Crave 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)
Jared Castillo **castil13@myumanitoba.ca**	jablue-12	Full Stack Developer / DevOps ● Account feature ○ Registration/Login/ Logout logic using JWT token with tests ○ Updating user info (backend/frontend) ○ Account forms creation on frontend with validations ● Comments feature including tests (backend) ● Dish/Ingredient feature including tests ○ Forms creation along with validations (frontend) ○ Objects creation and relationship (backend) ● CI/CD pipeline and Docker ● SonarCloud security ● Load testing with JMeter See all assigned tasks for Jared in details: [Jared - Issues · jablue-12/crave (github.com)]()
Xiaoran (David) Meng **mengx5@myumanitoba.ca**	xiaoran-MENG	Full Stack Developer ● Account feature ○ Registration and

		login forms ◦ Authentication context API ● Dashboard ◦ UI toolbar to support filtering and sorting ◦ Dish listing and rating ◦ Dish details with ingredients data visualization, comments sections, and OpenAI integration ● User profile ◦ User preference data visualization ◦ User past order listing ◦ User details editing ● Shopping cart ◦ Google map integration with the algorithm to find the closest branch ◦ Cart context API to support CRUD operations for cart items ● Navbar ◦ Application logo design with typewriter ◦ Nav icons layout ● Common UI components ◦ Submit button ◦ Loader ● Acceptance testing with Cypress See all assigned tasks for Xiaoran in details: [Xiaoran - Issues · jablue-12/crave (github.com)]()
Eddie Wat **wate@myumanitoba.ca**	ediwat	Frontend Developer ● User profile tabs layout ● Authentication context API

		See all assigned tasks for Eddie in details: [Edi - Issues · jablue-12/crave (github.com)]()
Koye Fatoki **fatokioo@myumanitoba.ca**	koy-e	Backend Developer ● Implementation of order feature on backend. ● wrote swagger api documentation for interacting with the backend. ● implement unit and integration tests for backend components. ● changed backend objects to include user authentications See all assigned tasks for Koye in details: [Koye - Issues · jablue-12/crave (github.com)]()

Project summary

1) Elevator pitch description at a high-level. 2) highlight the differences between the final version and proposal if applicable

Indulge in a culinary adventure with our web app! A multi-column dashboard offers easy navigation – filter dishes, explore ratings, and dive into detailed dish pages. What sets us apart? Our OpenAI-powered side dish recommendations, ensuring the perfect pairing with just one click.

The shopping cart sidebar adds convenience, featuring a map marking the nearest branch to pick up your order and a list of cart items for easy updates. Your user profile showcases past orders and dish preferences via a visual radar chart.

Admins can curate the menu seamlessly, crafting dishes with an image uploader, ingredient selection, and quantity customization. Elevate your dining experience – explore our web app for a perfect blend of technology and taste!

GitHub repository Link

Repository of the project can be found here: https://github.com/jablue-12/crave

Frontend Project is deployed on netlify: https://crave-comp4350.netlify.app/

Backend project is deployed on render: https://crave-backend-api.onrender.com/

- Initial load may take a few minutes since we are using the free tier services on Render for the backend and database.

DockerHub repository link

All of the instructions and links for the docker image repositories can be found here on our DockerHub wiki page: DockerHub · jablue-12/crave Wiki (github.com)

List of user stories (for **example**) for each sprint

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

Sprint1

User Stories created and planned on this sprint.

Sprint 2

US #1: As a new user, I want to be able to create a new account so that I can start ordering food. [Status: Done]

US #2: As a user, I want to be able to login to the application so that I can access my account and place orders. [Status: Done]

US #14: As a registered user, I want to be able to add items to my shopping cart so that I can track the items I want to purchase and proceed to checkout. [Status: Done]

US #11: As a user, I want to be able to see the restaurants in my area, including their ratings and ordering times so that I can quickly decide where to order from. [Status: Partially Done]

Sprint 3

US #7: As a user, I want to be able to leave comments and ratings for restaurants and food so that I can share my experience with other users. [Status: Done]

US #10: As a restaurant wonder, I want to be able to manage my restaurant's profile and menu details so that I can attract more customers. [Status: Done]

US #13: As a user, I want to be able to access my recent orders and track my order status from the dashboard. [Status: Done]

US: #15: As a registered user, I want to be able to view my order history so that I check the accuracy of my past orders and get recommendations of my preferred dishes. [Status: Done]

