First Iteration: Isshoni Sudoku

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

Advanced Software Engineering - Fall 2020

Part 1. Github Repo

https://github.com/mmfrenkel/KERMit

Part 2: User Stories/Acceptance Testing

Our first iteration covers the following three user stories. These correspond to user stories #1, #3 and #4 from our original/revised proposal (respectively). Note that the second requirement of user stories 2 and 3 below have been removed, as the puzzle will not be completed collaboratively in this iteration.

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 "win" a sudoku puzzle game

2. 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 have multiple games in progress, I need to be able to select which one to resume when I log back on

3. 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. I need to be able to see the leaderboard ranking with all players listed in descending order from most points to least points

Revised Acceptance Testing Plan:

Our acceptance testing plan will cover the following series of tests, which are expanded from our original acceptance testing plan in our original and revised proposal. For each test, there are valid and invalid cases that require their own event sequence. All tests are conducted by a user interacting with the system as a "black-box." Our expected results are provided along with the actual results we obtained when running the acceptance tests as a "user" ourselves.

Test 1: Ensure that a user can create a new game from scratch (User Story #1)

- **Valid Case**:
    - Event Sequence: A logged-in user clicks on the 'new puzzle' button in the homepage to create a new game.
    - Expected Result: The user is brought to a new, empty puzzle page that is ready to accept entries.
    - Actual Result: Users are redirected to a new, empty puzzle page.

- **Invalid Case**: There is no invalid case for this test; the user cannot make any additional errors if they are only allowed to click a button to create the puzzle and cannot specify any invalid options.

Test 2: Ensure that a user can add numbers to the Sudoku board (User Story #1)

- **Valid Case**: Use adds a valid number within the Sudoku puzzle limits (i.e., any number 1-9 for a 9x9 puzzle) to a position that is not a static puzzle piece.
    - Event Sequence: A logged-in user who has already created a new 9x9 puzzle will click on any empty Sudoku puzzle cell and enter the number 2.
    - Expected Result: The board should accept the user's input without error. When the user clicks out of the Sudoku cell, the input should persist.
    - Actual Result: The board accepts the user's input and will now show the number that the user inputted in that cell. Even after clicking out of the cell, the number persists.

- I**nvalid Case:** User adds a negative number (i.e., -1) to a valid puzzle
    - Event Sequence : A logged-in user that has already created a new 9x9 puzzle clicks on any empty Sudoku puzzle cell and enters the number -1.
    - Expected Result: The board should not accept the user's input as it is out of the range of acceptable values for the board. The number is not saved in the board.
    - Actual Result: The number the user entered is not saved and will not show up in the cell after the user clicks out of the cell.

- **Invalid Case:** User adds a number too large for Sudoku board limits (i.e., 10 for 9x9 puzzle)
    - Event Sequence: A logged-in user who has already created a new 9x9 puzzle will click on any empty Sudoku puzzle cell and enter the number 15.

    - Expected Result: The board should not accept the user's input as it is out of the range of acceptable values for the board. The number is not saved in the board.
    - Actual Result: The number the user entered is not saved and will not show up in the cell after the user clicks out of the cell.

- **Invalid Case:** User adds 0 to a valid puzzle
    - Event Sequence: A logged-in user who has already created a new 9x9 puzzle will click on any empty Sudoku puzzle cell and enter the number 0.
    - Expected Result: The board should not accept the user's input as it is out of the range of acceptable values for the board. The number is not saved in the board.
    - Actual Result: The number the user entered is not saved and will not show up in the cell after the user clicks out of the cell.

- **Invalid Case:** User attempts to add a value to a static piece on the Sudoku board
    - Event Sequence: A logged-in user who has already created a new 9x9 puzzle attempts to click on a static puzzle piece (i.e., piece that is part of the original board and cannot change).
    - Expected Result: Despite their attempt, user cannot click into any static piece that they attempt to edit.
    - Actual Result: The user is unable to edit the cell containing the static piece.

Test 3: Ensure that a user can remove numbers to the Sudoku board (User Story #1)

- **Valid Case:**
    - Event Sequence: A logged-in user who has already created a new 9x9 puzzle and entered several values will click on one of the puzzle pieces that they have entered a value into and delete it.
    - Expected Result: The value should successfully be removed from the puzzle, as if no original value was submitted in that cell.
    - Actual Result: The value is removed and the cell remains empty even after the user clicks out of the cell.

- **Invalid Case**:
    - Event Sequence: A logged-in user attempts to click on and remove a "static" Sudoku puzzle piece (i.e., a piece that is configured with the original board).
    - Expected Result: The user is not allowed to click into a remove the "static" piece as it is a built-in component of the board. The user will not be able to successfully remove the item.
    - Actual Result: The user is unable to click into the static piece cell. Therefore, they are unable to edit the static piece and remove it.

