Final Submission: Report

Checkmate

Introduction

Chess is a classical board game that has been played for centuries. In recent years, there has been a surge in popularity in chess that introduced many new users to the game. Many beginners are reluctant to immediately jump right into the action with another player without gaining a little bit of experience first. Therefore, this software project is a website that provides users with a platform to attempt various chess puzzles in a digestible and interactive way. The website, Checkmate, will contain a 2D 8x8 chessboard and presents users with a particular chess position, starting with relatively easy puzzles and progressively becoming more difficult as each puzzle is solved. Users are prompted to solve each puzzle by finding the best move in the given position. For each puzzle completed correctly, one point is added to the user's counter and the difficulty of the puzzle increases. If the user makes a single mistake on a puzzle, they are immediately taken to another puzzle around the same difficulty level as the previous puzzle they got incorrect. However, if the user makes three incorrect attempts in total, their progress is reset. When their progress is reset, the counter resets back to zero when they start the trial again. This encourages users to think critically and strategize before making their move. See figure 2 for an overview of this description.

The website will include a feature that provides a tutorial on how the chess pieces move and gameplay rules, which helps if a user is unfamiliar with rules or game piece movement. This is an optional tool at the user's disposal in the event that they need help. There will be many users with a wide range of skill levels, therefore it is important to cater to both beginners and masters of the game. See figure 3 for visual clarification.

As an extra layer of guidance, whenever the user taps on a piece they would like to move, there will be a series of dots that show the available legal moves the user can make with that particular piece. This will be a feature that the user can toggle on or off if they deem it necessary. Aforementioned, there has been a major increase in popularity in chess, therefore there are many newer players to the game. By adding this slight aid, it will allow users to not be discouraged to play due to them forgetting how the pieces move.

Additionally, the user is initially told which color's move it is. If the puzzle says "Black to move" or "White to move," then that is the color that the user must move with. In addition to this, to help the user understand the current position, the first move is automatically executed to show the user what the last move played by the opposite color. This allows the user to understand the context of the position and visualize the last action performed. The user will be challenged to "solve" the chess position by getting their color to win by checkmate in the allowed number of moves. This added goal encourages users to think critically and plan ahead to achieve checkmate.

Checkmate provides a fun and interactive way for users to improve their chess skills. The use of increasingly difficult, random puzzles matched with a user-friendly interface offers a unique and challenging experience for users. This project is an ideal tool for chess enthusiasts of all skill levels.

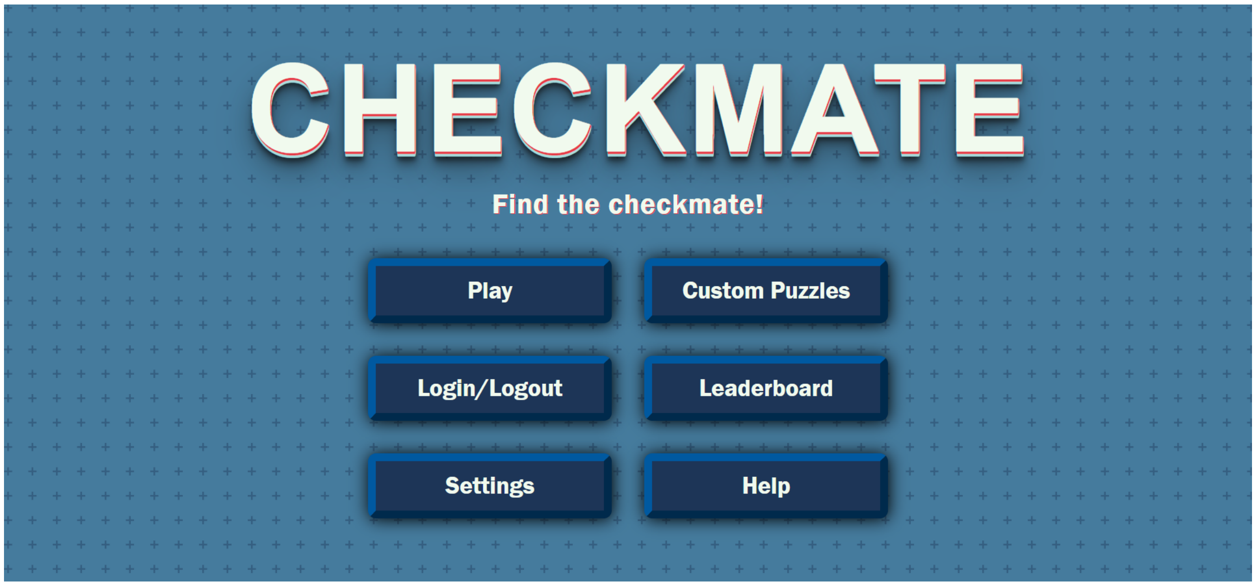


Figure 1. Checkmate's Main Menu Page UI. This is what users see when initially entering the website.