US: #12: As a user, I want to be able to search for specific restaurants or cuisines directly from the dashboard so that I can quickly determine what I am craving. [Status: Removed]

US: #5: As a user, I want to be able to know how busy a restaurant is at a glance, just like how I can tell by physically being at the restaurant location, so that I can know wait times at a glance. [Status: Removed]

US #6: As a registered restaurant admin, I want to be able to know what items are being ordered in high volumes, so that I can manage stock and prep food more effectively. [Status: Removed]

US #9: As a user, I want to see my order history in my account so I can keep track of my past orders. [Status: Done]

Sprint4

US #8: As a user, I want to be able to see the overall rating and the number of reviews for each restaurant so that I can make informed choices. [Status: Partially Done]

US #16: As a registered user, I want to be able to place an order for pickup from a nearby restaurant so that I can get my food quickly while avoiding delivery bills. [Status: Partially Done]

US #17: As a registered restaurant admin, I want to be able to receive and decline orders so that I can manage restaurant operations efficiently. [Status: Partially Done]

US #3: As a user, I want to be able to edit my account information such as my name, address, and phone number. [Status: Done]

Release

User manual

Provide instructions on how to run the application for each core feature.

All the steps for running the application for each core feature can be found on the User Manual wiki page: User Manual · jablue-12/crave Wiki (github.com)

Overall Arch and Design

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

All diagrams related can be found on this wiki page: Diagrams · jablue-12/crave Wiki (github.com)

Infrastructure

For each library, framework, database, tool, etc

Frontend	**Backend**	**DevOps**
React React Bootstrap React Google Map ReCharts	Maven Sprint Boot Sprint Security PostgreSQL	Docker Github Actions SonarCloud

Bootstrap React Icons Eslint Lodash Axios Swiper React Router	OpenAI	

Name and link
1 paragraph description of why you are using this framework, not other alternatives and why you didn't choose them.

The chosen tech stack reflects a strategic combination of tools and frameworks tailored to meet the specific needs of our project. React was selected for the frontend due to its component-based architecture and functional programming style, facilitating modular development and ensuring a seamless user interface. React Bootstrap was chosen for its integration with React, offering responsive and visually consistent UI components. The decision to employ React Google Map and ReCharts further enhances the user experience by providing interactive maps and robust data visualization capabilities. In the backend, Spring Boot was preferred for its efficiency in building scalable and secure applications, with Spring Security ensuring robust authentication and authorization mechanisms. PostgreSQL was chosen as the database, given its reliability and extensive feature set. OpenAI integration supports the recommendation of complementary side dishes. DevOps practices are streamlined through Docker, GitHub Actions, and SonarCloud, providing fast feedback loops. This stack not only aligns with our project requirements but also demonstrates a focus on modern, widely adopted technologies to ensure long-term maintainability and scalability. The exclusion of alternatives was based on careful consideration of factors such as community support, ease of integration, and alignment with project goals, ensuring a cohesive and effective tech stack.

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

For frontend naming convention, we use the standard ESLint package standard/eslint-config-standard: ESLint Config for JavaScript Standard Style (github.com)

For backend naming convention, we use Java naming conventions

We also tried to follow this guideline structure for both backend and frontend as possible on this wiki page: Coding Standards and Guidelines · jablue-12/crave Wiki (github.com)

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)
backend/src/main/java/com/crave/backend/config/CraveConfig.java View	A configuration file responsible for instantiating authentication-related files, as well as creating data based on different environments i.e. docker, local and production.
https://github.com/jablue-12/crave/blob/main/backend/src/main/java/com/crave/backend/service/JwtService.java View	The service manages JSON Web Tokens (JWTs), providing methods for secure token generation, validation with user details, and expiration handling. It enhances authentication capabilities by employing a secret key obtained from the environment and using the HS256 algorithm for encoding.
https://github.com/jablue-12/crave/blob/main/frontend/src/components/features/dashboard/core/Filter.jsx View	This UI component generates a set of badges representing dish tags for filtering. It allows users to toggle tags, dynamically updating the displayed dishes based on the selected tags and their corresponding ratings. The component integrates with external data for dish tags and utilizes the Lodash library for array manipulation.
https://github.com/jablue-12/crave/blob/main/frontend/src/contexts/CartContext.js View	The Context API manages a shopping cart to be used across multiple UI components, providing functions for adding, removing, and retrieving items, while ensuring synchronization with local storage for state persistence.
frontend/src/components/pages/Dashboard.jsx View	The multicolumn dashboard that contains a toolkit for sorting and filtering, listing for featured and regular dishes, and the details for the

