Finesse Nation Documentation

Aditya Pandey, Robert Beckwith, Krastan Dimitrov, Jeffrey Josol, Shilpa Rani

Table of Contents

Description **1**

User Manual **1**

End-User Instructions 1

Entering the App 1

Viewing Posts 1

Filtering posts 2

Adding a Post 2

Settings 3

Install the Frontend 3

Install the Backend 4

Process Followed **4**

Stand-ups: 5

Refactoring 5

Pair Programming 5

Version Control 5

Change Control 6

Collaboration 6

Continuous Integration/Continuous Delivery (CI/CD) 6

Frontend 6

Backend 6

Testing 7

Flutter Unit Tests 7

Flutter Integration Tests 7

Flutter Widget Tests 7

Mocha Tests 8

Jest Tests 8

Monkey Tests 8

Postman Tests 8

Coveralls 8

Risks 9

Flutter 9

Covid-19 9

Requirements (Use Cases) **9**

Architecture **10**
Flow Chart 10
Architecture Diagram 11
How the Framework Influenced Design 11
Flutter 11
Express 12
Reflections and Lessons Learned **12**
Robert Beckwith Reflection 12
Krastan Dimitrov Reflection 12
Shilpa Rani Reflection 13
Aditya Pandey Reflection 13
Jeffrey Josol Reflection 13
Appendix **14**
Dartdoc 14
JSdoc 14
HTTP APIs 15
Dependencies 15

Description

There are free food, events, and giveaways all around campus all the time. We will refer to these free items as “finesses”. There is a need for an app that would allow users to share where and when these finesses can be acquired by students. There is currently no simple centralized app to be able to share and notify users of finesses on our campus. Because of this, there are so many finesses on campus that we miss out on. To take full advantage of all these finesses there is a need for a centralized and specialized app to find these. Currently there is a 500+ person group chat to find finesses but it is difficult to moderate and people mute it because it is spammed. Our proposal to fix this issue is to make an Android app that can concisely and effectively display finesses and their details, such as time, location, duration, etc. Users will also be able to post images, comment on posts to follow up, and filter by type and expiration status.

User Manual

End-User Instructions

Entering the App

1. Open the app
2. If the user has a pre-existing account, they must enter their username and password then click login. Otherwise they must click signup and enter their email, and their chosen password 2 times then click signup again.

Viewing Posts

1. Once logged in, the user is greeted by a list of finesses. They can scroll to view most vital information including title, location and time since posting.
2. By tapping on an event, the user is brought to a more detailed view of the finesse. This also includes the status of the event, whether it is active or inactive, the email of the user that posted it, the number of upvotes/downvotes, and comments.
3. Tapping on the location text will open google maps and show the location on the map.

4. If the user has not previously voted on the post, they can click the upvote/downvote arrows across from the vote counter which will reflect their change. If the user has previously voted on the post, they can change their vote by clicking on the opposite arrow than they originally voted.
5. They can add a comment by typing in the text box that says “Add a comment”. Once they have written the post, they can click the send arrow on the right of the text box which will post their comment.
6. If a post has ended, a user can mark it as inactive. They can do this by tapping the 3 dot menu in the top right then clicking on “Mark as Inactive”.

Filtering posts

1. Filtering posts can be done from the main page of finesses. Tap on the upside down pyramid button in the top right.
2. A menu pops up where the user can click enable/disable inactive posts and non food posts.
3. Tapping the sliders changes the current selection to the opposite and the button changes color and text to reflect the change.

Adding a Post

1. From the main page of finesses, tap on the plus in a circle in the bottom right of the screen.
2. The user will then see a new page.
3. There they can enter the title and location which are mandatory.
4. They can also enter the type of finesse, the description and the duration.
5. Adding an image is also optional, and can be done by clicking the “Upload Image” button. Then tapping on either upload image from gallery or camera. If the user chose gallery, they can choose an image from their phone by tapping on it. If the user chose camera, then they are able to take a photo with their device.
6. Once the user is done adding as many details as they want, they can submit the post by tapping on the “Submit” button or clicking the left arrow in the top left if they want to discard the post.

