Single Player Trivia Game

Light Specifications—Revision 3——Updated July 8, 2025

Description

This is an update to the previously submitted spec, which simplifies the planned app to a single-player, self-serve game, with questions imported via API. The previous, now-deprecated original spec is noted below for reference.

This project is a browser-based web application that allows users to play a self-serve, multiple-choice trivia game. It uses Flask as the web framework and the Open Trivia Database (https://opentdb.com/api_config.php) API as the question engine. The app automatically calculates the user's score.

The game app flow is as follows:

1. User visits the app homepage →
2. User selects option to start game →
3. App presents first question →
4. Player uses radio button to select answer →
5. User selects submit button to confirm answer →
6. App displays message indicating whether their answer was correct →
7. Repeat steps 3–6 until user completes all questions →
8. App displays user's final score →
9. User chooses to play again or exit

Development includes:

* Python libraries and modules used:
  * Flask - web framework
  * requests - to fetch API data
  * html - for cleaning up data from API
  * random - to randomize index order of correct answer for each question
* External APIs:
  * Open Trivia Database for Q&A data

User Tasks

There are three steps in the user journey:

1. ***Visit Homepage; Start Game***
A user visits the trivia game web app using a desktop or mobile browser. On the homepage, they see a welcome message and a button that let's them launch a single player, 13-question general knowledge trivia game. They press the button to play. (Behind the scenes, the app calls the Open Trivia Database API to fetch the question and answer data.)

2. ***Play Game***
The user sees the first question on the screen as well as their available options. They use a radio button to select the answer, after which they see a message as to whether they were correct or incorrect. They can also see their cumulative score. They are then automatically taken to the next question and follow the same process until all 13 questions. (Behind the scenes, app displays questions from the API one at a time after they've been cleaned. The user's answer automatically submits after they select a radio button, and the app determines whether the answer selected matches the correct answer. It then calculates the score and moves to the next question.)

2. ***Finish Game; See Results***
Once all the questions are answered, the user is redirected to a page that shows their results as a raw number of questions answered correctly and as a percentage of the whole. They can choose to return to the homepage to start a new game. (Behind the scenes, the game calculates the final percentage score based on the number of correct answers/the total number of questions. Based on the score, the app shows a specific concluding message.)

—

DEPRECATED
Multi-Player Trivia Game

Light Specifications—June 2025

Description

This is a browser-based web app that allows online and in-person events to host live, synchronous multiplayer trivia games. The app automatically calculates each user’s score and determines winners based on the number of correct answers.

There are two user types…

* Game administrators:
  * Can create perpetual login credentials
  * Can create one or more unique trivia games by uploading questions and answers
  * Can start and administer a live, competitive trivia game

  * Can end game, see scores, and download player details
* Players:
  * Can create single-game (non-perpetual) login using their name and email
  * Can play a live game by answering questions during an allotted time window
  * Can see their scores and competitor scores after the game has concluded

Development will include:

* Web framework: Flask with Jinja2
* Database: SQLite database via SQLAlchemy
* Other anticipated Python libraries:
  * Pandas for data processing
  * Flask-Login for user session management
  * Flask-WTF and WTForms for forms
  * CSV for processing question file upload and player stats downloads
  * Socket.IO for real-time game play
* External APIs:
  * Google OAuth for SSO game admin account creation option

User Tasks: Game Admin

1. ## ***Create Account***

(Note: I may skip this on the first round to focus more time on the game set up and play)

Kristen is an event director responsible for orchestrating in-person and online events. She wants to run a trivia game as a luncheon activity at an upcoming in-person conference.

She navigates to the app homepage and selects “create account.” This launches a signup form, where she enters her name, email, and password to set up a new user account. Alternatively, she can choose to sign in via Google to create an account.

Once she has created her account, she can choose to “log in” with her email and password.

Technical details:

* Assume this requires two page templates: 1\) page that presents “create account” and “log in” options, and 2\) actual sign-up form, which will be processed and data saved
* Requires Flask-Login for session management, and Flask-WTF and WTForms for forms
* Requires API for Google OAuth; will likely wait to do this until the second part of the build
* Will need to set up a database to save admin and game details; while I originally assumed I’d do this with JSON, based on further research into real-time, multi-player Python games, it appears a SQLite database via SQLAlchemy is the best option

* Will need to address data security and privacy compliance if I decide to deploy publicly; this is not something I will worry about for the first part of the build

Ideas for future feature expansion:

* Recover lost password via email
* Change name or password
* 2-factor password authentication
* Include additional required or optional fields
* Enable multiple users to access the shared admin team account
* Email or SMS communications opt-in

2. ## ***Game Set Up***

Kristen sees her admin dashboard page. She selects “create new game,” which launches a game details page for her new game.