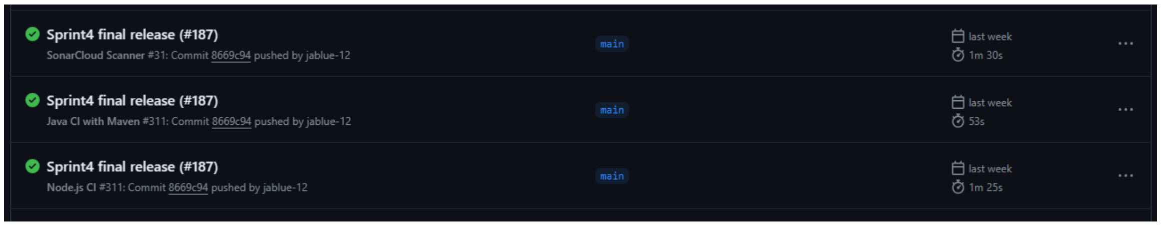
	selected dish.

Continuous Integration and deployment (CI/CD)

1) Describe your CI/CD environment and the clickable link to your CI/CD pipeline. For instance, if you use GitHub Action, provide the link to the workflow; if you use Jenkins, provide the link to the pipeline file.
2) Snapshots of the CI/CD execution. Provide one for CI and one for CD to demo your have successfully set up the environment.

We are using GitHub Action for our CI/CD environment which can be found under the .github/workflows directory

Here is a snippet of our CI pipeline being executed when we merge changes from develop to the main branch. This is a link to all of the executions of the CI Workflow runs · jablue-12/crave (github.com)



✓ **Sprint4 final release (#187)**
SonarCloud Scanner #31: Commit 8669c94 pushed by jablue-12
main
last week
1m 30s

✓ **Sprint4 final release (#187)**
Java CI with Maven #311: Commit 8669c94 pushed by jablue-12
main
last week
53s

✓ **Sprint4 final release (#187)**
Node.js CI #311: Commit 8669c94 pushed by jablue-12
main
last week
1m 25s

We have 3 yml files: 1 for the frontend, 1 for the backend and the other is for sonarcloud.

frontend.yml file

- This GitHub Actions YAML configuration file sets up a workflow named "Node.js CI" triggered by pushes to the "main" or "develop" branches and pull requests targeting these branches. The workflow defines a "build" job that runs on the latest Ubuntu environment. The job includes steps to checkout the code, install dependencies in the "./frontend" directory using npm, build the frontend application, and run tests. This configuration is suitable for a Node.js project with a frontend, providing continuous integration by automating code checks, dependencies installation, and testing processes upon specified events in the GitHub repository.

backend.yml file

- Similarly, this GitHub Actions YAML configuration file defines a workflow named "Java CI with Maven" that is triggered on pushes to the "main" or "develop" branches and on pull requests targeting these branches. The workflow includes a "build" job that runs on the latest Ubuntu environment, setting an environment variable "SECRET_KEY" (used for testing authentication-related files) from GitHub Secrets. The job involves checking out the code, setting up JDK 21 and Maven using GitHub Actions, building the backend with Maven in the "./backend" directory, and running tests. The Maven build commands include passing the "SECRET_KEY" as a parameter, allowing secure configuration for the Java application. This configuration is well-suited for a Java project using Maven, ensuring continuous integration, testing, and secure handling of sensitive information during the workflow execution.

sonarcloud-scanner.yml file

- This GitHub Actions YAML configuration file establishes a workflow named "SonarCloud Scanner" triggered on pushes to the "main" or "develop" branches and specific pull request events (opened, synchronize, reopened). The workflow includes a "build" job that runs on the latest Ubuntu environment. The steps involve checking out the code, setting up JDK 17 using GitHub Actions, caching SonarCloud and Maven packages to optimize subsequent runs, and finally building and analyzing the code using Maven and the SonarCloud Scanner plugin. The environment variables for GitHub and SonarCloud tokens are configured securely. The analysis is specifically targeted at the "backend" directory with Maven, and the Sonar project key is provided for proper identification in SonarCloud. This configuration is designed for integrating automated code analysis with SonarCloud into the continuous integration pipeline of a Java project, ensuring code quality checks during development.