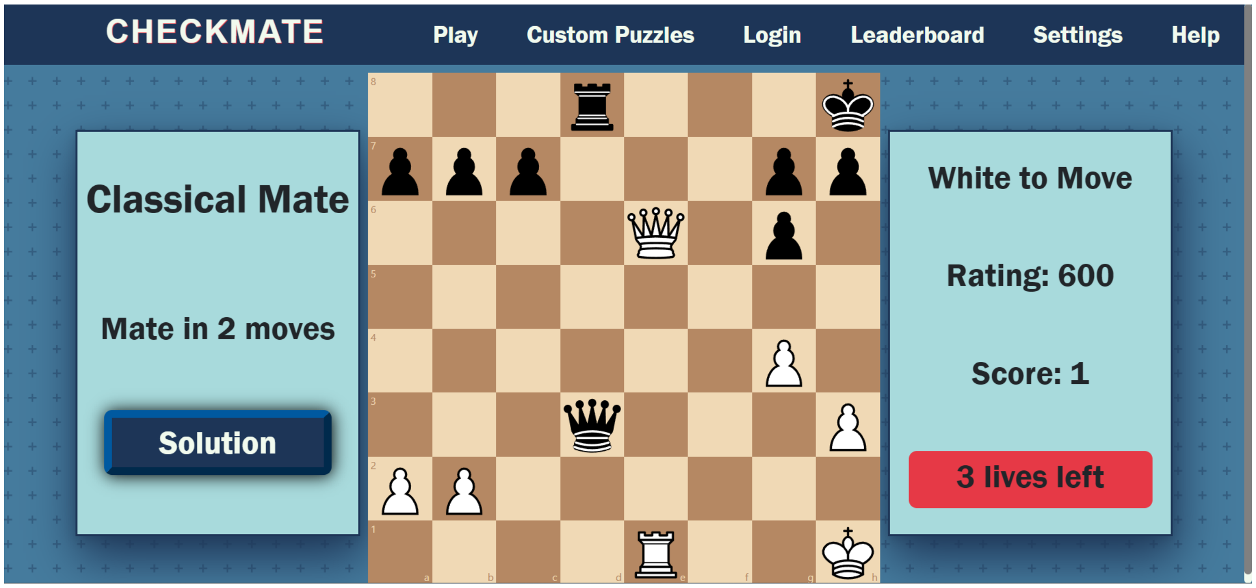
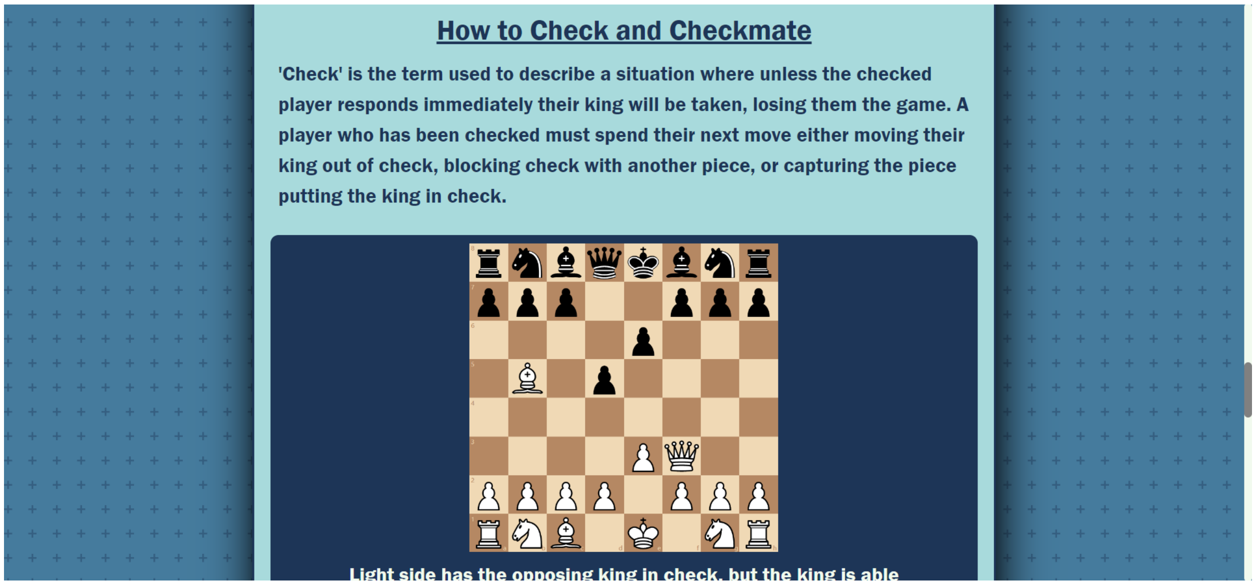


Figure 2. Checkmate's Classical Mode Page UI. This is what users see when they click start.


How to Check and Checkmate

'Check' is the term used to describe a situation where unless the checked player responds immediately their king will be taken, losing them the game. A player who has been checked must spend their next move either moving their king out of check, blocking check with another piece, or capturing the piece putting the king in check.

Light side has the opposing king in check, but the king is able

Figure 3. Checkmate's Help And About Us Page UI. Users can reference piece movement and rules here.

Technology

The technologies used for this project include React.js, HTML, CSS, Python's Pandas library, JavaScript's Chessboard.js library, the Lichess puzzle database, React's react-chessboard library, and Google Firebase. Checkmate was developed using Visual Studio Code and is housed on Github using Git as its version control software.

React.js was used for over 70% of the project's code base. It contains all logic for the system and bridges the HTML and CSS of the project. React.js was chosen early on over other similar technologies due to its user-friendly documentation and the familiarity developers of Checkmate had with React.js before starting the project. Although members were familiar with React.js, there was still a learning curve in order to execute a project of this scale and complexity. Despite this, all technologies used involving React.js were crucial to completing this project, and therefore made development easier in the long run.

