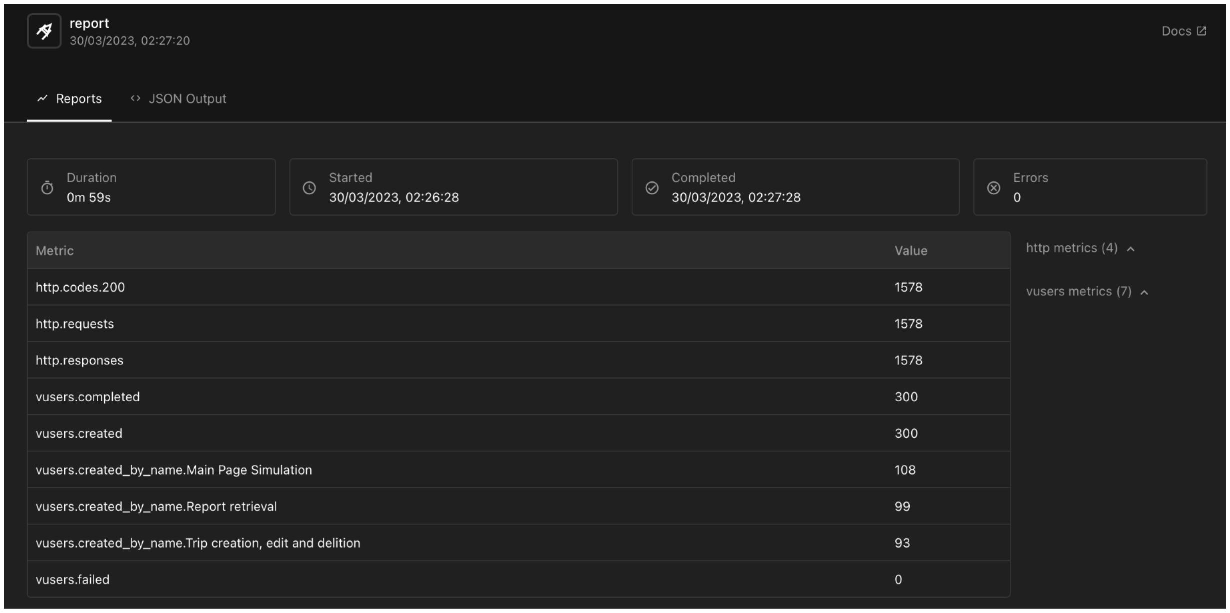
We used a tool called artillery to do our load testing. Our load test is configured using a .yml file [here] called load-test.yml and can be run with `npm run load-test` from our server folder (this requires that artillery has been globally installed on your device).

For our environment, we are using a local machine with a production environment from the Docker hub. Our load test runs for 60 seconds with 5 users created every second. As a result, we have around 300 users making requests to our system concurrently. The test cases consist of real-life scenarios of how users would use our application. It contains scenarios such as creating, removing and modifying the trip, login as a user to the application, navigating through our homepage and looking through the reports just to name a few.

2) Provide the test report for load testing.

Once we have run our load test, it will display the test result on the console and generate a json file which we can then turn into an html file to have a visual representation of the testing result.

Below is the visual representation of our load test report. Figure `Load-Test 1` shows the metric results of our load test such as the total duration, number of requests and users created. It also shows if there are any failed users to be created which there is none for our test. There are also bar graphs that show the request amount every 10 seconds such as figure `Load-Test 2` and more graphs on the html report.