Settings

1. From the main page of finesses, tap the 3 dots in the top right corner.
2. That will make a little pop-up that says “Settings”, tap on that.
3. That opens a new screen, where the user can toggle notifications whether they receive notifications when new posts are created by other users.
4. This is also where the user can log out if they so choose by tapping the “Logout” button.
5. If they don’t want to log out, or want to go back to the main screen, the user must tap the left pointing arrow in the top left.

Install the Frontend

1. Download and install the Flutter and Dart SDKs
    a. Flutter SDK releases
    b. Get the Dart SDK
2. Install and setup an IDE that is compatible with Flutter (we used Android Studio)
    a. Set up an editor
3. Clone Finesse-Nation-Frontend from Github
    a. `git clone https://github.com/Periphery428/Finesse-Nation-Frontend.git`
4. install all dependencies
    a. `flutter pub get`
5. Add the required environment variables to your computer
    a. [FN] Passwords, Tokens, and Usernames
6. Change the Flutter build release channel to master
    a. `flutter channel master`
7. Setup Firebase Cloud Messaging if you want push notifications
    a. Add Firebase to your Flutter app
8. Connect a device and run the project to build and install the app
    a. `flutter run`

Install the Backend

1. Download and install node.js
    a. Download
2. Clone the backend repo
    a. `git clone https://github.com/Periphery428/Finesse-Nation-Backend.git`
3. Install dependencies
    a. Server
        i. `npm install`
    b. Client
        i. `cd src/client -> npm install`
4. Add the required environment variables if you haven't already to .env file.
5. Start the server and client
    a. Server
        i. `npm run start`
    b. Client
        i. `cd src/client -> npm run start`
6. Set up heroku if you want a remote deployment as well
    a. Getting Started on Heroku with Node.js
7. Enable automatic deployment from master if you want to redeploy on changes
    a. GitHub Integration (Heroku GitHub Deploys)

Process Followed

The Finesse Nation team used Agile software development with the Scrum framework. We used biweekly sprints and followed a strict iteration plan throughout. At the end of each iteration, we would have the sprint review and sprint retrospective with the professor. These meetings went over requirements, progress, and demos ([FN] Iteration Agendas). We would then meet as a team for sprint planning, where we decide upon the use cases to be implemented for the current iteration based on the priority assigned to each backlog. Once we agreed on the user stories, we would not deviate during the sprint.

Stand-ups:

Scrum usually has daily stand ups. We, as students working part-time on this project, replaced this aspect with weekly and sometimes biweekly meetings Wednesday and Friday. The meeting minutes can be found here: [FN] Meeting Minutes

Refactoring

Part of our iterations also involved refactoring. Most of our refactorings involved eliminating duplicated code by extracting it into its own function. This allowed for easier maintenance in the future since we only needed to update code in one spot if we decided to change something. We made use of extensive testing to make sure that when we refactor, no changes are made to the functionality of the code.

Pair Programming

We did not utilize pair programming for this project. We decided it would be easier for each of us to work individually and pair program if we needed help with an issue. Scheduling with pair programming is difficult to do with student schedules. In highsight, due to everything going remote, we made the right choice as we are able to continue as normally.

Version Control

We set up our version control system into two separate repositories on github. The two repositories can be found here: https://github.com/Periphery428. We utilized these repositories to track our changes and merge collaborated code together.

We decided to separate the repositories because the frontend and backend repo have very different purposes. The separation made it easier to set up a different CI/CD configuration for each part of the application. It also allowed us to get backend code running on the server earlier which made it easier to develop on the frontend with the live apis.

Change Control

We implemented a form of change control by requiring all members to commit to a new branch rather than directly into master. We also required making a pull request and requesting reviewers before merging, especially for larger changes. This practice turned out to be very beneficial as reviewers often caught code smells or bugs that would have been problematic in the future. It also promoted collective code ownership so multiple people had the knowledge required to work on a feature, it wasn't just limited to the original code author.