Under that notion, libraries such as Chessboard.js and react-chessboard were used in order to provide functionality and user interface for the chessboard itself. The library react-chessboard provided pieces of logic broken down from an actual chess game and visual elements of it, but development of Checkmate entailed engineering these components to form a coherent, working chess puzzle website. Using a library such as react-chessboard was new to the members developing Checkmate, but documentation and tools provided ensured that coming up to speed with this library was possible. In addition, Chessboard.js provided some visual elements for Checkmate's custom puzzle mode. Without this library, the custom puzzle mode would have no spare pieces for a user to drag and drop onto the board to create their puzzle. Again, the documentation and tools provided ensured that members developing Checkmate could implement features under this library successfully and in a timely manner.

A crucial component to Checkmate is the Lichess chess puzzle database, a free to use database that contains over 3 million puzzles. Each puzzle has formatted metadata, which needed to be understood first by the developers, then filtered to the needs of Checkmate. In order to filter the number of puzzles down to a more workable number, Panda's python library was used. A python script was written that filtered out the unnecessary puzzles that were not relevant and left approximately 100,000 puzzles remaining. This made sure that unnecessary data that was not needed is not being stored and taking up extra resources.

Firebase is the main hosting platform for checkmate as well as its authentication system provider. The website is hosted via firebase using its free service and was developed with this technology in mind. The firebase open source authentication service was used in order to provide an easy way to allow users to log in and save their data. The GitHub for integrating this authentication can be found here: https://github.com/firebase/firebaseui-web.

Checkmate also uses HTML and CSS for information and styling purposes. These are core technologies to the project, and were selected because of their widespread usage when creating websites. Developers of Checkmate all had at least a base level of understanding as to how these languages functioned and worked together to create webpages, therefore the HTML and CSS of Checkmate was not a struggle to implement. Any problems that arose were aided by collaboration and widely available documentation. There was quite some time spent in the initial development in order to get CSS flexbox to cooperate, but once that was generally understood, the styling of other pages on the website was simpler. This is due to the fact that the design documentation outlined some pages similarly, and therefore allowed for code reuse for styling.

Please see figure 4 for a visual of the way these technologies make the foundation of Checkmate.

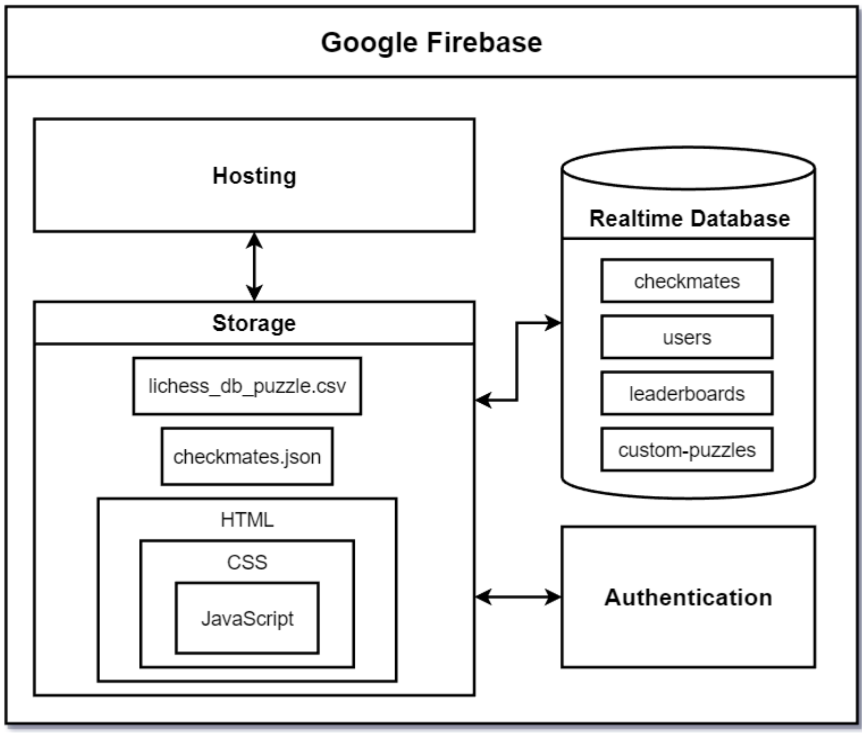


Figure 4. Checkmate's Block Diagram.

Design

The design of checkmate includes a lot of working parts. Included here is a breakdown of every part of the system. The aim here is to be as exhaustive as possible and include everything.

For the database portion of the site there are four different database sections (see figure 6 for visual):

checkmates: These include the information required to display and solve all of the chess puzzles and serve as the largest scale of storage in the project. These are the puzzles pulled from the Lichess open source database. These have the following fields:

- FEN: String data used to define the puzzles board state and the last move made
- Moves: String data that holds the solution to the given chess puzzle
- PuzzleId: A unique ID number for the puzzle. These numbers did not come from the Lichess database.
- Rating: A number describing the difficulty rating of a puzzle using the elo system. Generally, puzzles with higher values are intended to be more difficult.
- Themes: A string value describing what type of puzzle is being accessed (mate in 1, mate in 2, etc.).

custom puzzles: These include custom puzzles provided by users. These include the following fields

- fen: Identical to the “Checkmates” table.
- moves: Similar to the “Checkmates” table, this stores all of the moves in order to solve a puzzle. In this table, however, there are a series of strings, one for each move made, as opposed to one single string for each move.
- puzzleId: A unique identifying number for each puzzle
- username: The username of the account that submitted the puzzle

leaderboards: The high scores for each mode, this consists of classical mode data as well as bullet mode data. They contain the following fields:

- name: the username of the user that achieved the high score
- score: the score achieved
- time: The time it took said user to achieve that high score. This field is only present in the bullet mate leaderboard

users: The different users that sign up for the site have select data stored as well. The fields are as follows:

- <u>recentScores</u>: Holds recent scores achieved by users
- <u>userID</u>: Holds a unique ID for the user. This is not the same as a users username and is an assigned string
- <u>userSettings</u>: Holds setting information about a given user
- <u>userName</u>: The users chosen username

In addition to the database there are a large amount of javascript files with embedded html as well as a few html files. A breakdown of the project's important files is as follows:

- <u>documents</u>: documentation about the project. Specifics will not be elaborated upon here as none of these files are integral for the project to run.
- <u>node_modules</u>: storage folder for node based files. None of these should be edited or tampered with.
- <u>public</u>
    - <u>images</u>: Includes various images that are pulled for ui elements as well as the play board's pieces.
    - index.html
- <u>src</u>
    - Checkmate.js
    - firebase.js
    - index.js
    - <u>components</u>:
        - BulletUI.js
        - ClassicalUI.js
        - CreateUI.js
        - CustomPuzzlesUI.js
        - FailedPlayPuzzle.js
        - GameModeUI.js
        - GameOver.js
        - GameOverLeaderboard.js
        - HelpTutorialMenuUI.js

    - LeaderboardUI.js
    - LoginUI.js
    - MainMenuUI.js
    - Navbar.js
    - PlayPuzzleListUI.js
    - PlayUI.js
    - PuzzleSubmission.js
    - SettingsUI.js
    - SuccessPlayPuzzle.js
  - <u>styles</u>
    - BulletUI-style.css
    - ClassicalUI-style.css
    - CreateUI-style.css
    - CustomPuzzlesUI-style.css
    - FailedPlayPuzzle.css
    - GameModeUI-style.css
    - GameOver.css
    - HelpTutorialMenuUI-style.css
    - LeaderboardUI-style.css
    - LoginUI-style.css
    - MainMenuUI-style.css
    - Navbar.css
    - PlayPuzzleListUI-style.css
    - PlayUI-style.css
    - Settings-style.css
    - SuccessPlayPuzzle.css
- .gitignore

- package-lock.json
- package.json
- README.md

Class Descriptions:

Checkmate.js, firebase.js, and index.js are all located in the src folder and are a set of three different configuration files. Checkmate.js has a long series of includes used throughout the project as well as a series of routing paths for website navigation. firebase.js contains configuration for the firebase project that allows it to be hosted via firebase. index.js contains some includes and a single react function allowing the project to use react. All three of these files are pretty small.

Every file in the components folder of the file hierarchy has a corresponding file in the styles folder. The twin files in the styles folder are css files that contain the styling for the files found in the components folder. For the sake of avoiding redundancy the styling files will not be discussed here but there is one file for every component.

BulletUI.js is responsible for the UI in the bullet mate game mode. BulletUI.js and ClassicalUI.js share a lot in structure but the key difference is that the bullet game mode adds a timer to the base game. It also contains the relevant changes to send leaderboard data to the bullet mate leaderboard as opposed to the classical mate leaderboard.

ClassicalUI.js likewise is responsible for the classical mate user interface. This is the standard game mode that displays a game board, allows the user to make moves, verifies those moves as well as performing all other gameplay functions associated with this mode. There is a lot of shared functionality between classical and bullet mate but as they are not identical a distinction is made here.

CreateUI.js is responsible for the creating new puzzles page. This page is where users create custom puzzles to submit to the custom puzzle list.

CustomPuzzlesUI.js is the main menu for the custom puzzles section of the website. The main responsibility of this file to give users the option between creating or playing a custom puzzle when they enter the custom puzzles section of the website.

FailedPlayPuzzle.js is responsible for when a custom puzzle has been failed. It displays a popup that will notify the user that they failed a puzzle and give them the options to try again, return to the puzzle list, or return home entirely.

GameModeUI.js is responsible for the page after a user chooses the play option. It displays both the classic and bullet mate modes and lets the user choose between those two options.

GameOver.js is called when a user loses all of their lives in a game mode. It displays “Game Over! You have lost all lives.” and gives them the option to restart the game or return to the home page.

GameOverLeaderboard.js is called with the same criteria as GameOver.js but only if the user finishes with an eligible high score for the top ten leaderboards. Instead of displaying a standard game over message it will inform the user that they have achieved a top then score and will then prompt them to either restart the game or return home.

HelpTutorialMenuUI.js displays a variety of different information. This page begins with an about us section that gives a general overview of the project and the developers, and then follows up with tutorial information for new players. The page explains the movement of each piece, it explains how check and checkmates work, and it explains more esoteric rules such as castling and en passant. This is intended to be a simple resource for new players to be able to turn to and has relatively little functionality of its own, mostly just having standard navigation to and from the page itself.

LeaderboardUI.js displays the leaderboard information from the firebase database for both the classic mode and the bullet mate mode. This page pulls data from our leaderboards and displayed them side by side.

LoginUI.js is responsible for the login process. The page contains a prompt to log in using google with a button that begins the process. Logins are handled via firebase's authentication and was not hand developed here. Every part of this process is handled through firebase and account information is stored in the firebase real time database as well. We do not store any kind of password or sensitive information.

MainMenuUI.js is the display for the first page a new user will see. The main menu contains a title for our site as well as a sequence of buttons for a user to navigate through. A play button will take a user to the primary game modes, while other options allow users to navigate to the custom puzzles mode, adjust settings, access the help menu, and do anything else available on the site.