report
30/03/2023, 02:27:20

Duration
0m 59s

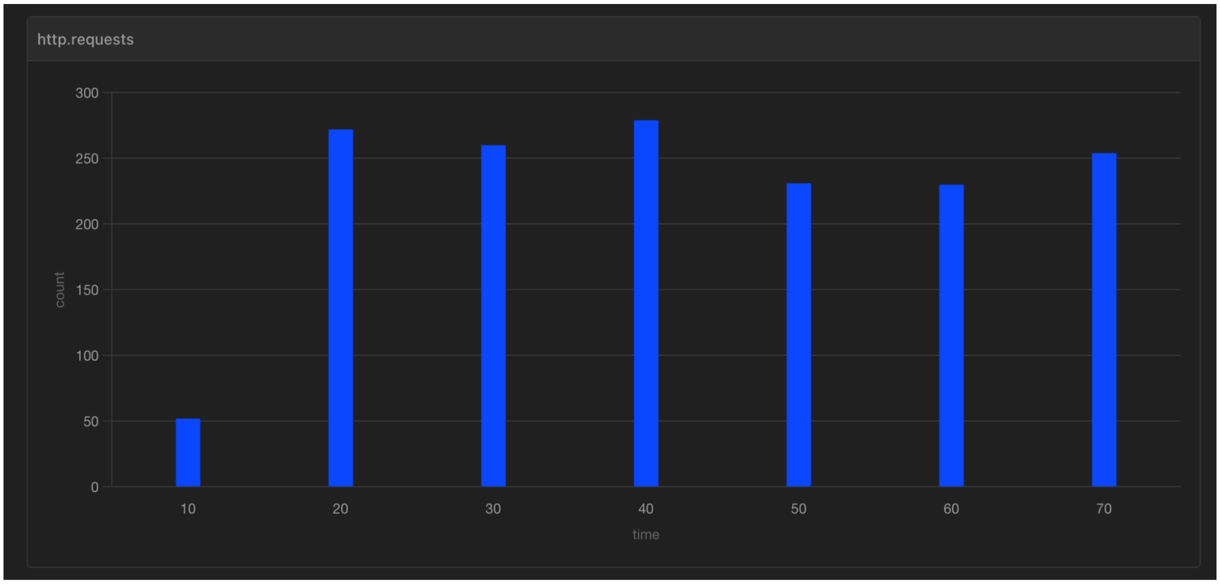
Started
30/03/2023, 02:26:28

Completed
30/03/2023, 02:27:28

Errors
0

Metric	Value
http.codes.200	1578
http.requests	1578
http.responses	1578
vusers.completed	300
vusers.created	300
vusers.created_by_name.Main Page Simulation	108
vusers.created_by_name.Report retrieval	99
vusers.created_by_name.Trip creation, edit and delition	93
vusers.failed	0

Load-Test 1



Load-Test 2

3) Discuss one bottleneck found in the load testing.

The load test we did so far has been rather smooth in execution and result. But, one bottleneck that might impact our load testing is for handling a large number of requests, concurrently. Given that our goal is to handle approximately 1000 requests by 100 users, we have achieved this goal as our load tests can handle around 1500 requests for 300 users. But if we were to handle a way larger number of requests and users, there can be timeout errors that happen. It may impact the accuracy and reliability of the load test results. This may be caused by many impacting variables from network bandwidth to our testing environment. In addition, our current hardware setup isn't optimal for high-intensity load testing which requires tests of many requests and users. As such, in the future, we can try to resolve this issue by splitting the load test so that one system doesn't need to handle all the users and requests. We can improve our resource allocation by scheduling and parallelization. In addition, revising the locking strategy and finding a better synchronization technique may be necessary.