Testing

Link to testing plan

Here is the link to the testing plan: crave/docs at main · jablue-12/crave (github.com)

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 class/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 test 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)
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/repository/DishRepositoryTest.java#L79 View	This test verifies that the dish repository correctly retrieves a list of dishes tagged with 'rave/blob/main/backend/src/test/java/cobeef' and validates the details of each dish, including name, description, tag, image, price, and rating.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/repository/DishRepositoryTest.java#L125 View	This test ensures that the dish repository correctly retrieves a list of dishes with either 'chicken' or 'beef' tags and validates the details of each

	dish, including name, description, tag, image, price, and rating.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/repository/AccountRepositoryTest.java#L21 View	This test verifies that the account repository correctly retrieves an existing account by email and validates the details, including ID, first name, last name, email, password, and user role.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/utils/ImageUtilsTest.java#L11 View	This test validates the successful compression of image data using the ImageUtils class, ensuring that the compressed data is not null and has a smaller size than the original data.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/utils/ImageUtilsTest.java#L26 View	This test verifies the successful decompression of previously compressed image data using the ImageUtils class, ensuring that the decompressed data matches the original data.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/service/DishServiceTest.java#L60 View	This test validates the successful creation of a dish using the DishService, ensuring that the repository's save method is called with the correct dish to create and that the created dish matches the expected dish.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/service/DishServiceTest.java#L38 View	This test verifies the correct retrieval of dishes using the DishService, ensuring that the repository's findAll method is called and that the returned list of dishes matches the expected list.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/service/IngredientServiceTest.java#L53 View	This test validates the correct creation of an ingredient using the IngredientService, ensuring that the repository's save method is called with the correct ingredient to create and that the created ingredient matches the expected ingredient.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/service/IngredientServiceTest.java#L34 View	This test verifies the correct retrieval of ingredients using the IngredientService, ensuring that the repository's findAll method is called and that the returned list of ingredients matches the expected list.
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/unit/service/CommentServiceTest.java#L59	This test validates the correct creation of a comment using the CommentService, ensuring that the

View	repository's save method is called with the correct comment to create and that the created comment's content, associated dish, and account match the expected values.

List the **5** most important integration test 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)
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/integration/DishServiceTest.java#L29 View	This test verifies the connection between the dish service and the database upon dish creation
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/integration/DishServiceTest.java#L54 View	This test verifies the connection between the dish service and the database upon fetching the dishes that match the tags specified
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/integration/AuthenticationServiceTest.java#L29 View	This test verifies the connection between the authentication service and the database upon registration using the user details specified
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/integration/IngredientServiceTest.java#L30 View	This test verifies the connection between the ingredient service and the database upon ingredient creation
https://github.com/jablue-12/crave/blob/main/backend/src/test/java/com/crave/backend/integration/CommentServiceTest.java#L60 View	This test verifies the connection between the comment service and the database upon comment creation

List the **5** most important acceptance tests with links below.If your acceptance tests are done manually, you should have detailed steps how to run the test and the expected outcome for 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
https://github.com/jablue-12/crave/blob/main/frontend/cypress/e2e/auth.cy.js#L9 View	This test intercepts and verifies the application successfully registers a user using the provided registration functionality.
https://github.com/jablue-12/crave/blob/main/frontend/cypress/e2e/comments.cy.js#L9 View	This test validates that users can successfully add comments to a selected dish by interacting with the user interface, entering comment content, and submitting the comment, with the application reflecting the new comment in the comments section for the respective dish.

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
https://github.com/jablue-12/crave/blob/main/frontend/cypress/e2e/auth.cy.js#L9 View	This test intercepts and verifies the application successfully registers a user using the provided registration functionality.
https://github.com/jablue-12/crave/blob/main/frontend/cypress/e2e/dashboard.cy.js#L42 View	This test verifies that the application correctly filters dishes by selected tags (e.g., 'Pizza' and 'Chinese') and ensures that the displayed dish images have tags matching the selected criteria.
https://github.com/jablue-12/crave/blob/main/frontend/cypress/e2e/cart.cy.js#L12 View	This test validates that selecting a dish from the displayed images, clicking 'Add to Cart,' and navigating to the shopping cart correctly updates the cart items count to reflect the addition of the selected dish.
https://github.com/jablue-12/crave/blob/main/frontend/cypress/e2e/cart.cy.js#L29 View	This test ensures that the application accurately calculates and displays the subtotal in the shopping cart based on the quantities and prices of the items present in the cart.

