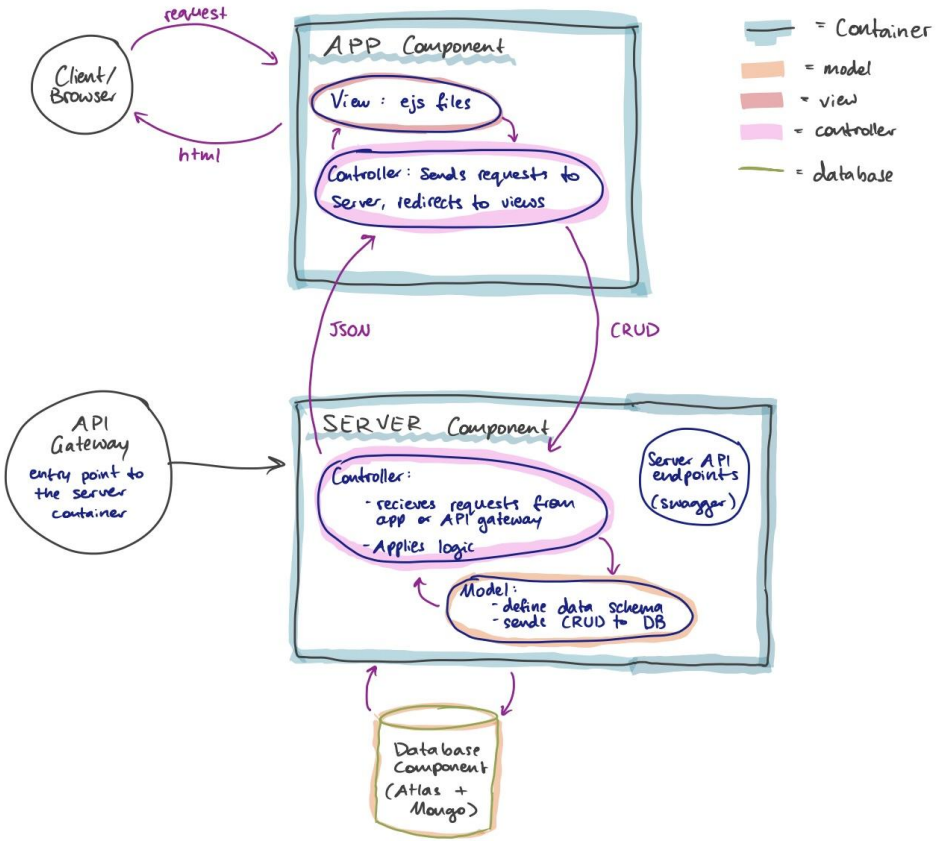
NWEN Final Presentation

Group Members: Serafina Slevin, Peter Walter, Billy Komene

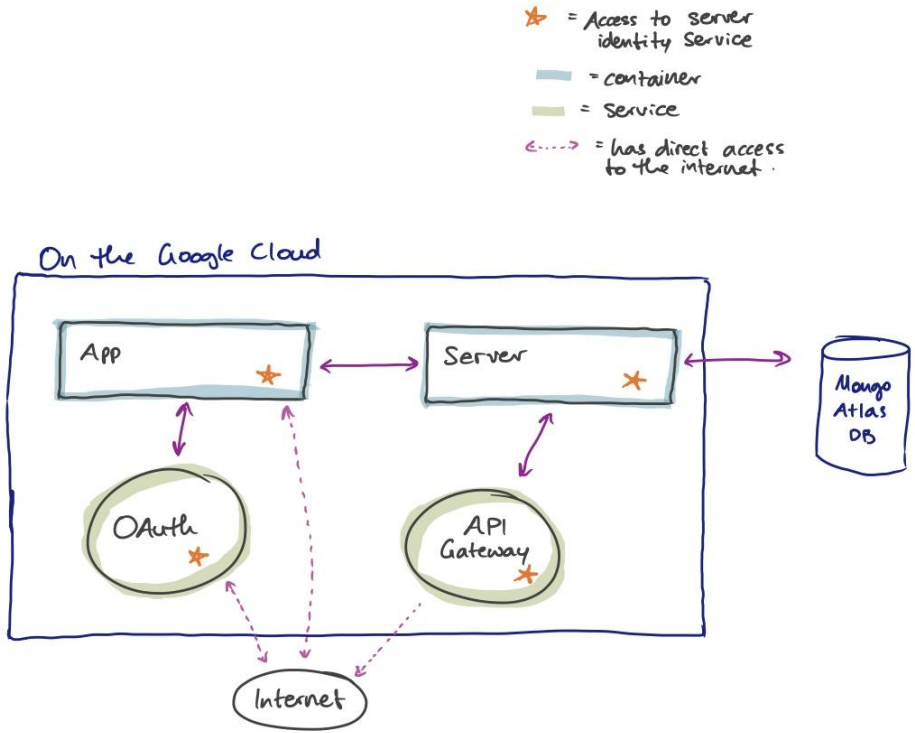
Architecture

Left: an overview of the architecture of our webapp. This shows how our various parts of work connect together and interact.