Collaboration

To collaborate we utilized pull requests through git and Facebook messenger. As stated earlier our team did not use pair programming. We instead worked on our own and would contact each other if we needed help. Overall this approach worked better for us because we could develop around our own schedule rather than scheduling time for two people. We were also able to split into our specialties better and implement what each member was good at i.e backend, frontend. Additionally, owning our own features has been more rewarding.

Continuous Integration/Continuous Delivery (CI/CD)

Github Actions is the CI/CD system we use for Finesse Nation. It runs on both the frontend and backend repositories and is triggered automatically when we push code.

Frontend

The frontend Github Actions runs all of the unit tests, deploys the website to github pages, and runs dart package analysis. We can also automatically build and release an apk (the final Android app) once we have finished an iteration by pushing to a tag.

Backend

The backend Github Actions runs coveralls, eslint and the unit tests for both the reactjs client and nodejs backend. Once all CI tests have succeeded, Github automatically signals Heroku to redeploy the new server.

Testing

We used many different frameworks to test our Finesse Nation application.

Flutter Unit Tests

To test the core function calls of our app we used Flutter's unit testing framework. We used this primarily to test our network methods. These methods would request and update data by calling the backend server. These tests were mainly useful as regression tests and have prevented many issues throughout the process.

Flutter Integration Tests

To test the whole user interface and system we used Flutter's Integration testing framework. This framework allowed us to find elements of the user interface by a key or by the text value. We then are able to input text or tap or swipe on the screen. This simulates an actual user using the system. The goal of these tests are to test the entire stack of the application by testing what the user can actually interact with.

Flutter Widget Tests

The flutter testing framework offers a third method of testing your application known as widget tests. A cross between unit tests and integration tests, widget tests will pump (i.e. build) an isolated, minimal instance of a widget that you can interact with by simulating taps, drags, text input, etc. We initially had a few widget tests, but we removed them in favor of unit and integration tests for a couple reasons. Primarily, because the widgets were created in an "artificial" test environment, the way they behaved wasn't exactly the same way it would behave in an actual production environment, which obviously isn't ideal. Another reason we avoided widget tests is because any HTTP calls made by interacting with a widget would intentionally fail, a problem not experienced by unit tests or integration tests.

Mocha Tests

To test the backend servers api calls we utilized Mocha test. This allowed us to test all paths of code in each api call to include successful/failed use cases and bad inputs. The mocha tests run in sequential order as they are listed and can exhibit flakiness on the CI/CD side since we are making real calls to the database.

Jest Tests

To test the reactjs client facing web app we used the jest testing framework. This includes checking for certain static components that exist when the page is loaded at runtime.

Monkey Tests

We made use of flutter's built-in monkey testing to test our front end UI and make sure that it cannot be broken by sending hundreds of random inputs to the app. It held up well and was not able to break the app, however it is important to continuously use it as new features are added as they can add new holes in the UI.

Postman Tests

We wrote integration tests in postman to test our APIs in the first iteration, when the APIs weren't implemented. A Postman mock server was used to mock the API calls.

Coveralls

To view the code coverage of both the mocha and flutter tests we integrated Coveralls onto our CI/CD pipeline. This allowed for easier detection of missing tests. Coveralls shows exactly which code is not covered by any tests. We can also run coveralls locally to check coverage percentage is satisfactory before pushing to our branches. We also set up the coveralls CI/CD workflow to require a certain coverage percentage (80%) and difference threshold (5%) to be met in order to pass.
Note: Coveralls for dart has a bug where we can't see what wasn't covered from the web interface. We utilized a local tool called gulp to see what we were missing.

Risks

Flutter

Everyone in the group had never used Flutter or dart before so that brought many new challenges but ultimately was a good learning experience. Another risk using Flutter and dart was that it was so new. Our team has encountered many missing features that are in many other languages that aren't implemented yet in Flutter. When searching for help there is a lack of Stack Overflow resources unlike when using native Android Java. We have made sure to factor this into our user story time estimates. Despite all the risks, Flutter has proven to be a good choice for the project to be able to learn a new framework and cross compile our application to more users.