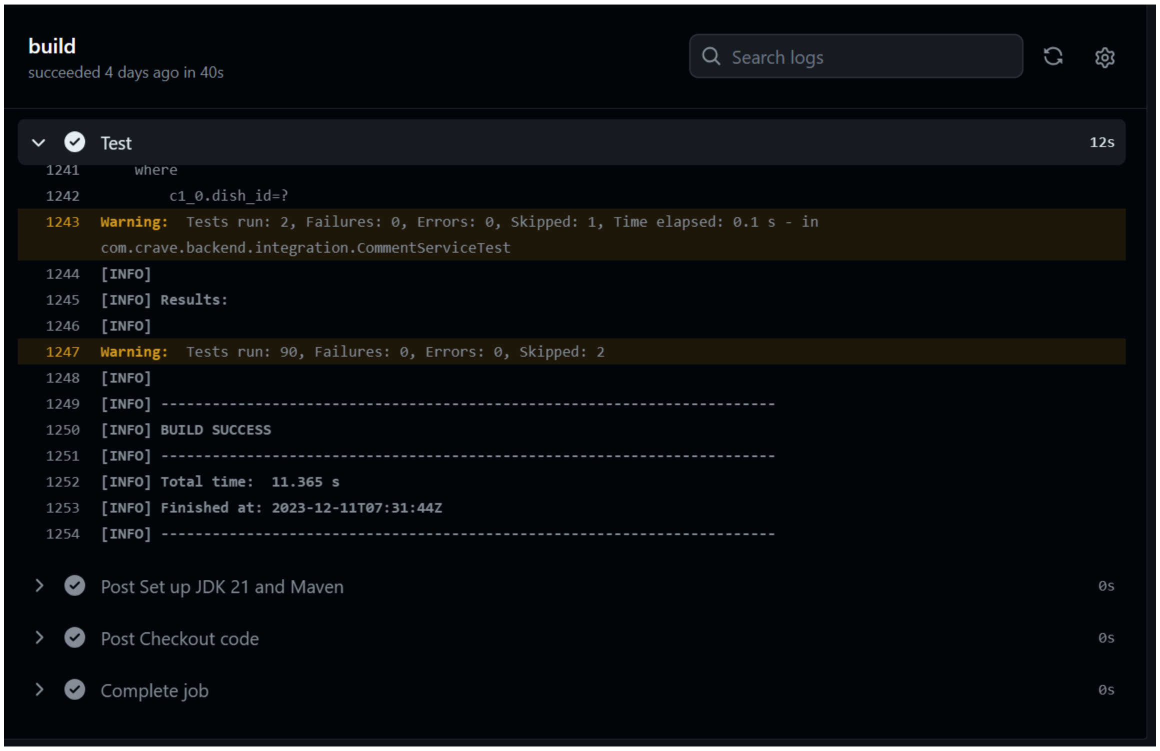
Regression testing

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 regression testing script and provide last snapshot of the execution and results of regression testing.

In our CI/CD pipeline for the Java project with Maven, regression testing is conducted during the "Test" step in the workflow. Maven is used to execute the tests located in the "./backend" directory. These tests cover a comprehensive suite of unit tests, and integration tests that collectively ensure the continued functionality of existing features after code changes.

As our regression testing is seamlessly integrated into our CI/CD workflow using GitHub Actions, you can view the latest execution and results directly on the GitHub Actions dashboard. Here's the link to the Actions tab for our repository: Merge pull request #188 from jablue-12/backend/update-readme · jablue-12/crave@3511b7c (github.com)

Here is a snippet of the recent execution of our regression testing.


build
succeeded 4 days ago in 40s

Test 12s

```
        where
            c1_0.dish_id=?
Warning:  Tests run: 2, Failures: 0, Errors: 0, Skipped: 1, Time elapsed: 0.1 s - in
com.crave.backend.integration.CommentServiceTest
[INFO]
[INFO] Results:
[INFO]
Warning:  Tests run: 90, Failures: 0, Errors: 0, Skipped: 2
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.365 s
[INFO] Finished at: 2023-12-11T07:31:44Z
[INFO] ------------------------------------------------------------------------
```

