

Front-End Documentation

Table of contents

1. Technologies
 - 1.1. HTML
 - 1.2. CSS
 - 1.3. JavaScript
2. Third-party software
 - 2.1. Bootstrap
 - 2.2. jQuery
 - 2.3. chessboard.js
 - 2.4. chess.js
 - 2.5. clipboard.js
 - 2.6. fen-validator.js
3. GUI & functionalities
 - 3.1. Chessboard
 - 3.1.1. Load FEN
 - 3.1.2. Change color
 - 3.1.3. Current FEN
 - 3.2. Suggested moves
 - 3.2.1. Get suggested moves
 - 3.3. Commentary
 - 3.3.1. Load match

1. Technologies

Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web.

1.1 HTML

Hypertext Markup Language (HTML) is a fairly simple language made up of elements, which can be applied to pieces of text to give them different meaning in a document (for example, paragraphs, bulleted lists, tables). You can structure a document into logical sections (e.g.: header, columns of content, navigation menu). You can embed content into a page such as images, links and videos or one of many other available elements or even a new element that you define.

We chose HTML thanks to its plenty advantages. It is easy to learn, use and understand even for novice programmers. It is free and we don't need any software for HTML, no plug-ins are needed. It is supported on almost every browser.

1.2 CSS

Cascading Style Sheets (CSS) is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, and reduce complexity and repetition in the structural content.

We chose to use CSS in our project because it contributes on the separation of style and structure. It's accesible and easier for us to maintain and update. Also, CSS brings a greater consistency in design, having more formatting options and a lightweight code. In fact, CSS stylesheets increase our website's adaptability and we can ensure that more visitors will be able to view our website in the way we intended.

1.3 JavaScript

JavaScript (JS), is a high-level, interpreted programming language with first-class functions. It is a language that is also characterized as dynamic, weakly typed, prototype-based and multi-paradigm.

We chose to use JavaScript in our project because it is relatively simple to learn and implement. And also, for its capacity to empathize with other languages due to the fact that JS plays nicely with diverse languages and can be used in a huge variety of applications.

2. Third-party software

2.1 Bootstrap

The world's most popular front-end component library, Bootstrap is an open source toolkit for developing using HTML, CSS and JS. Bootstrap is a framework to help you design websites faster and easier. It includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels, etc. It also gives you support for JavaScript plugins.

2.2 jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

The jQuery library contains the following features:

- HTML/DOM manipulation
- CSS manipulation
- HTML event methods
- Effects and animations
- AJAX
- Utilities

2.3 chessboard.js

A javascript chessboard library which depends on jQuery. It is designed to be "just a board" and expose a powerful API so that it can be used in different ways.

Main features:

- Easy integration
- Show game positions alongside your expert commentary
- Integrate chessboard.js and chess.js with a PGN database and allow people to search and playback games

License: [MIT License](#).

2.4 chess.js

A Javascript chess library that is used for chess move generation/validation, piece placement/movement, and check/checkmate/stalemate detection - basically everything but the AI.

chess.js has been extensively tested in node.js and most modern browsers.

You can find the complete documentation and more examples [here](#)

License: [BSD 2-Clause](#)

2.5 clipboard.js

A modern approach to copy text to clipboard. It uses event delegation which replaces multiple event listeners with just a single listener.

In this project it is used to copy the current FEN (more details in the section *GUI & functionalities > Chessboard > Current FEN*)

License: [MIT License](#)

2.6 fen-validator.js

A library that uses *RegExp* to validate the FEN notation.

License: GNU General Public License V3

3. GUI & functionalities

3.1. Chessboard

The interactive chessboard is at the center of UX, allowing users to simulate a match of chess from start to finish. The AI part of the application is not designed to play against humans thus the user is expected to interact with both the black and white pieces and an integrated chess logic module allows only legal moves to be executed.

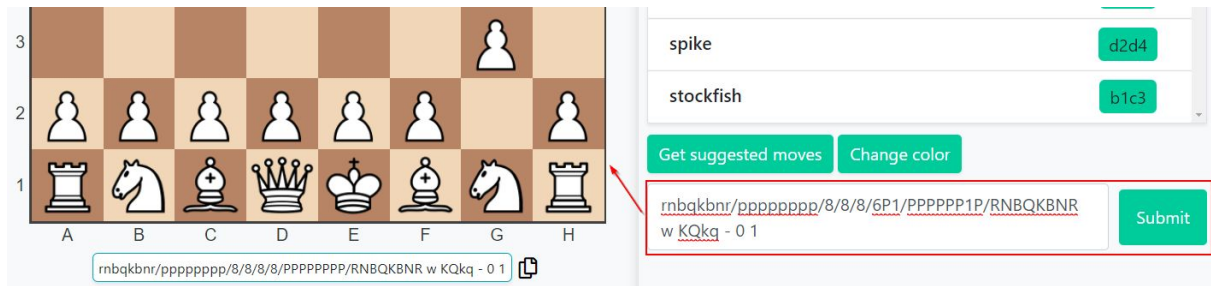
The chessboard is embedded in the application using two libraries: **chessboard.js** for the visual representation and **chess.js** for the chess logic. The component was designed using JavaScript's **module pattern** and is exposed as an object having two properties: *CHESSBOARD* and *CHESS*.

