CS 144 Final Project Paper - Triv.AI, a Live Multiplayer Trivia Game

Creators: Pablo Castro, Emanuel Zavalza

How we satisfied our spec items:

(1a) Canvas API:
We used the HTML5 Canvas API to create a dynamic and engaging UI element: the floating word grid. This allowed users to see themes animate and move fluidly across the screen during theme submission.

(2) Responsive Design:
The entire UI was built with a combination of Tailwind CSS and regular CSS, ensuring full responsiveness across screen sizes, from desktops to iPhones. Key components like input forms, game layout, and the leaderboard adapt gracefully on mobile.

(3,12) Offline Mode / PWA:
We implemented the app as a Progressive Web App (PWA). It includes a service worker that caches essential assets, allowing the app to load and display a fallback message when offline.

(4,9) HTTPS & Security:
The application enforces HTTPS via Google App Engine.

(5,15) Single Page App & Framework:
We used React to build a single-page app with smooth transitions between screens—lobby, theme entry, gameplay, and results—without any reloads.

(8) Authentication:
Users authenticate using Google OAuth. This enables session persistence, account personalization, and adds a layer of identity-based security.

(10,11) Backend & Database:
The backend was built using Node.js and Express, with MongoDB Atlas managing all game sessions, users, and leaderboard data. We used Mongoose as the ODM for data validation and schema management.

(14b) AI – OpenAI Integration:
The game uses OpenAI's GPT-4 API to generate trivia questions and answers based on player-submitted themes. This creates fresh and contextually relevant questions in real time.

(14c) Real-time Communication:
We used Websockets for real-time updates, including synchronized timers, player readiness, theme broadcasting, and score tracking.

(17) Deployment & CI/CD:
The project was deployed using Google App Engine.

Database Schema:

The project database is hosted on MongoDB Atlas, and it consists of two main collections. The Users collection stores essential user information, including a user's Google ID for identification, their username for web display, a boolean called ready that shows their status in a game, and finally, a collection of the top 5 scores for displaying a leaderboard in local gameplay. The second collection contains the room that manages active and past game sessions. The room stores a room code which is used to identify it, a list of players that have joined the room, a list of themes which players have inputted and will be used to generate questions, and also a map of scores that are mapped to each user's ID.

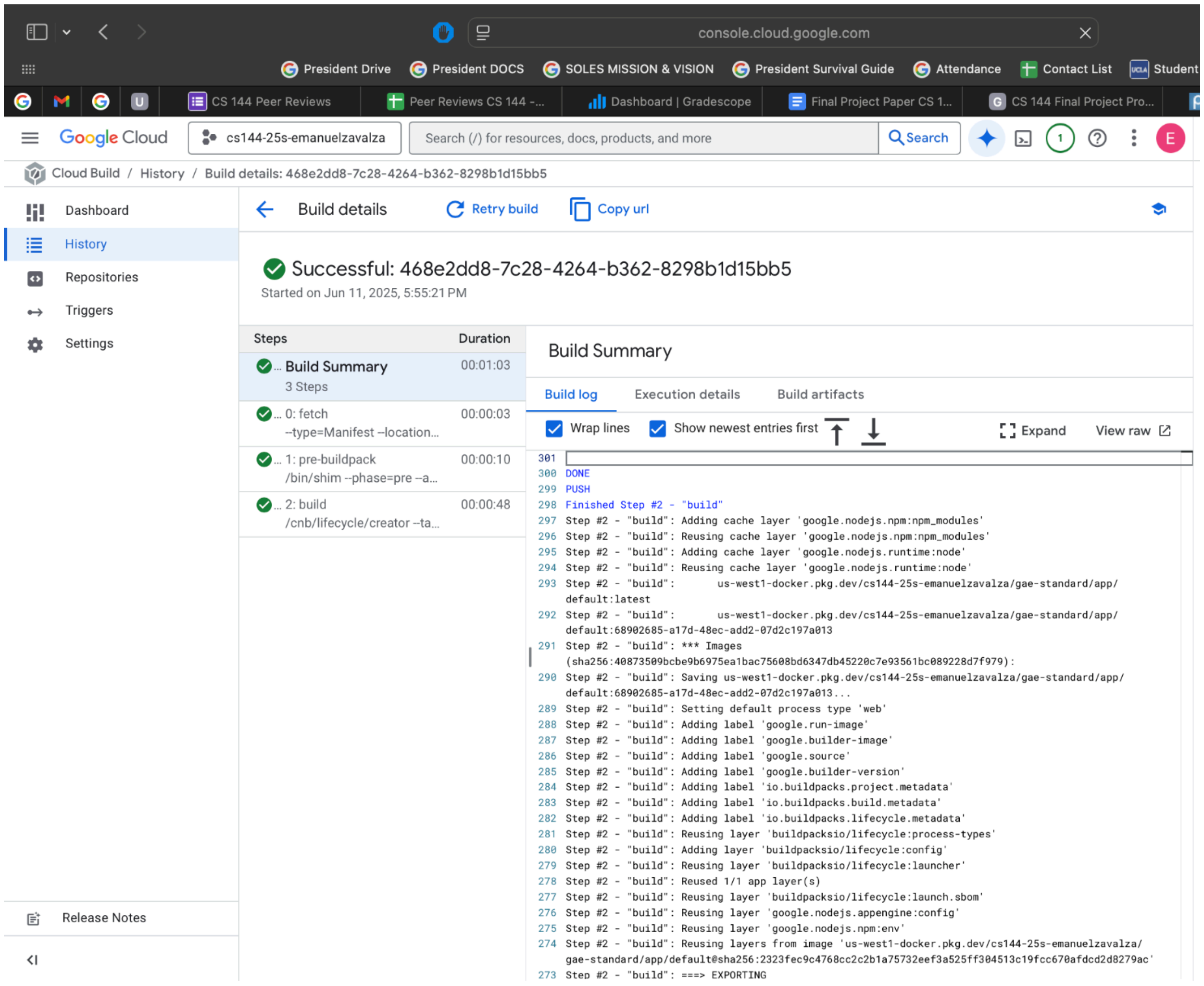
System Architecture Diagram:

