Revised Project Proposal:
Isshoni Sudoku

Created By **KERMit**
Emily Jin (ej2332), Meg Frenkel (mmf2171),
Riddhima Narravula (rrn2119), and Kundan Guha (kg2632)

Advanced Software Engineering - Fall 2020

0. Meeting with IA Mentor

We met with our IA mentor, Minxuan, during class on Thursday, October 29. We also shared our revised proposal and received additional feedback on November 1. We received feedback on our authentication scheme on November 5. All changes to the proposal based off of IA mentor feedback are highlighted in blue throughout the report.

I. Introduction

(a) About

Our project aims to create a web-based Sudoku puzzle platform using React (frontend) and Flask (backend). While other Sudoku puzzle applications provide puzzles that can be completed individually, our application allows users to create player groups and work on Sudoku puzzles collaboratively as a team. This multiplayer Sudoku platform can be used by anyone with an internet connection and a browser who is interested in collaborative games with friends and family.

Here is the general flow of the application:

- New users will be asked to register an account with a unique username and password. Users will then be able to login to the application using this credential (see more on login approach and timeout below).
- Once users are logged into the platform, they will be able to start a new Sudoku game with a specified difficulty level (easy, medium or hard), and invite up to three players to join the puzzle.
- Once the game has started, player(s) in the game will be able to click on individual positions within the puzzle and provide a number (0-9) for that position. Users can remove or change their entries on the puzzle at any point in time.
- Once a player is adding in a number, their square is highlighted in a specific color corresponding to their user. This enables other players to see where they are currently making a move. There will be a legend next to the puzzle indicating which color corresponds to which player.

- When one player is currently editing a square, other players will not be able to edit that square. For example, when another player clicks on the square, they will not see any options to place the number.
- When the Sudoku puzzle is completed, every player for that puzzle receives a score reflecting the level of difficulty of their completed puzzle (i.e., completing harder puzzles results in a larger score). Overall scores are reflected in the application's "leaderboard," which will show the top rankings players, ordered by their *total* player score from all puzzles they have completed, either as a team or individually.
- Once a game is completed, teams can continue playing together in another game if they choose.
- Users can be a part of multiple games at the same time.
- During the puzzle, users will be able to participate in a live chat with other users who are a part of the puzzle to enable better collaboration.

(b) Persistent Data

Registered users (username, etc.), puzzles (with their scores and statuses),, and puzzle configurations (all submitted entries) will persist in a Postgres database. This means that if players close their browser or the application crashes for any reason, players should be able to return to see the latest configuration of their Sudoku puzzle game. Additionally, all prior puzzle history will be kept in the database in order to allow users to build their user score over time.

Our database will additionally store all conversations from the live chat during the game. This means that if one user is not active for a few minutes, they'll still be able to login and see/respond to the earlier chat between their collaborators. These conversations will be encrypted with a common key encrypted by each user's individual key to prevent database administrators from being able to see conversations.

(c) API

In order to generate new Sudoku puzzles and check to see if a puzzle has been completed correctly, this application will make use of the `py-sudoku` python library. This library generates and solves m x n Sudoku puzzles of specified difficulty. Each puzzle that is created will have a specified puzzle `id` and each player will have a `playerId` that the API will expect to receive when registering a move. This will allow players to be in multiple games at the same time.

(d) Authentication

Users of our app will be able to login to our application using their Google credentials. We will configure our app in the Google console to allow the retrieval of user information such as their first and last name and email. The Google login step will take place on the React side. An example of how this has been done is provided here.

To provide a stateless authentication mechanism for the backend API, all requests to the API will require the Google oauth access token in the request header. The API will verify the access token and use the verified token to authenticate the user making the request. A helpful resource outlining how to authenticate with a backend server can be found here. Another helpful resource is here.