Kristen enters the game name, date, and description, and then selects “upload questions.” She chooses a CSV file that contains the game questions and answers for each question. It also indicates which answer is correct. She confirms her upload and saves her game.

She can now see all her game details and edit details prior to starting the game. She can also upload images for specific questions.

Technical details:

* Assume this requires three additional templates: 1\) admin dashboard, and 2\) game detail form, where key details are added or edited, and CSV file is uploaded, and 3\) a version of the game detail page that displays the full details, including uploaded questions, but doesn’t allow for editing
* Requires Flask-WTF and WTForms for forms, CSV for processing uploaded files, and SQLAlchemy for storing game details in the database

Ideas for future feature expansion:
(*Note: if time allows during the second round, I will consider tackling the first two*)

* Add or delete questions without uploading new file
* Enter questions via form rather than uploading a CSV
* Add image to question
* Set timer to auto-advance questions
* Select option to weight questions
* Set a time for the game as a whole and end game at that time regardless of question
* Select option to randomize order of questions or answers

* Set the number of questions to ask from a larger pool of questions
* Upload questions via API from third-party question/answer generator
* Create a new game by duplicating a previous game
* Add game-specific header or page styling for live game player view
* Integrations via API with online event platforms or in-person event apps
* Enable multi-player teams
* Assign players to random teams

3. ## ***Start and Administer a Game***

Kristen is ready to start the game. She selects the correct game from the list on her admin dashboard. On the game details page, she selects “start game.”

The first question and its answer options appear on her screen. After 30 seconds, she manually selects “next question” to advance all players to the next question on the list. She follows this same pattern for the remaining questions. On the last question, she can “end game.” Kristen sees the final game RESULTS in descending order by score.

Technical details:

* Assume this requires two additional templates: 1\) game play—admin view, and 2\) game results leaderboard
* Because this app needs to support real-time game play with multiple players (perhaps large crowds of 100 or more), it requires use of websockets to enable two-way communications between server and admin; without this, game-play may be delayed as players wait for their devices to fetch new data and/or players may need to refresh their screens to load new questions
* Requires Socket.IO (Python and JavaScript) for real-time game play (*AI disclosure: it’s been 15 years since I last wrote JavaScript, so I will likely be using the latest Claude model to assist with that*); also requires Pandas for processing player data to display on leaderboard

Ideas for future feature expansion:

* Preview game
* Restart game in progress or end game early
* Special “presentation” view—similar to PowerPoint—that allows admins to show players the current questions and answers (i.e., screen share or digital display) without displaying admin-specific controls

4. ## ***Download Player Data***

After ending the game, Kristen wants to download player details so that she can contact winners and distribute prizes. On the game-specific page, she can now select a DOWNLOAD button,

which allows her to download a CSV file with each user's NAME, EMAIL, FINAL SCORE, and ANSWERS for each question.

Technical details:

* Assume this does not require any additional page templates
* Requires CSV for downloading results file

Ideas for future feature expansion:

* Email report instead of downloading directly
* Email report to alternative addresses

User Tasks: Player

1. ## ***Join and Play Game***

Jason is an attendee at Kristen's conference. He uses his mobile device to access a link provided by the event organizers. He enters his preferred username and email to join the game.

The game admin hasn't started the game yet, so he sees a holding message on the screen. Once the game begins, he sees the current question and can select an answer using radio buttons. He can choose to "submit" to proactively save his choice; otherwise, the app will auto-save his answer when the game admin advances to the next question. He follows this same pattern for the remaining questions.

Important edge cases: 1\) Midway through the game, Jason accidentally navigates off the game page. When he returns to the game page, the game remembers him and allows him to pick up where he left off at the current question. 2\) Jason joins the game late. He starts on the current question and receives an "incorrect" score for any questions that closed before he joined.

Technical details:

* Assume this requires three page templates: 1\) simple holding screen shown prior to game start, 2\) add contact details to join game modal, and 3\) game play—player view
* Requires SQLAlchemy, Flask-Login, Socket.IO, and Pandas to store player credentials throughout game, display questions in real-time, save each player's answers, and calculate scores
* Need to address data security and privacy compliance if I decide to deploy publicly

Ideas for future feature expansion:

* See full game progress bar

* See countdown timer for current question (requires “auto-advance” as admin feature)
* See leaderboard throughout
* Join team of multiple players
* Join game using Google SSO
* Join via SSO using API connection to third-party event platform or app

2. ## ***See Results***

Once the game ends, Jason sees the RESULTS view. This includes three main areas: 1\) a hero area congratulating the TOP SCORERS, 2\) a LEADERBOARD showing all users/scores in descending order by score, and 3\) a list of all the CORRECT ANSWERS.