GCloud Architecture

- Cloud Run to host the frontend and backend containers
  - CI/CD deployment
- API Gateway Service
  - No API Key
  - Only specific GET endpoints
  - Directly accesses backend
- GCloud OAuth 2.0 API
  - Backend does not interact directly



MVC

We implemented a clear separation between model and view, using our two components app and server. View is in the app, with ejs pages being used to present different web pages to users. Model is in server, with our database structure being defined here.

There are controllers in both frontend and backend, which communicate between each other via API calls. In the backend, the controller is used to update the database as per requests from the controller in the frontend.

Client side rendering has been implemented for comments, in the file app -> public -> handlePostComment.js. When a comment is posted, this is rendered at the client side by editing the DOM. When a refresh occurs, the newly displayed web page will display the same comment, this time now from server side rendering.

Database

Foreign Keys:

- Post(boardName) -> Board(boardName)
- Post(username) -> User(username)
- Comment(username) -> User(username)

Schema:	Attributes:
user	username: {type: String, required: true, unique: true}
	email: {type: String, required: true, unique: true}
	password: {type: String, required: false}
board	boardName: {type: String, required: true, unique: true}
	boardDescription: {type: String, required: true}
post	boardName: {type: String, required: true}
	username: {type: String, required: true}
	content: {type: String, required: true}
	timestamp: {type: Date, required: true}
	title: {type: String, required: true, minLength: 3, maxLength: 50}
	comments: {type: [commentSchema], required: false}
comment	username: {type: String, required: true}
	timestamp: {type: Date, required: true}
	content: {type: String, required: true}

API + Dynamic Endpoints

- API Gateway
  - Direct access to the backend
  - Only exposes endpoints we deemed safe

- Endpoints
  - /boards/all
  - /posts/all
  - /posts/findByBoard?boardName={boardName}
  - /posts/{postID}

We have an internal swagger documentation which explains how the app component communicates/connects to the server components. This is in our codebase at the top level of the backend (server) as Server-API-Endpoints.yaml

We have dynamic endpoints in both the frontend and backend. These are used to dynamically load pages for a given board or post. The dynamic endpoints are detailed in the swagger documentation for the backend. Our frontend dynamic endpoints can be seen in app -> routes -> routes.js

Testing -> Mocha and Chai

POST Register a valid user

POST email contains whitespace

POST username starts with number

POST username contains special character

POST password does not contain symbol

POST password < 8 characters

POST password > 20 characters

POST duplicate user

POST duplicate email

POST create valid post

POST invalid jwt

POST empty boardName

POST empty username

POST empty content

POST empty title

POST title too short

PATCH edit valid post

PATCH edit non-existing post

DELETE delete non-existing post

DELETE delete existing post with wrong author

DELETE delete existing post

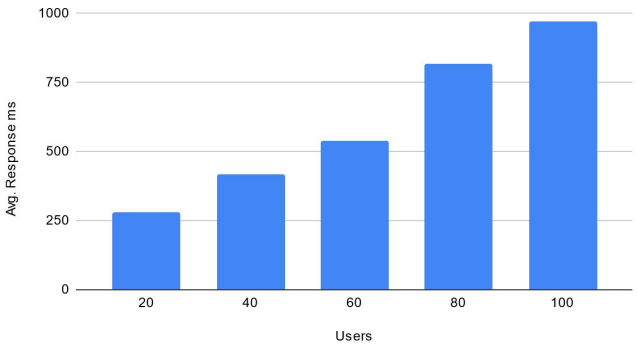
Testing -> Postman

- Frontend testing (user traffic stuff in postman)
- Frontend functionally tested with postman
- We ran out of free test runs in postman, so we've just included the ramp up to 100 on the final version of the project for this part (did have more results but they are on slower older version)