~~To provide a stateless authentication mechanism for the backend API, this application will use JSON Web Tokens (JWT) using built-in Flask functionality and support from the `flask-praetorian` library. When a user provides their username and password on initial login, the backend server will create and send back a JWT to the client. The client will store the JWT and include it in the header of every subsequent request to protected endpoints. Meanwhile, the server will validate the JWT with every request from the client. Note that an expiration time for the JWT is easily configured using the `JWT_ACCESS_LIFESPAN` Flask configuration setting. Once the JWT expirest, the user will need to login back into the web application.~~

~~An example React and Flask application that has used this approach can be found here.~~

(e) Demo

Because this project is a web app, it will be straightforward to demo via a screenshare of a browser window. To begin the demo, one group member will login to the Sudoku application and start a new “easy” Sudoku game. That player will enter numbers into the puzzle and complete it. Upon completion, the player will go to the Sudoku app’s leaderboard and show that his/her username is listed on the leaderboard with the points associated with the puzzle s/he just completed.

Next, that player will invite the rest of our group members to join a new “hard” Sudoku game. The rest of our group members will login and join the game, and the demo will include each player entering numbers into the puzzle. One player will log out, while the other three continue playing. When the player who had logged out logs back in, s/he will see the progress that the rest of the players have made on the puzzle, and s/he will resume filling in numbers. Upon completing the puzzle, we will demo how each player in our group is provided with the score associated with the puzzle, and the player who had also initially completed the individual puzzle has the most points and is listed as first in the Sudoku app’s leaderboard.

Part 2: User Stories

1. As a player, I want to be able to fill in the sudoku so that I can solve the puzzle. My conditions of satisfaction are:
    a. I need to be able to start a new game from scratch
    b. I need to be able to place number and delete numbers throughout the game
    c. I need to be able to complete the sudoku puzzle and see that I have “won”

2. As a player, I want to be able to collaborate on a puzzle with another player so that I can have fun with my friends and family. My conditions of satisfaction are:
    a. I need to be able to invite another player to play in the game
    b. I need to be able to see any numbers that another player enters into the grid
    c. I should be able to delete the other player's move if I would like to change it
    d. Only one player can edit a cell at a time
    e. Once we have completed the sudoku puzzle, I need to see that we have "won"
3. As a player, I want to be able to save my current progress so that I can resume a game at a later time if need be. My conditions of satisfaction are:
    a. If I am playing the puzzle by myself and I log off, then I will see the puzzle as I left it when I log back in
    b. If I am playing the puzzle with my friends and I log off, but they are still working on it, then I will be able to see any progress they made while I was away when I log back in
    c. If I have multiple games in progress, I need to be able to select which one to resume when I log back on
4. As a player, I want to be able to see the leaderboard so that I can compare my puzzle solving scores with other players. My conditions for satisfaction are:
    a. If I complete a puzzle by myself, I am awarded the points associated with that puzzle, and my updated total puzzle solving score will be reflected on the leaderboard
    b. If I complete a puzzle with my friends, each member of our team is awarded the points associated with that puzzle, and each player's total updated puzzle solving score will be reflected on the leaderboard.
    c. I need to be able to see the leaderboard ranking with all players listed in descending order from most points to least points
5. As a player, I want to be able to chat with other players so that we can work together to solve the puzzle. My conditions of satisfaction are:
    a. At any point, I should be able to send a message via the chat box to all other players participating in the puzzle.
    b. If someone sends me a message, I should be able to see their message in the chat box section.
    c. I should be able to easily determine who sent each message in the chat box.
    d. If I logout of the puzzle application while other members are playing, I should be able to login and see any chat history from while I was away.

<u>Part 3: Acceptance Testing</u>

We will conduct acceptance testing at the user-level to ensure that all common and special cases for each of our user stories are covered.