Covid-19

We have taken measures to mitigate the risks associated the Covid-19 pandemic

1. We will remain vigilant.
2. All of our passwords to the project have already been shared in the chat. We have consistently been keeping a low bus score.
3. The pandemic hit after we had already completed the core of the Finesse Nation features. All the new ones were fun features to improve the app's appeal.

Requirements (Use Cases)

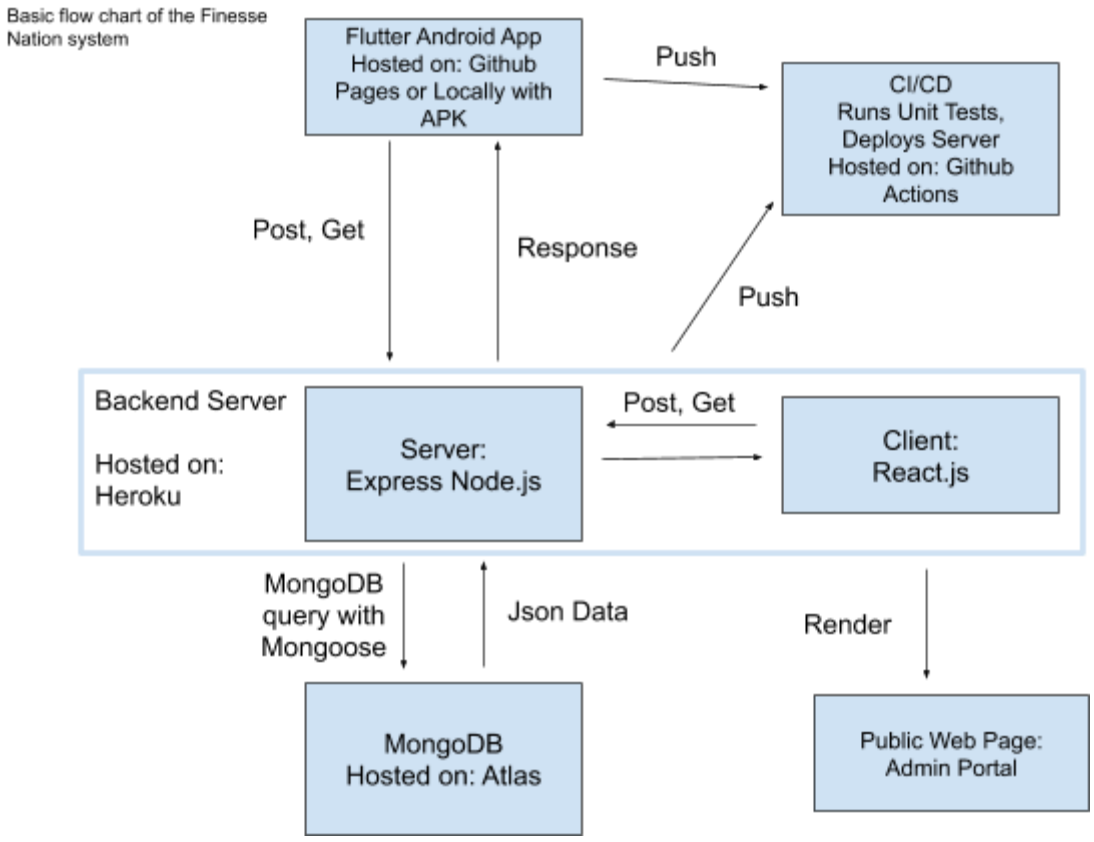
We followed a strict iteration plan throughout the Finesse Nation project.