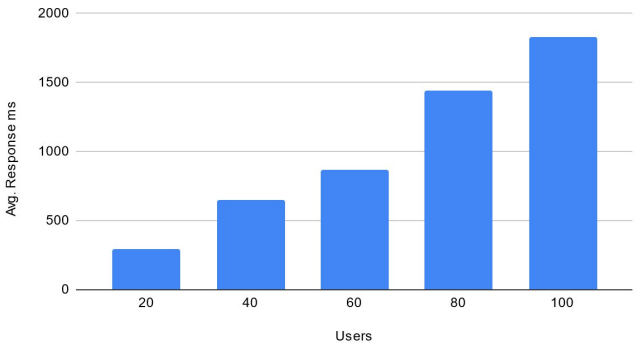
- See ramp up to 100 report -> this is what the following bar charts are based off

Testing -> Postman: Barcharts 1

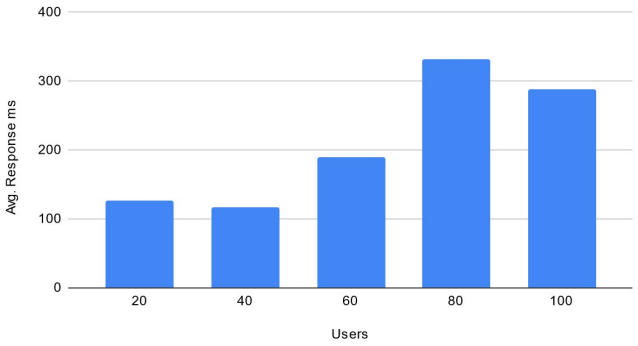
GET Board Page



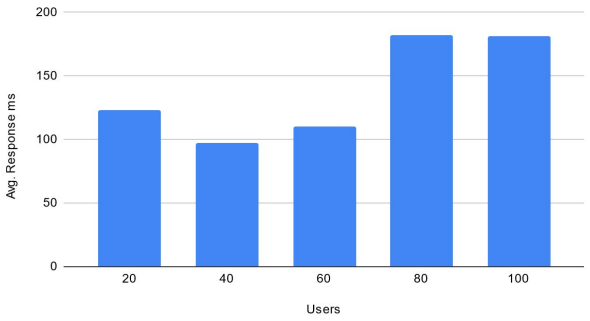
GET Home Page



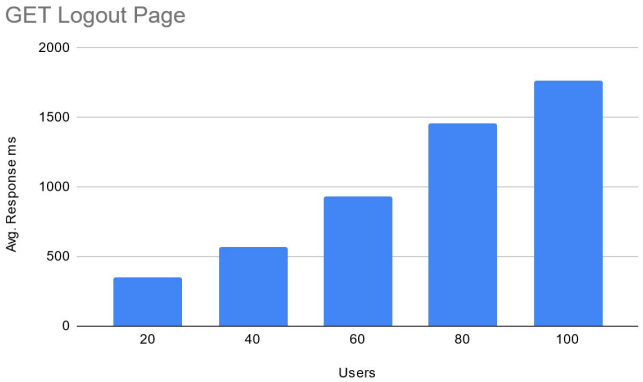
GET Register Page

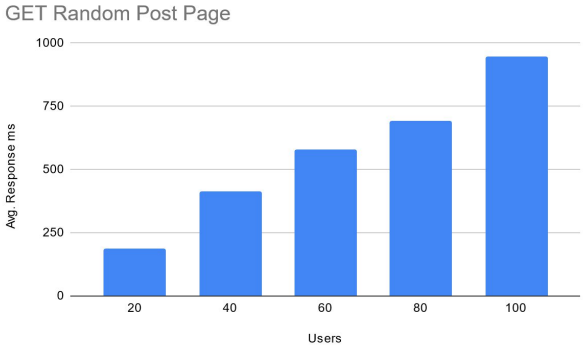


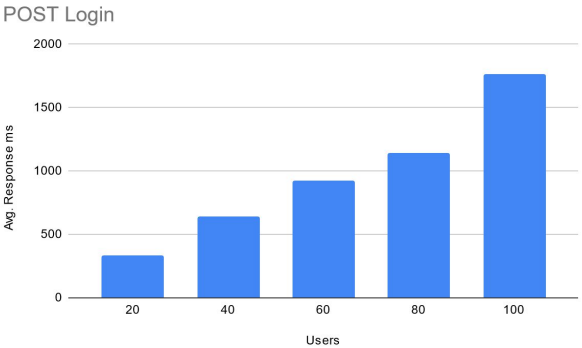
GET Login Page

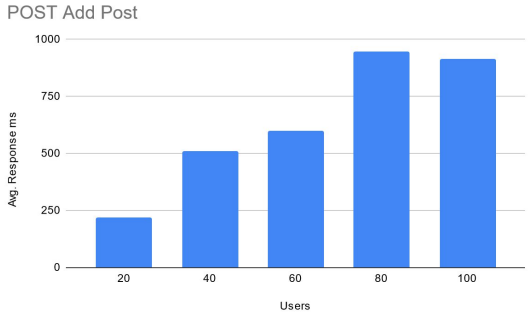


Testing -> Postman: Barcharts 2



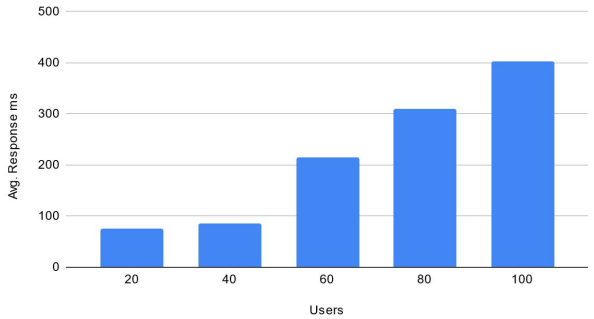




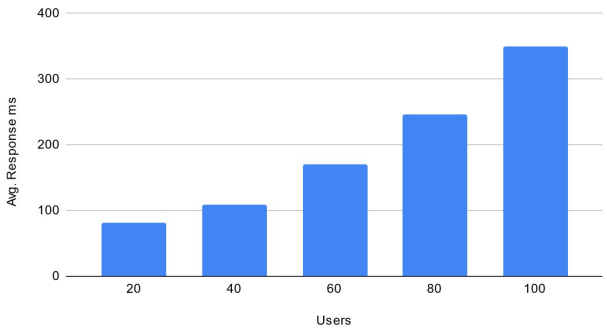


Testing API Gateway -> Postman: Barcharts