Navbar.js is responsible for creating the navigation bar at the top of most of the webpages. This is how the user will often be moving from page to page in order to avoid forcing them to back up all the way to the main menu to change from one webpage to another.

PlayPuzzleListUI.js is what a user encounters when attempting to play a custom puzzle submitted by another user.This file is responsible for displaying the list of custom puzzles and giving users the ability to scroll around and select what puzzle they want to attempt to complete next.

PlayUI.js is responsible for letting users play the custom puzzles submitted by other users. After a player selects a custom puzzle, this file loads the board state and runs the game. What differentiates it from classical mate's associated file is that this game mode has no leaderboard or score.

PuzzleSubmission.js is responsible for sending a puzzle to the firebase real time database and is called when a user attempts to do so.

SettingsUI.js is responsible for all of the settings in the site. This page has two main features. It displays a non-exhaustive list of recent scores a user has achieved assuming they are logged in, and will give a user the ability to toggle movement help dots on and off. This setting will be saved to the user's account.

SuccessPlayPuzzle.js is called when a user successfully completes a custom puzzle and will display a message congratulating them. They will be prompted to either return to the puzzle list or return back to the main menu.

UI Design Goals: The goal of the UI layout with all of these classes is to give the user most of the option they need early on so that navigation is as quick as possible. Six options are available from the main menu and even after going a few layers down the page hierarchy, the navigation bar will always give users the links they need to move around seamlessly. Most of the game logic is kept separate from each other on different game modes. This is to ensure that exploits, glitches, and breakdowns in the system stay localized. If any given game mode runs into an issue, the rest should remain functional.

Game Design Goals: As for the design of the game modes themselves, each mode is designed to fill a specific niche. The primary game mode "Classical Mate" was the starting point for the project and is the most straightforward game mode on display. This is a well worn type of chess play that many people are known to enjoy.

The "Bullet Mate" mode is meant to apply a time pressure. For many people, particularly advanced players, standard chess becomes stale as it is possible to take a large amount of time and plan out every possible move. The addition of the bullet chess timer adds the same time pressure we see in chess variants like bullet chess, from which the name is derived. The timer begins at a set value of 1 minute and adds 10 seconds to the clock for every successful puzzle solved. This allows skilled players to build extra time by blazing past early puzzles very quickly so the more difficult puzzles are less rushed.

And lastly, the custom puzzle mode allows players to try their hands at creating their own puzzles so they can express some creativity. Chess is a very honed game where new players are not given chances to be expressive as "expressive" moves in low level play are often just bad moves. This allows new players to the game to feel like they are being creative in a space where there are not necessarily wrong answers and it lets older players theorycraft possible endgames and cater to their own skill level as well.