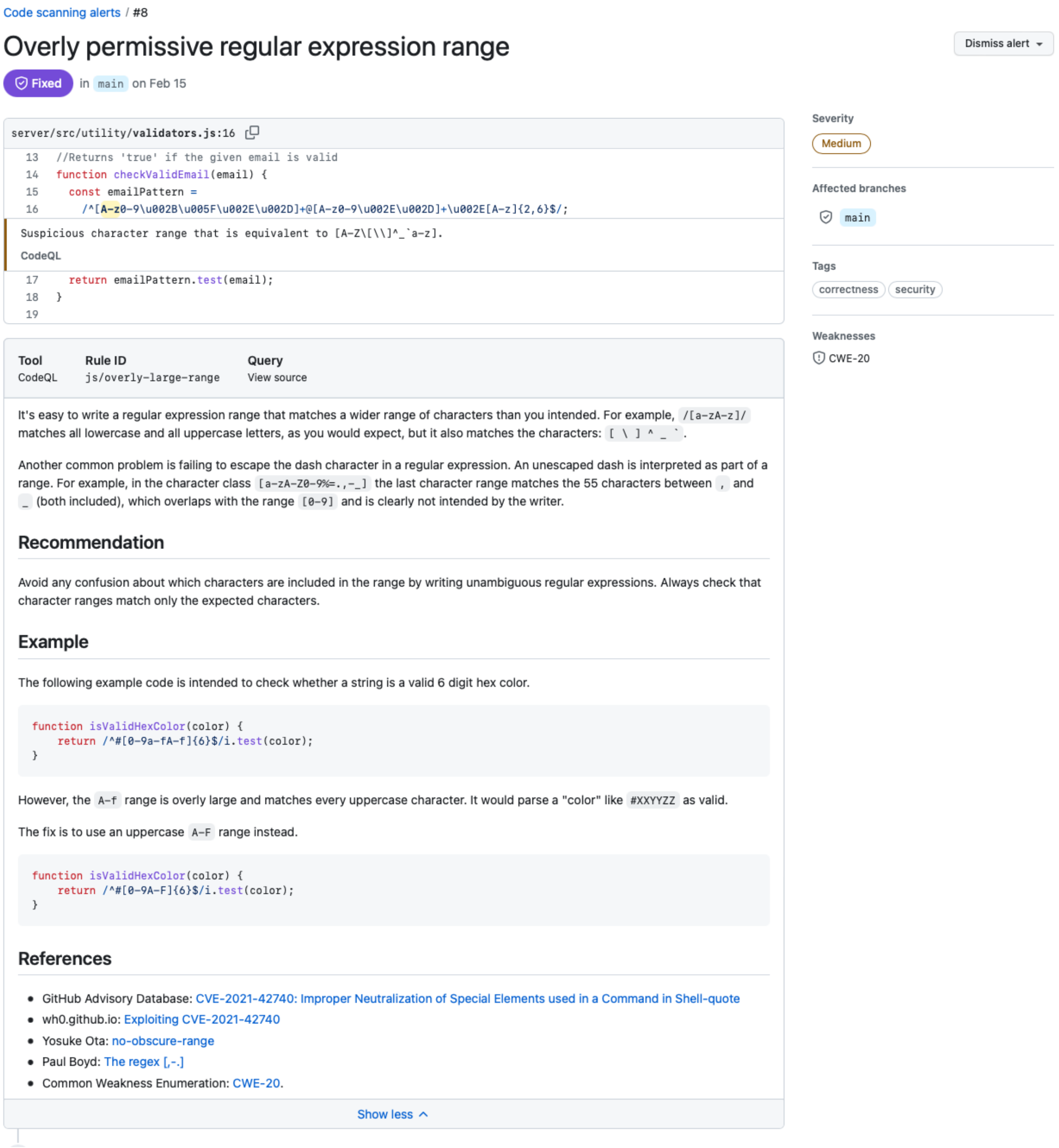
Security analysis

1) Describe the choice of the security analysis tool and how you run it. The security analysis tool should analyze the language that is used in the majority of your source code.

We use CodeQL as a security analysis tool. CodeQL scanning is a static code analysis technique to identify security vulnerabilities and bugs in software code, and it is developed by GitHub. It supports multiple programming languages, such as JavaScript and Go. We decided to use CodeQL as a security analysis tool because it was easy to integrate with our GitHub repository and supports JavaScript, our source code language. CodeQL scanning runs when a developer creates Pull Request and regularly runs at “29 11 * * 4” (Every Thursday at 11:29 PM). You can see the workflow (.yaml) file in the .github/workflows/codeql.yml and can see the reports on the Security>Code scanning page.

