Resume Record

TaskBin: Serverless Project Board

This project was a serverless project board fully built on the cloud using AWS components. The purpose of this project was to provide users with a place for quick project management, with low complexity. Users can create project boards and invite other users to this board. On this board, members are able to manage tasks and the updates are sent to other members connected to the board in real time using WebSockets.

The components used were AWS Lambda, API Gateway, DynamoDB, Cognito, Amplify and Websockets. Cognito was used for user authentication, users needed to sign up with an email which would be used as their user id.
The frontend was built with React and Vite (veet). Then it was deployed on the cloud using Amplify.
The backend was built using Lambda, API Gateway, and DynamoDB. The API Gateway consisted of two api endpoints: one for websocket and one for the rest api. Each endpoint has its own lambda integration so when that endpoint is called, the corresponding lambda is invoked.
What was stored in the database: The database used a primary key and sort key structure. (PK, SK). Using (PK,SK) for dynamodb lookup gives much better lookup time in comparison to using just PK or SK. So for example when a board is created, these entries in the database are created:

- (BoardID, ‘METADATA’), (BoardId, userId), (userId, BoardId)

The way we implemented users joining a board, is creating an access code that is unique within the database and maps to the boardId. This code creates two entries into the database that have a time to live of 1 hour. Another user can join a board using the access code and it will create entries:

- (BoardId, userId), (userId, BoardId)

For real time updates, when lambdas are invoked and the action should be updated for the users connected to a board. The lambda sends the request to the websocket, which is received by the frontend with a websocket listener. The frontend can then propagate and render the changes for connected users.

For load testing, only tested a subset of the endpoints.

Type	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/boards/:id/tasks/create	5733	152	138.74	13	541	140.08	33.65	0.89
POST	/boards/create	1926	48	69.77	19	445	109.06	11.3	0.28
	Aggregated	7659	200	121.39	13	541	132.28	44.95	1.17

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/boards/:id/tasks/create	140	140	150	150	160	160	190	540
POST	/boards/create	68	71	75	80	88	94	110	450
	Aggregated	130	140	140	150	160	160	180	540

Failures Statistics

# Failures	Method	Name	Message
152	POST	/boards/:id/tasks/create	HTTPError('503 Server Error: Service Unavailable for url: /boards/:id/tasks/create')
48	POST	/boards/create	HTTPError('503 Server Error: Service Unavailable for url: /boards/create')

Most of the failures came from 503 Server error meaning thread pool exhaustion. We can solve this with increasing pool size, horizontal scaling, or reducing blocking calls.

[Github]

Bullets:

- Created infrastructure deployment and teardown automation scripts, significantly reducing AWS operational expenses during development
- Designed the API Gateway architecture (REST + WebSocket) with dedicated Lambda integrations to handle both synchronous and real-time operations
- Coordinated frontend-backend integration to ensure user inputs were correctly processed and reflected by backend services
- Devised a board invitation system with 1-hour TTL access codes, automating expiration and securing access

Questions to consider:

- How we handled cold starts and how to mitigate them
- DynamoDB access pattern and partition key design
- How you handled auth flows with Cognito
- Observability

NBA Season Prediction

This project was for my machine learning graduate course, working with a partner to create an NBA season prediction using real NBA data. We gathered 5 seasons worth of NBA team data using scripts. We trained a model for game outcome prediction for a game, the model would predict which team would win given the features for both teams. With the best trained model, which was logistic regression, we used it in a Monte Carlo simulation to simulate a whole season.

Created scripts to collect data from an NBA api (https://github.com/swar/nba_api). Collected 5 seasons of team data.

Created a lot of features to try to capture the dynamics between teams. Features included ratios between the home team and opp team such as off vs def. Also captured temporal trends so stats over the last 3, 5, 10 days for relevant stats. Win streak. Home vs away.

Each game had two rows of data, one for each teams perspective. We want to use two rows per game to make it an easier binary problem. Instead of giving two teams data and having the model decide which team is team A and which is team B and making it a bit awkward. Having each teams perspective of the game makes it easier for the model to decide if this team won relative to their opponents stats.

Trained different models on the first four season of NBA data. Tested on the fifth season. Logistic regression performed the best. Now that we have a model trained to predict NBA game outcomes given a team's features and opp features, we can use it in a simulation to create a season prediction for each team. The simulation would follow the order of the seasons schedule. Simulated games would be built on each other so temporal features would not have any data leakage. Simulation would be run 10,000 times, resulting in a distribution of 10,000 wins for each team. We would take the average of this distribution to be the predicted wins for that team in the season.