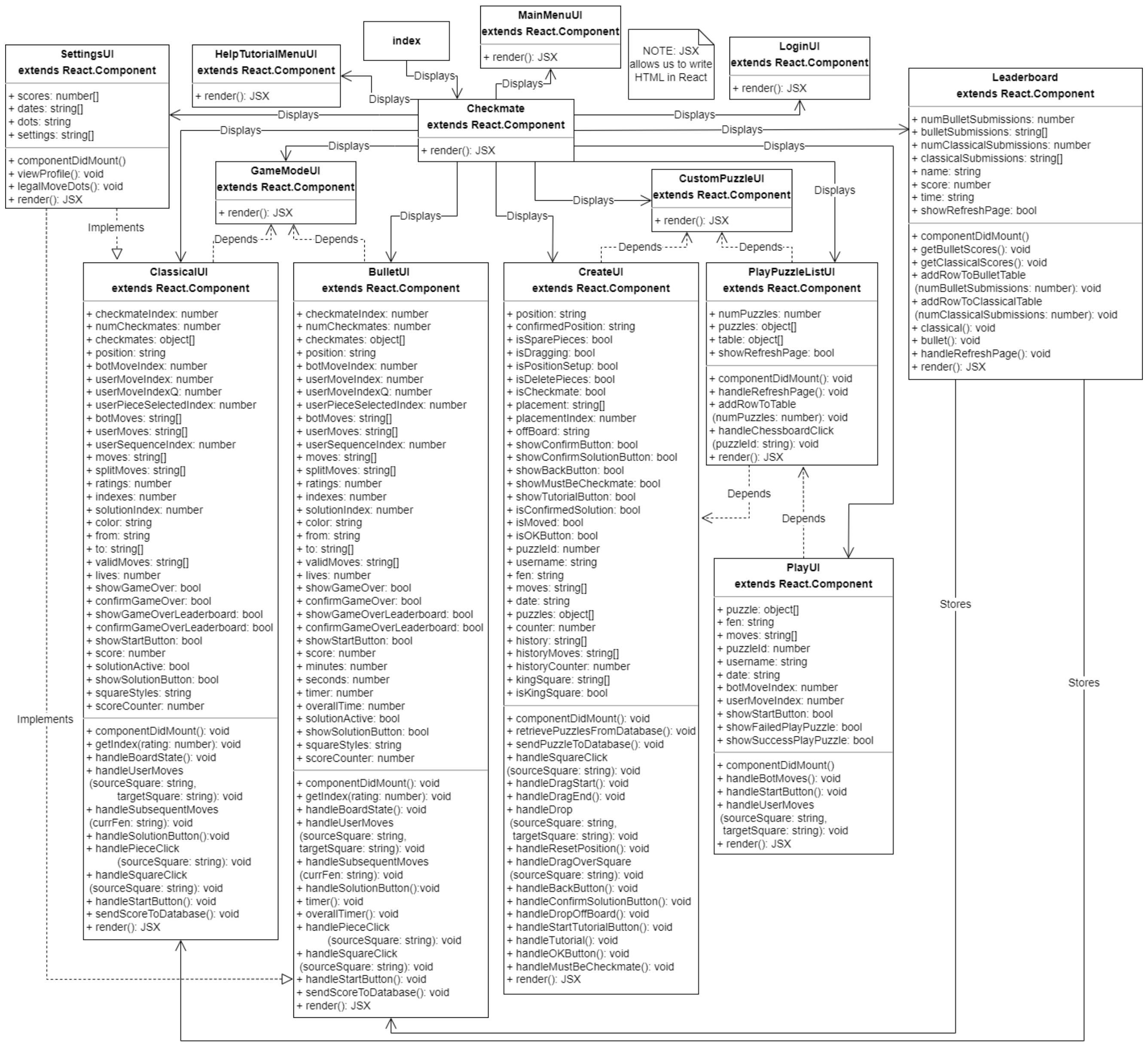


Figure 5. Checkmate's Component Diagram.

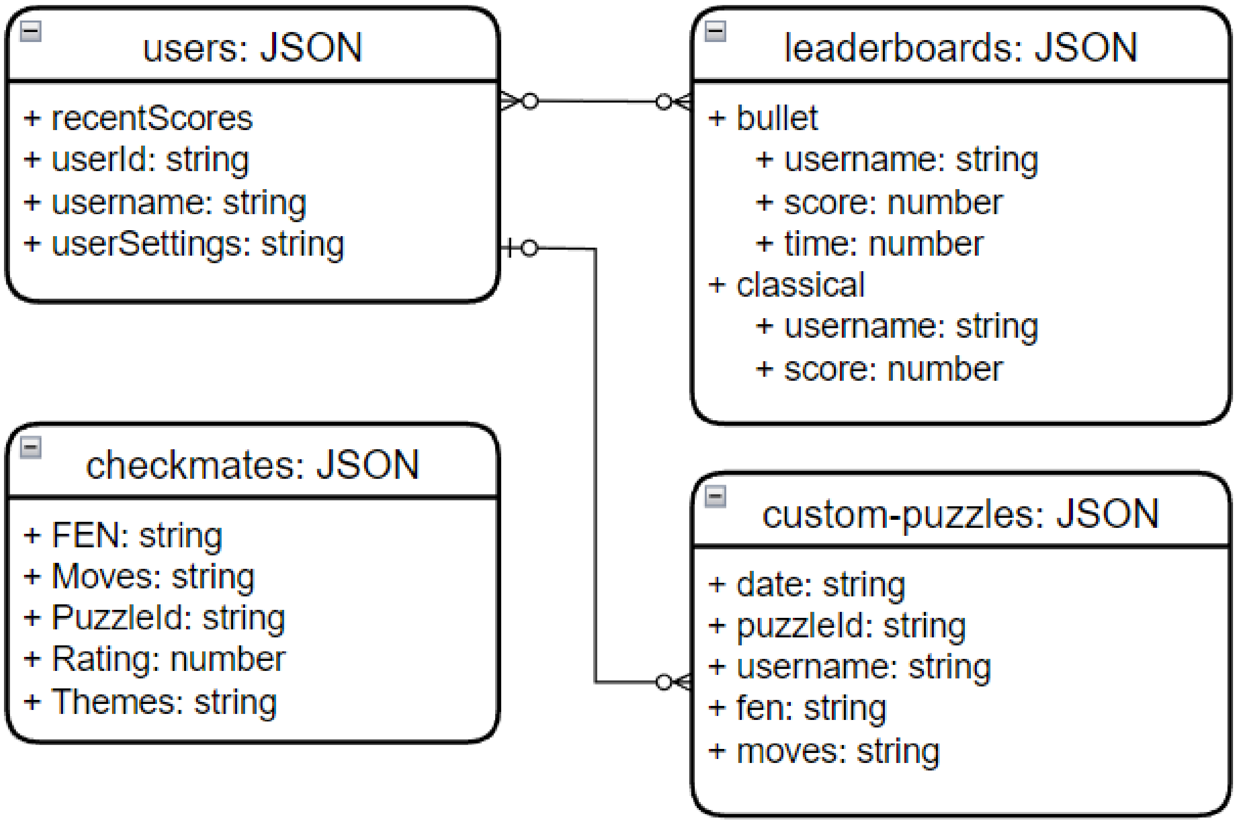


Figure 6. Checkmate's ERD Diagram.

Deployment Instructions

Introduction to Firebase

Firebase is a cloud-based platform developed by Google for building mobile and web applications. It offers a range of services and tools to help developers build high-quality, scalable apps quickly and easily. It provides real-time database services, authentication, hosting, storage, machine learning, and analytics tools that allow developers to focus on building great apps instead of worrying about server infrastructure and maintenance.

Prerequisites

The entirety of this project leveraged the services that Firebase provided. Some prerequisites that are needed is a Firebase account, installing Node.js, and having a code editor. We recommend using Visual Studio Code because this is the editor that was utilized throughout the entirety of development and will be used for further developments with this application.