User Story #1 (Playing sudoku individually):

- A player should be able to start a new sudoku puzzle from scratch.
    - Sample input: The id of the player and the id of the puzzle they want to play

    - Result: We would expect a new puzzle to be created in the database for that player and for the player to see an updated UI displaying the puzzle chosen.
- A player should be able to place numbers within any non-fixed cell on the board.
    - Sample input: The id of the player, the id of the puzzle they are currently playing, a cell on the current board and a number to place
    - Result: The move should be stored within the database and the player's view should update to show the move.
- A player should not be able to place numbers within any fixed/given cell on the board.
    - Sample input: The id of the player, the id of the puzzle they are currently playing, a cell on the current board and a number to place
    - Result: The move should be evaluated to be invalid and the player's view should not update to show the move.
- A player should be able to delete numbers from occupied cells on the board.
    - Sample input: The id of the player, the id of the puzzle they are currently playing, a cell on the current board
    - Result: The board stored within the database should reflect this move and the player's view should update (with the cell cleared)
- If a cell currently has a number, a player should be able to replace the number within the cell with another number.
    - Sample input: The id of the player, the id of the puzzle they are currently playing, a cell on the current board
    - Result: The move should be stored within the database and the player's view should update to show the move.
- A player should be able to complete a puzzle.
    - Sample input: The id of the player, the id of the puzzle they are playing (this should be triggered by the user using the UI to indicate they are done with the game)
    - Result: The UI should check whether the puzzle has been completed correctly and display the status to the player. If completed, the player's score should be updated and the puzzle should be marked as complete for this player.
- A player should be able to see if they correctly solved the puzzle after submitting.
    - Sample input: The id of the player, the id of the puzzle and the player's moves
    - Result: The UI should indicate if the player completed the puzzle correctly.
- A player should be able to continue a game if they have incorrectly solved the puzzle.
    - Sample input: The id of the player, the id of the puzzle and the player's moves
    - Result: The player will be able to continue to make moves based off of the state of the board from before they submitted their solution.

User Story #2 (Playing sudoku collaboratively):

- When a player starts a collaborative puzzle, they should be able to invite up to 3 other players using their usernames.
    - Sample input: Player id and usernames of other players

    - Result: Other players should be able to view the puzzle they are invited to and join. Information about the team and the puzzle they are playing should be stored within the database.
- Each player should be able to place, replace or delete numbers from cells on the board, regardless of previous activity within the cell (i.e. Player 1 can delete a number than Player 2 previously placed).
    - Sample input: The id of the player, the id of the puzzle they are currently playing, a cell on the current board and if replacing/placing, a number to place
    - Result: The move should be stored within the database and all players' views should update to show the move.
- When someone else in the game is attempting to enter a number, the puzzle square is highlighted and all other players can see that the player is making a move.
    - Sample input: One player clicks on a cell, attempting to write in a value.
    - Result: All other players participating in the game can see the cell highlighted with the color corresponding to the player making a move until the player exits the cell and completes their submission.
- Players should not be able to place or delete numbers within any fixed cell (provided on the original game board).
    - Sample input: The id of the player, the id of the puzzle they are currently playing, a cell on the current board and a number to place
    - Result: The move should be evaluated to be invalid and all the players' views should not update to show the move.
- Only one player can edit a cell at a time.
    - Sample input: Player id (using UI, player clicks on a cell)
    - Result: While one player is editing a cell, other players should not be able to click on the same cell and make edits.
- Every player in the puzzle game should be able to complete a puzzle.
    - Sample input: The id of the player, the id of the puzzle they are playing (this should be triggered by the user using the UI to indicate they are done with the game)
    - Result: The UI should check whether the puzzle has been completed correctly and display the status to all players in the game. If completed, all players' scores should be updated and the puzzle should be marked as complete for all players.
- Every player in the puzzle should be able to see if they correctly solved the puzzle after submitting.
    - Sample input: The id of the player, the id of the puzzle and the player's moves
    - Result: The UI should indicate if the players completed the puzzle correctly.
- Every player should be able to continue a game if they have incorrectly solved the puzzle.
    - Sample input: The id of the player, the id of the puzzle and the players' moves
    - Result: The player will be able to continue to make moves based off of the state of the board from before they submitted their solution.