Notes:

- Include more dynamic features such as individual players features, injuries, location in hopes for a more accurate model

Bullets:

- Evaluated multiple classification models and identified Logistic Regression as the top-performing model with 67% game outcome accuracy, powering Monte Carlo simulations for season forecasts
- Built an prediction pipeline combining supervised learning and a 10,000-iteration Monte Carlo simulation to forecast the 2025 NBA season standings, yielding a mean absolute error of 9.7 wins

Questions to consider:

- Why do you think Logistic Regression beat the other models?
- How I prevented data leakage with temporal trends

Briefly

Briefly was a full stack web application that aggregates news articles and provides ai powered summarization for users.

Users would interact with our frontend. Our frontend nicely displayed different articles that we already have stored in our database. There are different genres for the users to go through. We have an admin role, so if logged in as admin you have the ability to edit and delete articles. These functions are hidden to regular users.

Users have the ability to generate articles. They input words to search by and the application takes the input and sends it to news api to get relevant articles. We limit the search to 5 articles as to not overload the application. Then the app takes the urls from the response from news api, and sends it to our scraper which will scrape the urls for content. This content is sent to gemini api for summarization. The summarization and article metadata are saved together in mongodb and sent back to the user to show what articles were chosen and summarized.

Users also have the ability to save articles which would add the article id to their likes array and there would be a section on the page where they can view all the articles they have liked.

What I did:
Webscraping using beautifulsoup and news api. News api didnt provide full article content, but it did provide url to the actual article. So collected article urls from news api, used beautifulsoup with the urls to scrape content. Created a dynamic webscraping solution that works for most sites, by looking for common html tags. If no tags are found then scrape the whole site. Scraped content is saved and sent to gemini for summarization. After summarization, the summarized content is saved into mongodb along with the metadata for the article. I set up mongodb schemas to store the articles in a consistent manner.

I connected these parts to create the pipeline to save summarized articles when users input a search in the front end.

I also created many rest api endpoints so that our frontend could communicate with backend to request necessary resources.

Questions to consider:

- How you evaluated summary quality?
- Bottlenecks in pipeline
- How you handled rate limits with NewsAPI or GeminiAPI
- Why MongoDB vs relational

Bullets:

- \resumeItem{Designed a MongoDB schema to efficiently manage user and article metadata across \textbf{2} collections, simplifying querying and feature expansion}
-

Quizify Bot

Created a music quiz discord bot. The bot handles hosting the quiz by being summoned to a voice channel. To be summoned, a public spotify playlist must be provided. The bot takes that playlist url and processes the songs. The songs are shuffled and a snippet of the song is played to the users in the voice channel. The users can then guess the song by typing into the discord chat.

MongoDB was used to keep score, once a game was finished the data is cleared from the database. Implemented a similarity algorithm so that approximate answers would be seen as correct.

Questions to consider:

- Race conditions and concurrency handling
- How state is isolated per server
- Performance issues with voice streaming
- Abuse cases (spam, disconnects)

Bullets:

- Integrated Discord-Player to manage music queues, play songs, and deliver synchronized audio to voice channels

ClipStream

A full stack web app to display twitch clips from followed channels in one place. A problem that I noticed on twitch was that there is no place to view clips from only channels you follow. I wanted to solve that problem by creating that place.

ClipStream | ReactJS, NodeJS, TwitchAPI, MongoDB Project Repo
• Created a full-stack web application that aggregates Twitch clips from channels that the user follows, streamlining
content discovery
• Constructed a REST API endpoints enabling dynamic data retrieval and seamless integration between front-end
and back-end systems
• Integrated JWT authentication, establishing an access control system to protect sensitive endpoints
• Optimized clip retrieval performance through asynchronous concurrent request partitioning, achieving a 60%
reduction in data fetch runtime

Network Intrusion Detection System

Personal RAG Agent (Retrieval Augmented Generation)

Stress Detection:

Data augmentation,
Leave one event out, leave one subject out validation

Relevant Coursework

Machine Learning, Wearable AI, Cloud Computing, Natural Language Processing, Data Structures and Algorithms, Operating Systems, Machine Org/Assembly,