Test 4: Ensure that a user is able to complete the Sudoku puzzle (User Story #1)

- **Valid Case**: User fills the puzzle board and the configuration is correct

    - Event Sequence: A logged-in user who has created a 9x9 puzzle enters in the correct values to complete the puzzle board
    - Expected Result: The game automatically alerts the user that the puzzle has been successfully completed, locks the board, and awards the user with the point value of the puzzle.
    - Actual Result: After a user fills in all cells correctly, the puzzle locks so that the user can no longer edit it and a message pops up indicating that the user has won. The puzzle is marked as completed and the user is awarded the point value of the puzzle.

- **Invalid Case**: User fills the puzzle board but the configuration is incorrect
    - Event Sequence: A logged-in user who has created a 9x9 puzzle enters in the *incorrect* values to fill the puzzle board and clicks on the “check puzzle” button.
    - Expected Result: The check runs and successfully determines that the puzzle configuration is not correct. The user is alerted to the fact that the puzzle is not complete because several of the submitted entries are incorrect.
    - Actual Result: Currently, we do not have a button to support checking for mistakes in the puzzle. Users will know that they have not finished the puzzle if they have filled all the cells but the puzzle does not lock, allowing them to continue to make edits and fix any mistakes. Also, the winner message does not show.

Test 5: Ensure that leaderboard reflects points associated with the users completed games (User Story #3)

- **Valid Case:** Upon completion of their first puzzle, a user clicks on the leaderboard and sees their point value listed as the point value of the puzzle.
    - Event Sequence: A logged-in user who has just completed their first 9x9 puzzle clicks on the leaderboard tab.
    - Expected Result: The user is able to see a list of the top-ranked players for the board, desc. Since the player has successfully completed a puzzle, they are able to see their own name and cumulative score on the leaderboard. Since the player has only completed one puzzle, their score should be equal to the point value of the puzzle.
    - Actual Result: Same as expected.

- **Invalid Case:** Test that the user does not see his/herself on the leaderboard when no puzzles have been completed.
    - Event Sequence: A logged-in user who has NOT completed a puzzle clicks on the leaderboard tab, expecting to see their name listed as a top-ranked player.
    - Expected Result: The user is unable to see their own name listed, because they have yet to complete in any puzzles. If this person is the first user, they should not see *anyone* listed on the leaderboard.
    - Actual Result: Same as expected.

Test 6: Ensure that leaderboard reflects points associated with top users (User Story #3)

- **Valid Case:** User clicks on leaderboard and sees players listed in descending order by their total point score
    - Event Sequence: User clicks on leaderboard after more than one game has been completed in the puzzle system. At this point, the user has successfully completed three puzzles.
    - Expected Result: The user is able to see a list of the top scoring players, including themselves. Their own score should reflect the correct *cumulative* score across all three completed puzzles, correctly.
    - Actual Result: Same as expected.

- **Invalid Case:** No players have completed any games yet. As a result, the leaderboard should be empty since no players have a score above 0. Note that there is no "invalid" case here, per say, just this edge case where all users have yet to complete any games.
    - Event Sequence: A logged-in user has just registered with the Sudoku puzzle application but has not completed any puzzles yet. All other users also have yet to complete any puzzles.
    - Expected Result: Upon navigating to the leaderboard page, the user sees a message indicating that no users have completed any puzzles yet.
    - Actual Result: Same as expected.

Test 7: Ensure that a user is able to logout/log in and resume game (User Story #2)

- **Valid Case:** User logs out and logs back in, seeing their puzzle exactly as they left it (non-collaborative case).
    - Event Sequence: A logged-in user starts a new 9x9 puzzle, enters a valid number in three random locations within the puzzle. The puzzle is not complete. The user then logs-out of the puzzle system, waits 10 seconds, and logs back in.
    - Expected Result: Upon re-logging in, the user sees the incomplete puzzle that they were working on when they logged out. Upon selecting that puzzle, they are able to see the puzzle board in the configuration that they left it in (all three submitted entries prior to logging out have persisted, with their correct value and original location).
    - Actual Result: Same as expected result.

- **Invalid Case:** There is no invalid case for this option. The user has no way to provide invalid input in a log-in/log-out scenario, except if they lose internet connection.

Test: Ensure that a user is able to login and select which game to resume (User Story #2)

- **Valid Case:** User logs in, creates puzzle and logs out, then logs in and resumes puzzle.

    - Event Sequence: A logged-in user creates three new Sudoku puzzles to work on. The user then logs out and logs back in.
    - Expected Result: Upon logging in and returning to the game homepage, the user is able to see all three newly created Sudoku puzzles. The user is able to click on any of the three puzzles and resume play.
    - Actual Result: Same as expected result.

- **Invalid Case:** There are no puzzles for the user to select yet; this is the case when they have just recently registered and have not played a game yet. Note that there is no "invalid" case here, per say, just this edge case where the user has yet to start any games.
    - Event Sequence: A logged-in user has just registered with the Sudoku puzzle application and has not created any puzzles to participate in yet.
    - Expected Result: Upon logging in, the user sees no Sudoku puzzles under their homepage.
    - Actual Result: The user does not see any Sudoku puzzles under their homepage but instead sees a message encouraging them to start a new Sudoku puzzle.