The iteration plan and the descriptions of the user stories can be found here:

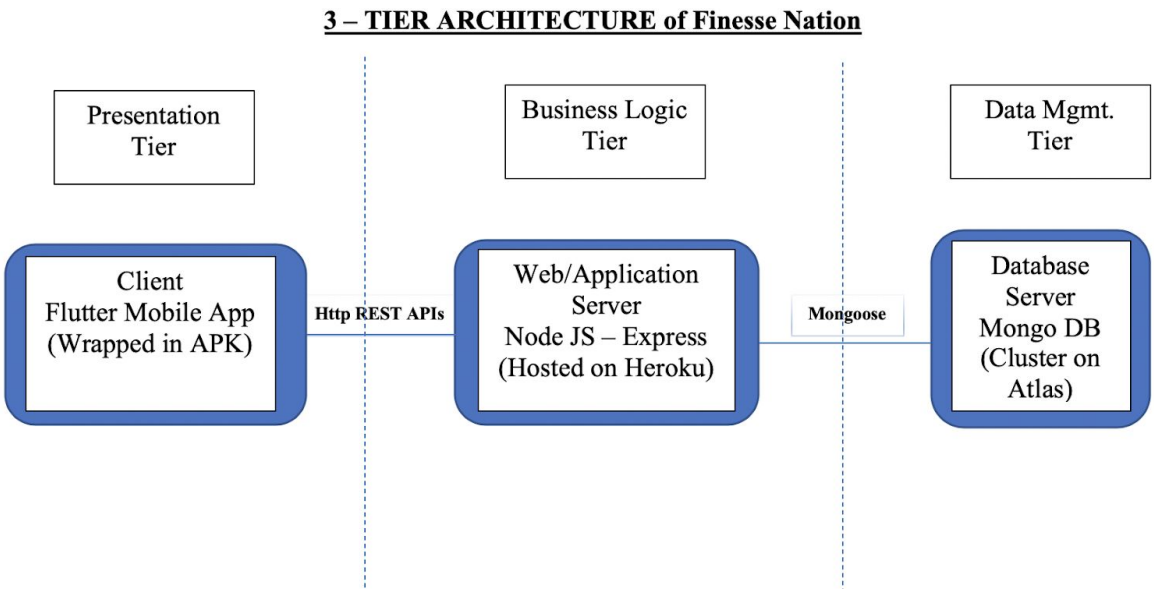
https://docs.google.com/document/d/1NMDJx-9s6HQjk3N8sILE-BZya6iZ8bdYF4RTEv0XjAk/edit#

Architecture

Flow Chart



Architecture Diagram



How the Framework Influenced Design

Flutter

The frontend mobile application was developed with the Flutter framework. Flutter is a new framework for the dart programming language developed by Google. The main decision to use this for this project was the ability to cross compile to Android, iOS, and web at the same time. Flutter compiles the code into native ARM code to run on mobile devices so there is no performance loss like there is in React native. Our overall architecture was designed around the theme of cross compiling the android app. We made sure to check what features would work in both the web and the android app.

Express

The backend server was developed with the express framework for Node.js. We could have chosen many different languages for this task. Our team decided to go with Node.js and Express because it is widely used and industry standard. This allowed us to get a lot of support online and use a lot of pre built tools. For example eslint is a great tool to enforce coding standards and Node.js allowed us to run that locally and run that on the CI/CD very easily. Node.js also has support for a very powerful MongoDB interface called Mongoose which allowed for easy querying of the database. Communicating with a frontend through REST APIs and a database through Mongoose made the three-tier architecture the obvious choice.

Reflections and Lessons Learned

Personal reflections from each Finesse Nation team member

Robert Beckwith Reflection

Throughout this project I have learned a great deal about developing a software project on a team from start to finish. I was also able to pick up a new language with dart and the flutter framework. Specifically the areas I learned the most from are testing and CI/CD. Prior to this project I had never worked with a CI/CD system and setting up our own and modifying it was a good learning experience. I have only done some brief testing in 428 prior to this course. The emphasis on testing forced me to get into the mindset of testing and try out testing. I now have a better mindset of the limits of testing and the advantages of testing going forward.