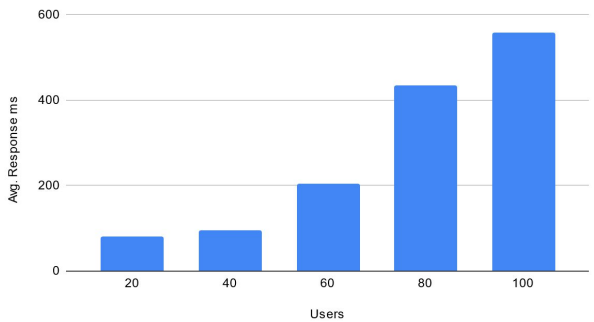
GET All Boards



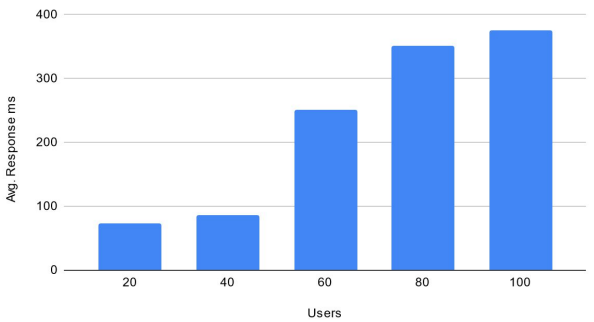
GET All Posts



GET Posts by Board



GET Post by ID



Privacy + Security

Security

-> We have implemented encrypted communication between the frontend and backend

-> There is no way to directly access the backend, you must use the API gateway

-> Our backend pre-hashes passwords

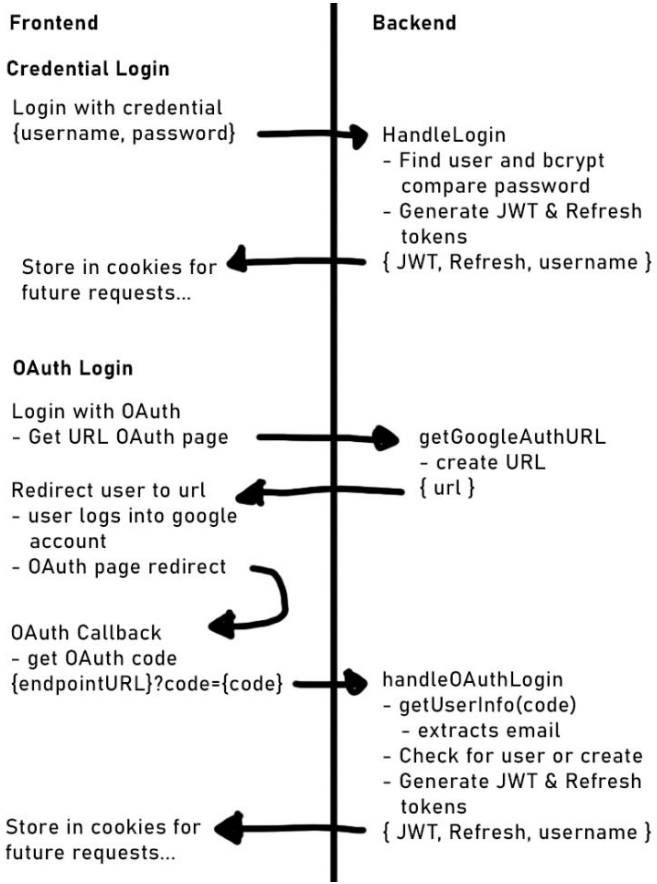
-> We used JWT to maintain access, due to their design if stolen they pose a security risk. We have mitigated this with Refresh tokens and recycling JWT tokens every 10 minutes.