Post Set up JDK 21 and Maven 0s
Post Checkout code 0s
Complete job 0s

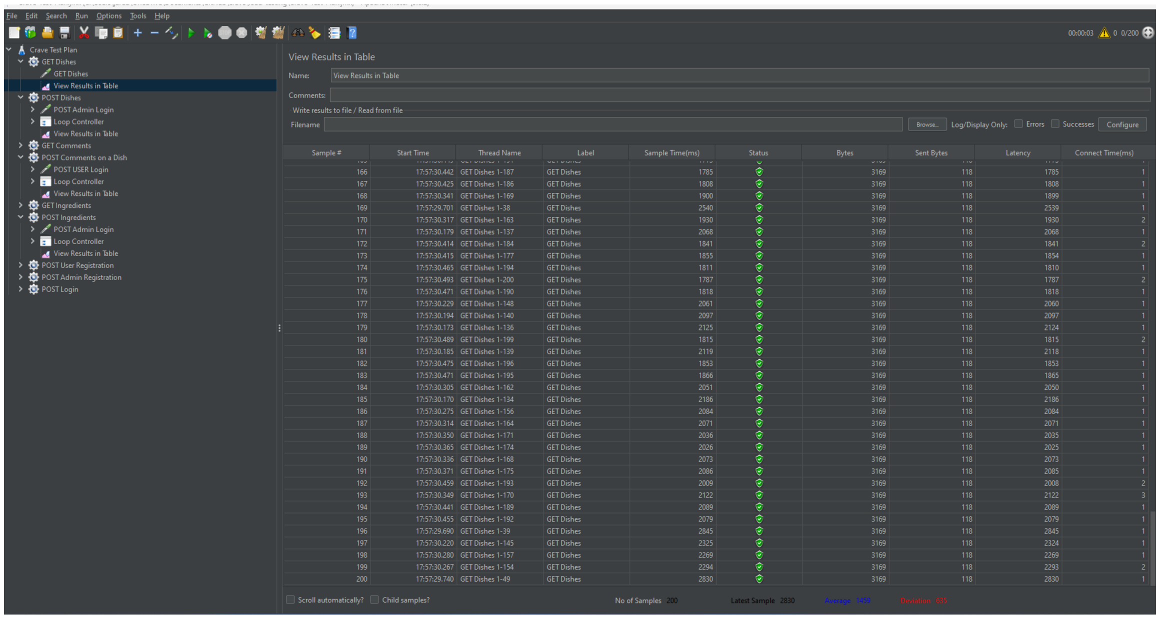
Load testing

1) Describe the environment for load testing, such as tool, load test cases. 2) provide the test report for load testing. 3) discuss one bottleneck found in the load testing. For instance, if you use Jmeter, please upload the jmx file on GitHub and provide link. Also a snapshot of the results, discuss whether the load testing is passed or not.

Our load testing can be found on this page which consists of .jmx and csv files for results: crave/load-testing at main · jablue-12/crave (github.com)

In our Apache JMeter load testing environment, we focused on testing various features of our system. Simulating real-world scenarios, we employed 200 threads for each of the following: user and admin registration, admin actions such as adding dishes and ingredients (with 1 thread looping 200 times for each action), and user interactions like adding comments on a dish (1 thread looping 200 times). Additionally, we examined the performance of retrieving dishes, ingredients, and comments for non-authenticated users, allocating 200 threads to each retrieval action. The comprehensive CSV report capturing key metrics, including response times and error rates, is available on this page, while the JMX file detailing the JMeter script and configurations can be accessed here. Notably, our load testing identified a bottleneck in the database under heavy load during certain actions, underscoring the importance of optimizing these specific functionalities for enhanced system performance.

Below is a snapshot of the result for retrieving the dishes with 200 threads.