Krastan Dimitrov Reflection

As a first time app developer, everything I did for this project was a new experience for me. Writing code using a new framework in Flutter and a language I had never used before in Dart, gave me new challenges and taught me a lot about developing a front end. The emphasis on testing and refactoring was also very useful for me as testing played a huge role in allowing us to refactor without breaking the app and allowed us to write cleaner and less buggy code.

Working with a team also brought new challenges in coordination, teamwork, and splitting responsibilities. I learned that working in harmony takes a lot of communication and everyone doing their part.

Shilpa Rani Reflection

I have often worked in a setting wherein there have been dedicated teams assigned for even the smallest chunks of software development - backend development, frontend development, testing, documentation, project management etc. To a considerable extent, I got into a mindset of being slightly oblivious to what is being done in other teams. Working in this project has been a great experience for me since it provided me with an exposure to end to end software development and its intricacies involved. I firmly believe that this experience would play a pivotal role in my future endeavours in this field.

Aditya Pandey Reflection

As someone planning on moving into the software field, CS 428/9 was one of the most useful courses I have taken in the CS curriculum. This class more than any other taught me how to produce high-quality software while navigating through learning curves, deadlines, and other challenges. The fact that I had no experience with Dart, Flutter, or Github Actions was definitely a setback initially, but it was very rewarding in the end since I came out with a lot of experience with new technology that not a lot of people have. This project also taught me the value of Agile processes and iterative development. After completing the project, I can say that a process such as waterfall would have been highly ineffective in our case since we wouldn’t have been able to account for new information or challenges on the fly. I learned that just diving into the code isn’t a good way to start a project, you definitely need to pick the right procedure or you’ll risk wasting a lot of valuable time.

Jeffrey Josol Reflection

As a whole I did enjoy working on this project using a fairly modern and new tech stack for the frontend and the backend pieces. Having developed full-stack apps previously both for personal projects and in a professional setting it was refreshing to experience this again with a group of

other developers to collaborate and lead. I have done Node.js in the past so I was familiar with the framework and was comfortable in creating features and functionality for the backend. The frontend was a new learning experience for me as I have never used the Flutter framework before but after writing some code on the frontend, I can say that I am a fan of it and especially for the fact that it's performant by converting to native android and iOS code. We first wrote the code then tests after but I now realize how beneficial it may be to have a TDD approach. For the CI/CD, I have experienced it previously with Jenkins and OpenShift but not with Github actions. I also liked the incorporation of coveralls in our CI/CD workflow for ensuring good code coverage. I do like how Github actions are built into the Github platform where we host our repo and all the tooling and custom functionality we wanted all work pretty well together. There were some quality issues in the main branch of code that could have been avoided if pull request processes were followed closely. Overall I do like the choice of technologies we used and the processes that we followed to help us produce our app.

Appendix

Dartdoc

In order to generate documentation for a dart/flutter project, simply run `dartdoc` from the root directory of the project. In our case, we had to include the source code for an external login package in order to run integration tests, so make sure you have the dartdoc_options.yaml that will exclude these files automatically. The command will create a new directory at ./doc/api/ that will contain documentation in the form of HTML files. You can view the files directly or you can start a server (ex. `python -m http.server`) from the directory in order to get the full functionality. A preview of some of the generated documentation can be found here.

JSdoc

For generating the documentation for the Node.js project, change into the src/server directory and run `jsdoc <path>/<filename>.js -d ../../doc/` to output each jsdoc into the doc directory in the form of HTML files. You can also view these files directly on your browser.

HTTP APIs

We documented our HTTP apis using the Swagger OpenAPI specification. We used the `swagger-jsdoc` inside the Node.js app for building and generating the HTTP api documents. An endpoint path has been separately created in our app to view these documents. You can view them here.

We also have used Postman which is one of the most famous REST clients used during the development of API. Find the Postman collection for the Finesse Nation APIs here. Import the collection to your locally installed postman application for making the API calls.

Dependencies

Frontend: Dependencies and version number can be found in pubspec.yaml

Backend: Dependencies and version number can be found in client(package.json) and server(package.json)