Privacy

One concern that we still have is regarding OAuth. When a user is registered using OAuth, they are unable to set username, and so their email is used instead. This exposes their email to other users on the website.

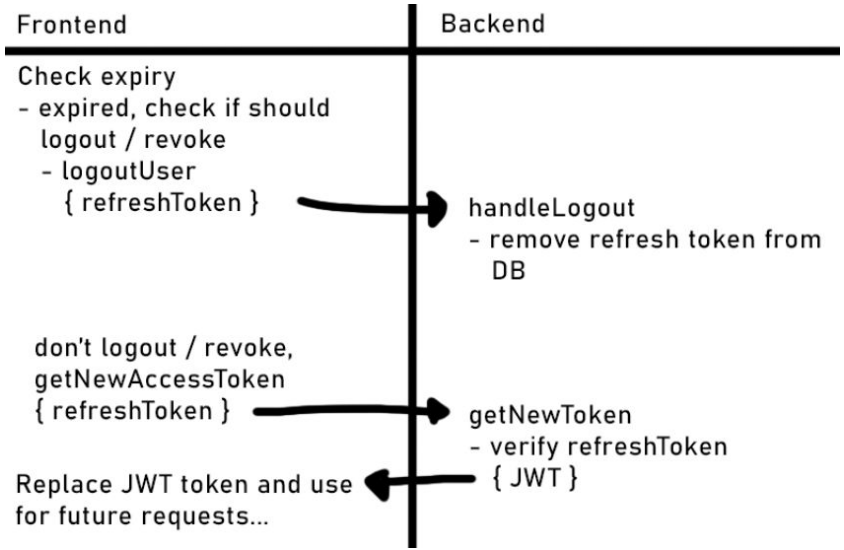
Authentication

- JWT Tokens
  - Last 12 minutes
- Refresh Tokens
  - Stored in DB
  - Can be revoked
  - Revoked after 20 minutes of inactivity
- OAuth
  - Google OAuth 2.0

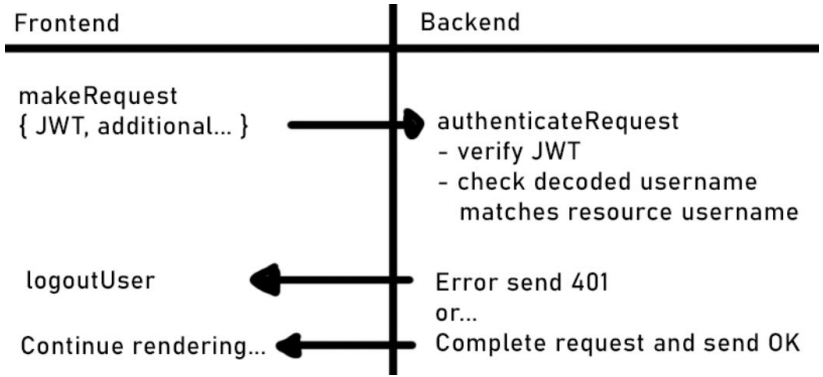


JWT Session Management

Token Refreshing



Protected Request Handling



Workload

Peter

- mocha and chai
- postman tests
- setup the database and atlas components
- implemented backend server logic
- API documentation

Billy

- OAuth implementation
- CI/CD
- authentication middleware on front and back end
- setup google cloud environment
- docker containers
- refresh tokens
- API gateway

Serafina

- client side rendering for new comments
- frontend CRUD implementation with ejs templating and css
- dynamic URLs & pages at frontend
- model schemas in backend