View Results in Table

Name: View Results in Table

Comments:

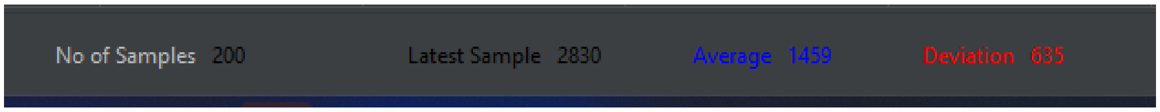
Write results to file / Read from file

Filename

Log/Display Only: Errors Successes

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
166	17:57:30.442	GET Dishes 1-187	GET Dishes	1785	✓	3169	118	1785	1
167	17:57:30.425	GET Dishes 1-186	GET Dishes	1808	✓	3169	118	1808	1
168	17:57:30.341	GET Dishes 1-169	GET Dishes	1900	✓	3169	118	1899	1
169	17:57:29.701	GET Dishes 1-38	GET Dishes	2540	✓	3169	118	2539	1
170	17:57:30.317	GET Dishes 1-163	GET Dishes	1930	✓	3169	118	1930	2
171	17:57:30.179	GET Dishes 1-137	GET Dishes	2068	✓	3169	118	2068	1
172	17:57:30.414	GET Dishes 1-184	GET Dishes	1841	✓	3169	118	1841	2
173	17:57:30.415	GET Dishes 1-177	GET Dishes	1855	✓	3169	118	1854	1
174	17:57:30.465	GET Dishes 1-194	GET Dishes	1811	✓	3169	118	1810	1
175	17:57:30.493	GET Dishes 1-200	GET Dishes	1787	✓	3169	118	1787	2
176	17:57:30.471	GET Dishes 1-190	GET Dishes	1818	✓	3169	118	1818	1
177	17:57:30.229	GET Dishes 1-148	GET Dishes	2061	✓	3169	118	2060	1
178	17:57:30.194	GET Dishes 1-140	GET Dishes	2097	✓	3169	118	2097	1
179	17:57:30.173	GET Dishes 1-136	GET Dishes	2125	✓	3169	118	2124	1
180	17:57:30.489	GET Dishes 1-199	GET Dishes	1815	✓	3169	118	1815	2
181	17:57:30.185	GET Dishes 1-139	GET Dishes	2119	✓	3169	118	2118	1
182	17:57:30.475	GET Dishes 1-196	GET Dishes	1853	✓	3169	118	1853	1
183	17:57:30.471	GET Dishes 1-195	GET Dishes	1866	✓	3169	118	1865	1
184	17:57:30.305	GET Dishes 1-162	GET Dishes	2051	✓	3169	118	2050	1
185	17:57:30.170	GET Dishes 1-134	GET Dishes	2186	✓	3169	118	2186	1
186	17:57:30.275	GET Dishes 1-156	GET Dishes	2084	✓	3169	118	2084	1
187	17:57:30.314	GET Dishes 1-164	GET Dishes	2071	✓	3169	118	2071	1
188	17:57:30.350	GET Dishes 1-171	GET Dishes	2036	✓	3169	118	2035	1
189	17:57:30.365	GET Dishes 1-174	GET Dishes	2026	✓	3169	118	2025	1
190	17:57:30.336	GET Dishes 1-168	GET Dishes	2073	✓	3169	118	2073	1
191	17:57:30.371	GET Dishes 1-175	GET Dishes	2086	✓	3169	118	2085	1
192	17:57:30.459	GET Dishes 1-193	GET Dishes	2009	✓	3169	118	2008	2
193	17:57:30.349	GET Dishes 1-170	GET Dishes	2122	✓	3169	118	2122	3
194	17:57:30.441	GET Dishes 1-189	GET Dishes	2089	✓	3169	118	2089	1
195	17:57:30.455	GET Dishes 1-192	GET Dishes	2079	✓	3169	118	2079	1
196	17:57:29.690	GET Dishes 1-39	GET Dishes	2845	✓	3169	118	2845	1
197	17:57:30.220	GET Dishes 1-145	GET Dishes	2325	✓	3169	118	2324	1
198	17:57:30.280	GET Dishes 1-157	GET Dishes	2269	✓	3169	118	2269	1
199	17:57:30.267	GET Dishes 1-154	GET Dishes	2294	✓	3169	118	2293	2
200	17:57:29.740	GET Dishes 1-49	GET Dishes	2830	✓	3169	118	2830	1

Scroll automatically? Child samples? No of Samples 200 Latest Sample 2830 Average 1459 Deviation 635