Technical details:

* Assume this requires one-page templates: 1\) results display, including leaderboard and questions/correct answers
* Requires Pandas to process and display data

Technical Flow: Game Admin

1. ## ***Account Creation***

* User inputs name, email, and password (all required) on form, and selects “create”
* If data for any field is missing, app alerts user to error
* App checks that email is in a valid format; if it is not, alerts user to update
* App checks that email is unique; if not, alerts user that account already exists and they should log in
* App checks that password conforms to rules and that second entry is the same as the first; if not alerts user to specific error
* If all is correct and as expected, data is accepted and saved to database
* User is automatically logged in and redirected to admin dashboard

2. ## ***Account Log In***

* User inputs email address and password on form and selects “log in”
* App checks database for an admin account associated email; if it exists, checks that password is correct for that account
* If either is incorrect, alerts user of error
* If correct, user is logged in and redirected to admin dashboard

3. ## ***Create Game***

* User selects “create new game” on dashboard and is redirected to game creation form

* User inputs game name (required), game date (optional), and game description (optional)
* User selects “upload questions”; user navigates to the location of the CSV file with their questions and answers, selects file, and confirms “upload”
* App checks that file is a CSV; if not, alerts user to select another file
* User selects “save”; app reads question/answer data from the CSV file, and saves the game name, date, description, and question/answer data along with a unique game ID in the database under the game admin’s account

4. ## ***Edit Game***

* User selects “edit” from game detail page; user is redirected to game creation form pre-filled with existing game details
* User clicks into any form field to change contents; user can also upload new CSV to replace current questions/answers
* User selects “save”; app updates the data fields for the game ID in the database
* Admin get a unique URL to provide to players so they can navigate directly to game

5. ## ***Administer Game***

* User selects “start game” from game detail page; app saves the date and time live game was started to database
* User is redirected to the game play page; app pulls the question and answer options for the first question from the database and displays them on the page
* Page also indicates “question X of Y”, which updates X sequentially by 1 with each new question
* User selects option to move to “next question”; app loads the next question from the database as it did with the first question
* App pushes this new question to users
* Once user reaches the final question, the option changes from “next question” to “end game”
* Once game is ended, app calculates results; game admin sees data as a leaderboard of all users who joined the game at any time, listed in descending order by final score
* Results are saved to the database under the user’s account for one week, after which they are automatically deleted

6. ## ***Download Results***

* From the game detail page, user selects “download player report”
* App pulls player username, email, and final scores, and writes CSV file listing players in descending order by score
* Game is saved to user downloads folder as “player-score-report_MMDDYY” with MMDDYY being date game was played

Technical Flow: Player

1. ## ***Join Game***

* Player navigates to game's unique URL
* Player signs into game with preferred username (required) and email (optional), and selects "play"; these credentials are only good for this specific game (does not create a perpetual account like game admin)
* App checks that username is unique; if it is not, alerts user to choose a different name
* App checks if email is entered; if it is, checks that it is in a valid format; if it's not, alerts user to error
* If everything passes inspection, user is redirected to game play view; if not, user cannot see game once it begins
* Player details are added to the database for the specific game ID; going forward, all the player's answers will be associated with their unique player username in the database

2. ## ***Play Game***

* Admin starts game, immediately pushing the first question in database to all players; player selects their answer using a radio button
* Player can either select "save" to confirm answer selection or wait until the admin advances the game to the next question; in both cases, the answer corresponding to the selected radio button is saved for that question
* App scores their answer to the question: correct \= 1, incorrect \= 0, no answer \= 0; the progressive score is tallied and saved to the database
* If user joins game late, questions asked before they joined are scored as "no answer"
* App uses JavaScript to temporarily save game credentials and load game in progress in the event a user's page reloads or they accidentally navigate off the page

3. ## ***Play Game***

* After the last question ends, player is redirected to the results page
* App tallies each player's answers to determine final scores and updates the database; app sorts users in descending order by final score
* Users with the top five highest scores are displayed in a congratulatory hero at the top of the results page; all users who joined the game and their scores are displayed on a full leaderboard on the lower left side of the page
* The app pulls all the game's questions and their correct answers from the database and lists them on the lower right side of the page