2) Attach a report as an appendix below from static analysis tools by running the security analysis tool on your source code. Randomly select 5 detected problems and discuss what you see. Note that you are not required to fix the alarms (bugs and vulnerabilities) in the course.

https://github.com/Taehoya/Adventure-Audit/security/code-scanning/8



Overly permissive regular expression range

Fixed in `main` on Feb 15

server/src/utility/validators.js:16

```
//Returns 'true' if the given email is valid
function checkValidEmail(email) {
  const emailPattern =
    /^[A-z0-9\u002B\u005F\u002E\u002D]+@[A-z0-9\u002E\u002D]+\u002E[A-z]{2,6}$/;
```

```
Suspicious character range that is equivalent to [A-Z\[\\]^_`a-z].
```

CodeQL

```
  return emailPattern.test(email);
}
```

Tool	Rule ID	Query
CodeQL	js/overly-large-range	View source

It's easy to write a regular expression range that matches a wider range of characters than you intended. For example, `/[a-zA-z]/` matches all lowercase and all uppercase letters, as you would expect, but it also matches the characters: `` [ \ ] ^ _ ` ``.

Another common problem is failing to escape the dash character in a regular expression. An unescaped dash is interpreted as part of a range. For example, in the character class `[a-zA-Z0-9%=.,-_]` the last character range matches the 55 characters between `,` and `_` (both included), which overlaps with the range `[0-9]` and is clearly not intended by the writer.

Recommendation

Avoid any confusion about which characters are included in the range by writing unambiguous regular expressions. Always check that character ranges match only the expected characters.

Example

The following example code is intended to check whether a string is a valid 6 digit hex color.

```
function isValidHexColor(color) {
    return /^#[0-9a-fA-f]{6}$/i.test(color);
}
```

However, the `A-f` range is overly large and matches every uppercase character. It would parse a "color" like `#XXYYZZ` as valid.

The fix is to use an uppercase `A-F` range instead.

```
function isValidHexColor(color) {
    return /^#[0-9A-F]{6}$/i.test(color);
}
```

References

- GitHub Advisory Database: CVE-2021-42740: Improper Neutralization of Special Elements used in a Command in Shell-quote
- wh0.github.io: Exploiting CVE-2021-42740
- Yosuke Ota: no-obscure-range
- Paul Boyd: The regex [,-.]
- Common Weakness Enumeration: CWE-20.

Severity
Medium

Affected branches
main

Tags
correctness security

Weaknesses
CWE-20

We have a total of 9 security vulnerabilities that are detected by the CodeQL scanning tool. The report is about the regular expression we used to validate email format, which is the “overly permissive regular expression range”.

```
const emailPattern =
/^[A-z0-9\u002B\u005F\u002E\u002D]+@[A-z0-9\u002E\u002D]+\u002E[A-z]{
2,6}$/;
```

An overly permissive regular expression range is a regular expression pattern that matches a broader range of characters than you intended and potentially leads to unexpected behaviour or security vulnerabilities in a software expression. It said that any confusion should be avoided by writing unambiguous regular expressions and constantly checking that character ranges match only the expected characters. To resolve this issue, instead of using the regular expression to validate the email format, we decided to use the joi library and create a schema to validate the email format and not use regular expression.

For our invite system, a user can invite another user to their trip by email. For this, we needed to validate emails on the frontend before we gave the input to the backend. For this, we defined the input field to cause an error if an invalid email was entered. However, the default html email field only checks if the “@” character is present. To not cause HTTP 400 errors, we used the following regular expression to check if the given email had a domain:

```
/\.[A-z]{2,3}$/.test(email)
```

However, this was also flagged as an “overly permissive regular expression range”. After a few attempts, CodeQL considered the following regular expression as safe:

```
/\.([A-Z]|[a-z]){2,3}$/.test(email)
```