User Story #3 (Resuming game):

- A player should be able to view puzzles they have not yet completed upon login.
    - Sample input: The id of the player
    - Result: The UI should show all the ongoing puzzles that a user has. Puzzles that have been completed should not be shown.
- When a player or team exits a game, all information about the current game must be saved within the database.
    - Sample input: player id, puzzle id, current moves
    - Result: Current moves and other puzzle information are updated in the database.
- When a player or team resumes the game, they should be able to view the most up-to-date game state, including moves the other team members have made while the player was away.
    - Sample input: Player/team resuming puzzle (player/team id, puzzle id)
    - Result: UI updates to show the game board with the most up-to-date state information for the game.
- If a player or team is logged out or the application crashes during the game, the game state including the last move executed must be saved within the database.
    - Sample input: player/team id, puzzle id
    - Result: Database is updated to contain puzzle information from before the crash.
- If a player attempts to make a move but is logged out (or if application crashes) before a request to an endpoint and the dispatch of the response, the move should not be saved within the database and the players' views should not include that move.
    - Sample input: player id, puzzle id, attempted move
    - Result: Database is not updated to include that move and the remaining players' views do not change. When the player who was logged out logs back in, their view should not include the move that they had attempted to make.

User Story #4 (Viewing leaderboard):

- A player should be able to view the leaderboard at any time.
    - Sample input: Player using UI to view the leaderboard.
    - Result: The player will be able to view the top players and their corresponding scores in descending order.
- If a player has completed at least one puzzle, their score should show up on the leaderboard.
    - Sample input: Player using UI to view the leaderboard.
    - Result: The leaderboard shown should include the player and their score.
- If a player has not completed any puzzles, their score (which is 0) will not show up on the leaderboard.
    - Sample input: Player using UI to view the leaderboard.
    - Result: The leaderboard shown should not include the player and their score since they have not completed any puzzles.
- If a player has completed multiple puzzles, then their total score from all the completed puzzles will appear on the leaderboard.
    - Sample input: Player using UI to view the leaderboard.
    - Result: The leaderboard shown should include the player and their total score.

- If multiple players have completed one puzzle together, each player should be assigned the points for that puzzle, and their score should be updated on the leaderboard
    - Sample input: Player using UI to view the leaderboard.
    - Result: The leaderboard shows every member who completed the puzzle with a score that includes the increased points from the completed puzzle

User Story #5 (Chat):

- Once a player joins a game, they should be able to see a chat window and send messages to other players.
    - Sample input: Player using UI to join game and view puzzle
    - Result: A chat window should show up alongside the puzzle. The player should be able to send messages at any point during the game.
- Chats sent during a game should show up chronologically on all players' chat windows.
    - Sample input: Player using UI to view group puzzle
    - Result: Chat messages should appear on each players' chat windows in the order that they are received by the backend and saved into the database.
- If a player attempts to send a chat message but is logged out before the request to an endpoint and the dispatch of the response, the chat message should not be saved within the database and the players' chat windows should not reflect that chat message being sent.
    - Sample input: Player attempting to send a chat message through the UI
    - Result: If the player is logged out before the response to the request is dispatched, the chat should not successfully send (and will not be saved in the database) and the other players' chat windows should not reflect the chat.
- If a player is logged out of the puzzle application while other teammates are still playing, they will be able to log back in and view the chat history from while they were away.
    - Sample input: Player being logged out of account during group puzzle
    - Result: After logging back in, the player can access the group game and view all chats sent during the game (including chats sent before the player was logged out and chats sent while the player was offline).

Part 4: Tools

Runtime: CPython
IDE: VS Code
Database management system/persistent data store: Postgres (SQL)
Frontend (JS/React):

- Package manager: npm
- Style checking: eslint
- Unit testing: React testing library or Jest
- Coverage tool: Jest
- Bug finder: jshint

Backend (Python 3.8):

- Package manager: pip
- Style checking: pylint and flake8

- Unit testing: pytest
- Coverage tool: coverage
- Static analysis / security / bug finder: bandit and mypy
- Documentation generation: sphinx