The system architecture consists of a React frontend and an Express.js backend that communicate using WebSockets for real-time updates during gameplay. The backend handles all game logic and connects to a MongoDB Atlas database, which stores user information, game rooms, and scores. It also interfaces with the OpenAI API to generate trivia questions and multiple-choice answers based on themes submitted by players. This setup allows the app to support live gameplay, score tracking, and AI-powered question generation while maintaining fast and reliable communication between all components.

Security/Privacy Features:

The app keeps things secure by using HTTPS, which protects data as it travels between devices. Users log in with Google, so we don’t have to store any passwords. It’s also hosted on Google App Engine, which has strong protection against attacks like too many fake requests. Even though the app works offline for basic stuff, important actions only work when you’re connected to the internet.

CI/CD Log



Cloud Build / History / Build details: 468e2dd8-7c28-4264-b362-8298b1d15bb5

Build details — Retry build — Copy url

Successful: 468e2dd8-7c28-4264-b362-8298b1d15bb5

Started on Jun 11, 2025, 5:55:21 PM

Steps	Duration
Build Summary 3 Steps	00:01:03
0: fetch --type=Manifest --location...	00:00:03
1: pre-buildpack /bin/shim --phase=pre --a...	00:00:10
2: build /cnb/lifecycle/creator --ta...	00:00:48

Build Summary

Build log — Execution details — Build artifacts

Wrap lines [✓] — Show newest entries first [✓]

```
DONE
PUSH
Finished Step #2 - "build"
Step #2 - "build": Adding cache layer 'google.nodejs.npm:npm_modules'
Step #2 - "build": Reusing cache layer 'google.nodejs.npm:npm_modules'
Step #2 - "build": Adding cache layer 'google.nodejs.runtime:node'
Step #2 - "build": Reusing cache layer 'google.nodejs.runtime:node'
Step #2 - "build":       us-west1-docker.pkg.dev/cs144-25s-emanuelzavalza/gae-standard/app/
default:latest
Step #2 - "build":       us-west1-docker.pkg.dev/cs144-25s-emanuelzavalza/gae-standard/app/
default:68902685-a17d-48ec-add2-07d2c197a013
Step #2 - "build": *** Images
(sha256:40873509bcbe9b6975ea1bac75608bd6347db45220c7e93561bc089228d7f979):
Step #2 - "build": Saving us-west1-docker.pkg.dev/cs144-25s-emanuelzavalza/gae-standard/app/
default:68902685-a17d-48ec-add2-07d2c197a013...
Step #2 - "build": Setting default process type 'web'
Step #2 - "build": Adding label 'google.run-image'
Step #2 - "build": Adding label 'google.builder-image'
Step #2 - "build": Adding label 'google.source'
Step #2 - "build": Adding label 'google.builder-version'
Step #2 - "build": Adding label 'io.buildpacks.project.metadata'
Step #2 - "build": Adding label 'io.buildpacks.build.metadata'
Step #2 - "build": Adding label 'io.buildpacks.lifecycle.metadata'
Step #2 - "build": Reusing layer 'buildpacksio/lifecycle:process-types'
Step #2 - "build": Adding layer 'buildpacksio/lifecycle:config'
Step #2 - "build": Reusing layer 'buildpacksio/lifecycle:launcher'
Step #2 - "build": Reused 1/1 app layer(s)
Step #2 - "build": Reusing layer 'buildpacksio/lifecycle:launch.sbom'
Step #2 - "build": Reusing layer 'google.nodejs.appengine:config'
Step #2 - "build": Reusing layer 'google.nodejs.npm:env'
Step #2 - "build": Reusing layers from image 'us-west1-docker.pkg.dev/cs144-25s-emanuelzavalza/
gae-standard/app/default@sha256:2323fec9c4768cc2c2b1a75732eef3a525ff304513c19fcc670afdcd2d8279ac'
Step #2 - "build": ===> EXPORTING
```