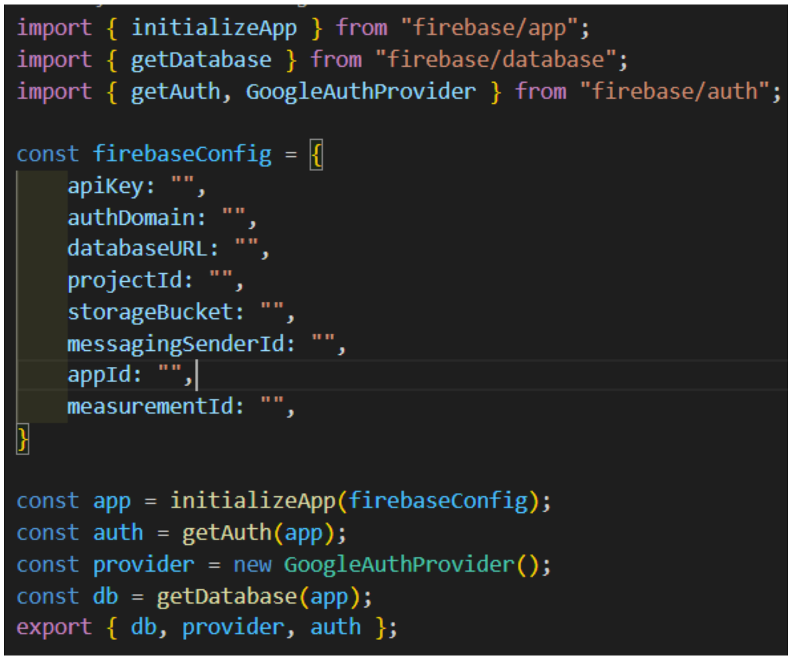
Cloning Repo

The repo for this application lives on GitHub. In order to get to the proper repo, go to https://github.com/esaijimenez/Checkmate. From here, there are various way to clone this repo. Inside of the command line in Visual Studio Code, the repo can be cloned at this location https://github.com/esaijimenez/Checkmate.git. Additionally, it can be downloaded via zip file or opened with GitHub Desktop.

Configurations

Once the repo is loaded into Visual Studio Code, the user must ensure that they have Node.js installed on their machine, then in the command line at the root of the project they run "npm install", "npm install firebase", and lastly "npm install -g firebase-tools". Running npm install loads in all of the items in the package-lock.json and package.json and stores that information inside of a node_modules folder.

Next, the user needs to create a firebase.js file that includes the configuration for their firebase project and add the following information. They can find this information inside the project settings inside of the Firebase Console of their project.



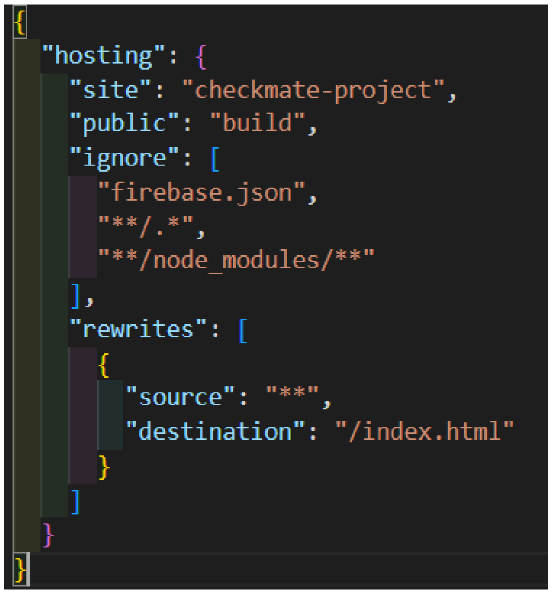
```
import { initializeApp } from "firebase/app";
import { getDatabase } from "firebase/database";
import { getAuth, GoogleAuthProvider } from "firebase/auth";

const firebaseConfig = {
    apiKey: "",
    authDomain: "",
    databaseURL: "",
    projectId: "",
    storageBucket: "",
    messagingSenderId: "",
    appId: "",
    measurementId: "",
}

const app = initializeApp(firebaseConfig);
const auth = getAuth(app);
const provider = new GoogleAuthProvider();
const db = getDatabase(app);
export { db, provider, auth };
```

Figure 7. firebase.js configuration.

By adding this information, this allows the user to use Firebase's different services that they offer. For this project, we only used the Realtime Database, Authentication, and Hosting. However, Hosting is not included inside the firebase.js. The Hosting services is taken care of inside of the firebase.json file that is created when the user runs the "firebase init" command. Inside of this file, the user need to put their own "site".



```
{
  "hosting": {
    "site": "checkmate-project",
    "public": "build",
    "ignore": [
      "firebase.json",
      "**/.*",
      "**/node_modules/**"
    ],
    "rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ]
  }
}
```

Figure 8. firebase.json configuration.

Compiling the Code

Once the configurations are set, the user must run "npm run build". This creates a build folder that allows Firebase to deploy the site. In order, in the command line, the user must run "npm install", "npm install -g firebase-tools", "npm run build", "firebase login", "firebase init", then "firebase deploy". Inside of the "firebase init" step, the user must ensure that they use the build folder because Firebase automatically wants to use the the public folder. If the public folder is used, the application will not deploy properly, this is why creating the build folder is extremely important.

Enabling Realtime Database

Like previously mentioned, the way we are storing our data is inside of the Firebase Realtime Database. In order to set one up, inside of the Firebase console, the user simply must navigate to the Realtime Database and follow the steps they provide. For all services, Firebase offers easy-to-follow instructions. Once created, the user can upload a JSON file if they already have information they want to store, or they can add however many tables they want. The Firebase Realtime Database is a NoSQL database so it is essentially a huge JSON file that we are able to pull information from.