No of Samples 200 Latest Sample 2830 Average 1459 Deviation 635

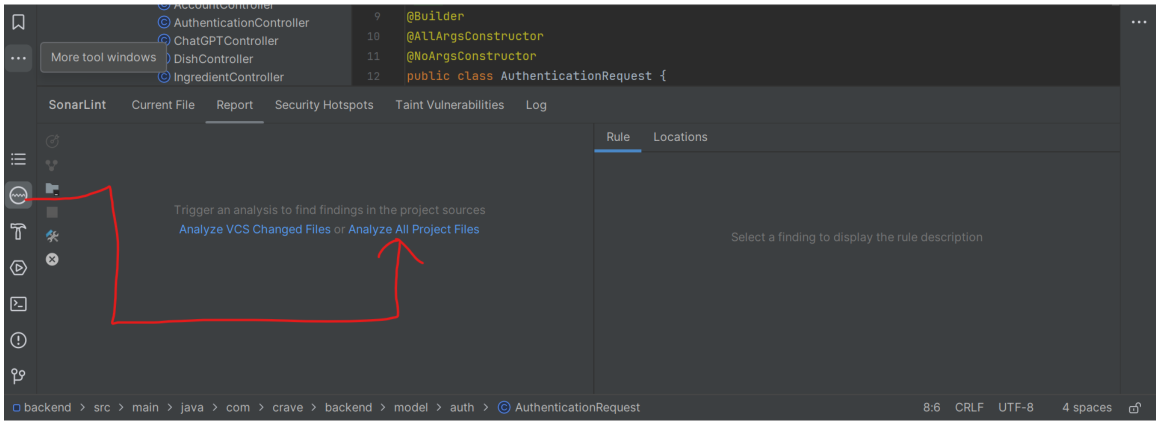
Security analysis

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

For our project, we used the SonarCloud for security analysis tool. We have created a yml file for this tool to be used in our CI/CD pipeline which is located under the .github/workflows directorySonarCloudScanner. This action is triggered whenever we make any push or pull requests on main and develop branches.

SonarCloud is used in our backend application and we use SonarLint which is an extension in IntelliJ IDE to give us reports on our java classes.

Clicking the 'Analyze All Project Files' will run the sonarlint for the project as shown below.



```
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class AuthenticationRequest {
```

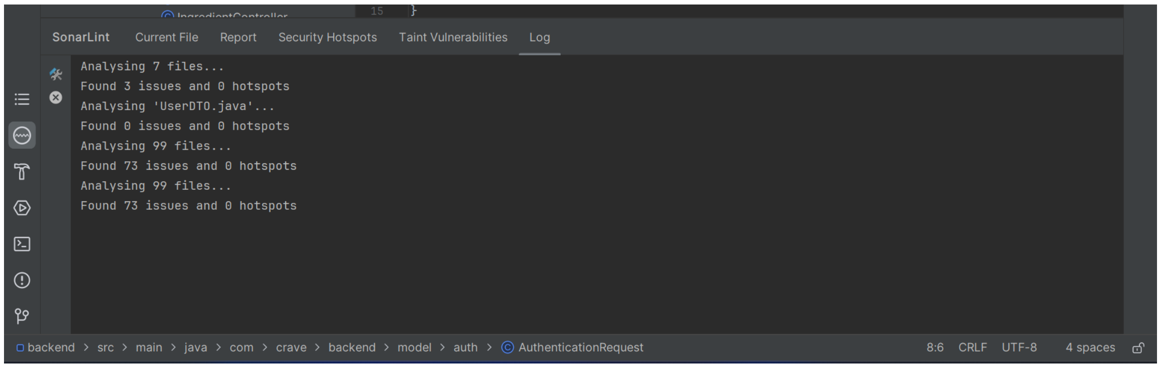
Trigger an analysis to find findings in the project sources
Analyze VCS Changed Files or Analyze All Project Files

Select a finding to display the rule description

backend > src > main > java > com > crave > backend > model > auth > AuthenticationRequest   8:6 CRLF UTF-8 4 spaces

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.

Report:



```
Analysing 7 files...
Found 3 issues and 0 hotspots
Analysing 'UserDTO.java'...
Found 0 issues and 0 hotspots
Analysing 99 files...
Found 73 issues and 0 hotspots
Analysing 99 files...
Found 73 issues and 0 hotspots
```

backend > src > main > java > com > crave > backend > model > auth > AuthenticationRequest   8:6 CRLF UTF-8 4 spaces

Here is an instance of when the SonarCloud github action is built:
[Integrate SonarCloud to CI · jablue-12/crave@70f885f (github.com)]()