Part 3: Test Case Results

Frontend:

Our test cases for the frontend are run using a script located in our package.json file: https://github.com/mmfrenkel/KERMit/blob/main/app/package.json. Our test cases are all located within the app directory. We have split our tests up into many folders titled __tests__ in order to keep tests in directories that are close to the components that they are testing. Here are some examples of test files:

- https://github.com/mmfrenkel/KERMit/tree/main/app/src/__tests__
- https://github.com/mmfrenkel/KERMit/blob/main/app/src/puzzle/__tests__/Puzzles.test.js

Report Folder: https://github.com/mmfrenkel/KERMit/tree/main/reports/frontend

Example Unit Tests report:
https://github.com/mmfrenkel/KERMit/blob/main/reports/frontend/clean_frontend_unit_tests_report.txt

Example Coverage Report:
https://github.com/mmfrenkel/KERMit/blob/main/reports/frontend/recent_frontend_coverage_report.txt

Backend:

All backend tests are run using pytest. A bash script found in the project's /bin directory automatically runs the pytest and saves the output to the following test report directory, where you can find several samples of testing output:
https://github.com/mmfrenkel/KERMit/tree/main/reports/backend/tests

The bash script that runs the tests can be found here:
https://github.com/mmfrenkel/KERMit/blob/main/bin/run_backend_tests.sh

Example Unit Tests report:

- https://bit.ly/38Mkves (older - failing cases)
- https://bit.ly/38JCWR0 (newer - passing cases)

Example Integration Tests report:

- https://bit.ly/3pzt1U7 (older - failing cases)
- https://bit.ly/36BuEbi (newer - passing cases)

An example (final) coverage report for this iteration can be found here: https://bit.ly/3IM4KYB

Part 4: Automated Style Checker/Automated Bug Finder Results

Frontend:

Style reports are created by running eslint (see lint script in package.json). Since ESLint is a VSCode plugin, we were notified of and fixed many errors during development. Note that when there are no errors, there is no output so the report file will be empty. However, we've included some in the reports that we did not fix till recently:

- https://github.com/mmfrenkel/KERMit/blob/main/reports/frontend/messy_frontend_style_report.txt
- https://github.com/mmfrenkel/KERMit/blob/main/reports/frontend/recent_frontend_style_report.txt

Backend:

Backend style and automated bug checks were completed using pylint, a source code, bug and code quality checker for Python. Similarly to the tests, a bash script is used to automatically run the style/bug checks and save the output to a style/bug report directory, found here:
https://github.com/mmfrenkel/KERMit/tree/main/reports/backend/style_bug_checker

The bash script that runs these checks can be found below. This bash script runs separate pylint configurations for checking source code (more strict) and test code (less strict). Explanations on these configurations are provided below.

https://github.com/mmfrenkel/KERMit/blob/main/bin/run_backend_bugs_style_check.sh

Example Style/Bug Reports for Source Code:

- https://bit.ly/32Z4KNz (old -- dirty)
- https://bit.ly/32XlzbM (new -- clean)

Example Style/Bug Reports for Testing Code:

- https://bit.ly/2UE93JQ (old -- dirty)
- https://bit.ly/3kHNsuF (new -- clean)

Note that the script that runs the pylint tests has a few flags for disabling warnings. I outline why each flag is used below.

For the source code checks, all bug/style checks passed in the end, except for a few warnings related to cyclic imports. The cyclic import presence is known in setting up Flask projects, as most Flask documentation suggests re-importing models and routes at the bottom of your app configuration file in order to register the endpoints correctly. This approach is even supported in tutorials like the one linked below, which was created by the author of several leading books and tutorials on Flask web development.
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

For the test code, several other pylint warnings were disabled, largely because pylint doesn't seem to understand how the pytest framework works out of the box. Here are some more specifics on why each "--disable" flag was used:

- Redefined-outer-name: This warning happens when a test file imports a particular shared mocking function that lives in another file and then uses it as a parameter in the test function that needs that mock, as pytest requires. Pytest, in fact, requires that you pass the mock as a function argument if it is not automatically used, so this is not a problem.
- Unused-argument: In order to properly mock functions, we have to define functions that take arguments (*args) that would be used by the "real" function, otherwise the test fails. However, in the test setup, the mock function is never explicitly called; it's just plugged in as a mock. This means that pytest complains the arguments provided are not used, when they are, in fact, used when the test runs.
- No-self -use: In creating working Mocks, it is necessary that the mock function/method is able to properly mock the behavior of the real class; pylint here wants us to create static functions, but we cannot do this because they are not static in the real class.
- Too-few-public-methods: Here pylint complains because there may only be one public method in a Mock class. It is OK if a mock class only has the methods defined that it needs in order to satisfy the mock behavior, so we ignored this.

- Duplicate code in multiple files (R0801): The issue flagged here is multiple uses of a MockBaseQuery class in several test files that require a mock of the “real” SQLAlchemy base query class. Because each MockBaseQuery class is defined uniquely for each test it would be difficult to reduce duplicate code here.