Simple Startup

There is an option to quickly startup the application without deploying the actual website. Once the user clones the repo, the user must run “npm install”. This command creates a node_modules folder that stores all the items inside of the package.json and package-lock.json. Then the user must run “npm start”. After following these steps, this starts up the application and runs it on the user’s default browser inside of the localhost.

Known Bugs

Checkmate had around three months of actual development time, and time beforehand for planning and research. During this time, bugs have arisen and been addressed. While no bugs significantly inhibit a user’s ability to utilize Checkmate’s features, they still need to be fixed in order to ensure the website will remain functional and future work will not be affected.

One of the most apparent bugs is a visual glitch of the chessboard when playing either Classical or Bullet Mate mode. The board itself will repeatedly pop in and out when the page initially loads, and the only way to fix it is if the entire page itself is reloaded. This glitch initially persisted in early development about one in four times when page was rendered, however, that number has reduced significantly as development progressed. Despite the small chance that the board rendering will glitch, it is still possible. This will most likely be fixed in future iterations by styling and rendering the board according to a user’s screen size.

Another known bug is during gameplay, when the queen is supposed to deliver a checkmate but another piece checks the king on the exact same square that the queen was supposed to move to, the game essentially stops working and does not know what to do about this position. This incident was only observed once by sheer accident and we are unsure how to replicate this particular phenomenon. If this issue becomes more apparent in more positions, then this bug can be easily resolved, but for now, it is a mystery as to why it did this during this one instance.

In addition, there is a small visual bug during gameplay where the pieces of the chessboard will ‘fly’ off when a puzzle is completed. This may be due to a number of reasons, but it is most likely from the react-chessboard library integration with the Checkmate website. A deep dive on both the source code and react-chessboard documentation will be conducted in order to fix this bug.

Future Work

With the release of version 1.0 complete, ideal development of version 2.0 will include a number of features that were set aside while version 1.0 was in development. Version 2.0 will be released within the next four months, and all features mentioned may or may not make it to the final 2.0 version. This is not an official outline of requirements, as later documentation will need to officiate all version 2.0 requirements. All bugs above will, ideally, be addressed and fixed as well.

One design problem discovered during playtesting was that experienced players would spend much of the early periods of a run bored. Since puzzles increase in difficulty starting with very easy, proficient players get little out of the early puzzles and must grind through them in order to reach the puzzles that actually give them a satisfying challenge. One potential solution to this is a practice mode that lets a user set a specific starting elo for the puzzles they attempt. This would make it so that players who are already very proficient would not need to wade through the easy puzzles to get to the actual satisfying portion of play for them. During this game mode leaderboards would be disabled. Any player starting at a higher difficulty would have an inherent disadvantage anyways but in order to avoid potential exploits in the system and to more clearly define this option as a practice mode, disabling leaderboard submissions would likely be best for these modes.

A simple yet integral quality of life feature included on many modern day websites is a dark and light mode toggle for the entirety of the website. Such a toggle would give users maximum comfortability when using Checkmate, which is important because we would like all users to have the option to play under their preferred settings to reach their maximum performance when completing chess puzzles. Similarly, themes or preferred backgrounds from a predetermined set of colors and choices would allow users to have more freedom to play chess as they prefer. A common chess website feature is the ability to change piece and board style, as well as some board options (such as the ability to flip the board and see it from the opponent's perspective). Checkmate could take stylistic changes a step further, with overall website background changes and themes across the entire website, not just in-game. Base-level options, such as board flip and labeled squares, should also be included before more stylistic features, such as the aforementioned, are added.

A more complex feature involving both frontend and backend development would include the ability to compete in real-time puzzle races against other players. This would involve more than just stylistic changes, as the frontend and backend would both need to work together in tandem with other technologies in order to provide a space for players to match up online and compete with each other in real-time.

In addition, a user that is logged in would benefit from having a dedicated Profile page, as it would serve as a hub for all of their information and statistics. As of version 1.0, a majority of user information is simply stored in a database and not displayed. However, the user's most recent scores are currently housed on the settings page, so there is already the implementation of some user-specific information being shown. While not necessarily out of place, moving this section to the user's own page would improve the overall flow and usability of the website. Other features could be housed on the user's profile page as well, such as a profile picture, graphs showing the user's progress on Checkmate, the email associated with their account, their username, date their account was created, achievements such as previous leaderboard rankings, and more. This would make using Checkmate more modern and keep users coming back due to the fact that all of their data is in one place.

Some features could also be added to improve gameplay, including a record of all moves made so far on a puzzle, additional modes, and sound when pieces move. Keeping a record for players as moves are made is a common practice for chess websites with real-time play, but it could also be highly beneficial for a competitive chess puzzle website like Checkmate. Again, this could also be an option to the player. Whether toggled or not, it gives players who prefer a record of all moves the option to have it displayed. In addition, the Help and Tutorial page would need to be updated in order to describe chess notation for those who are unfamiliar, as these move records are often in chess notation.

Accessibility is essential to all websites, as everyone deserves to experience a website regardless of their needs. This will be focused heavily in future versions, as we want Checkmate to be enjoyed by as many people as possible, as soon as possible. The timeline for these features is unclear, as no members on the development team of Checkmate have experience with implementing accessibility features on this level. However, updates will follow, as we want to ensure inclusion and accessibility for all.

Time permitting, it would be critical to also fully support other browsers as well. As of version 1.0, Chrome is the only browser that is declared as fully supported. When loading Checkmate on other browsers such as Firefox and Edge, visual issues and color deviations arise. Ideally, Checkmate will be consistent in all ways across all browsers in order to provide the most professional and modern experience to users.