Its initialization is on the first line of a `$(document).ready()` **jQuery** function to ensure that the element is ready before the user can access any of the application's functionalities.

CHESSBOARD is a reference to the **chessboard.js** library and is responsible for all the actions that concern user interaction (e.g. moving the pieces, loading a FEN, flipping the table).

CHESS is a reference to the **chess.js** library and is used to keep track of the chess logic involved (allow only legal moves, create a history based on of what moves were played, extract the FEN notation of the current board configuration).

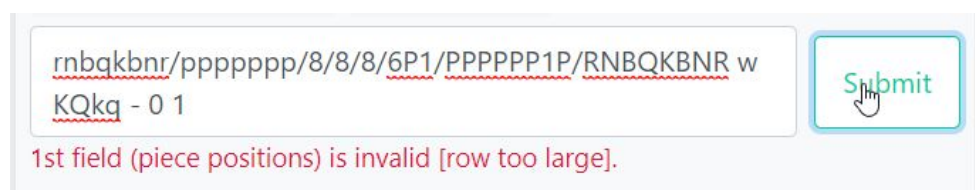
3.1.1. Load FEN



You can make a move by simply using load FEN code functionality. In the right side of the table, you can input your FEN board state, click submit and the desired state will be represented on the board. Basically when “Submit” button is clicked, the value inserted in the FEN textarea is parsed and validated using the library **fen-validator.js**.

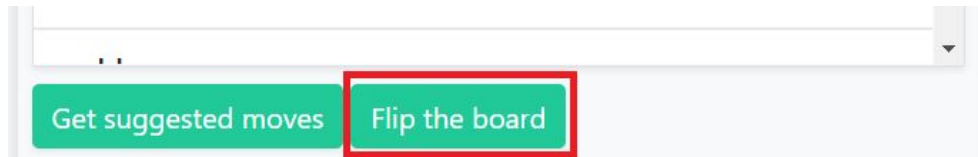
```
export function setChessboardFen() {  
  // grab the textarea from the .fen-loader component  
  const fenTextarea = document.getElementById('js-fen-textarea');  
  // element that displays an error if  
  // a non-FEN string is entered in the textarea  
  const fenFeedback = document.querySelector('.fen-loader ~ .fen-feedback');  
  // get the value of the textarea  
  const fen = fenTextarea.value;  
  
  // check if fen is valid  
  if( parseFEN(fen) ) {  
    CHESS_COMPONENT.history = "";  
    CHESS_COMPONENT.CHESSBOARD.position(fen);  
    CHESS_COMPONENT.CHESS.load(fen);  
  } else {  
    let validationResult = CHESS_COMPONENT.CHESS.validate_fen(fen);  
    $(fenFeedback).html(validationResult.error);  
    // make it visible  
    fenFeedback.style.opacity = 1;  
    // hide it after a delay of 1.5s  
    setTimeout( function () {  
      fenFeedback.style = '';  
    }, 3000);  
  }  
}
```

If the inserted FEN state is valid, its value will be interpreted using the library chesboard.js in order to render the state on the board and update the current FEN value. If the inserted FEN value is invalid, the error returned from validation will be displayed in *fen-feedback* area:



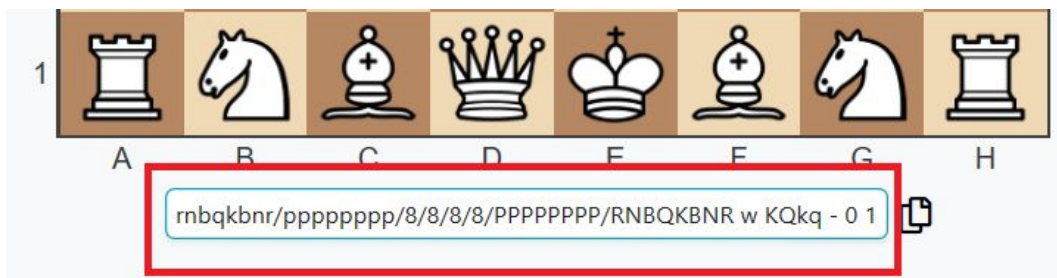
3.1.2. Flip board

The *Flip Board* button provides the functionality to rotate the board so that either set of pieces is facing you. This is done using a method exposed through the *CHESSBOARD* member of the *CHESS_COMPONENT*. The rotation is achieved with a call to the *.flip()* method of the **chessboard.js** library.



3.1.3. Current FEN

In the application, just under the chess table, you can find a container with a seemingly strange string of characters; that would be the current FEN, the encoded current state of the board.



Just next to it, to the right, is a copy button, which will save the current FEN into your clipboard. This way you can save your progress, close the application, and start again from where you left off at a later date.



The application renews the contents of the “Current FEN” container everytime a move is made (e.g. when the board state changes). When the application loads for the first time, or a move is made, the following function is called into action:

```
export function renderCurrentFEN( currentFEN ) {  
  const fenContainer = document.getElementsByClassName('current-fen')[0];  
  const width = document.getElementById('js-chessboard').firstChild.firstChild.style.width;  
  const currentFenEl = document.getElementById('js-current-fen-value');  
  
  fenContainer.width = width;  
  currentFenEl.innerText = currentFEN;  
}
```

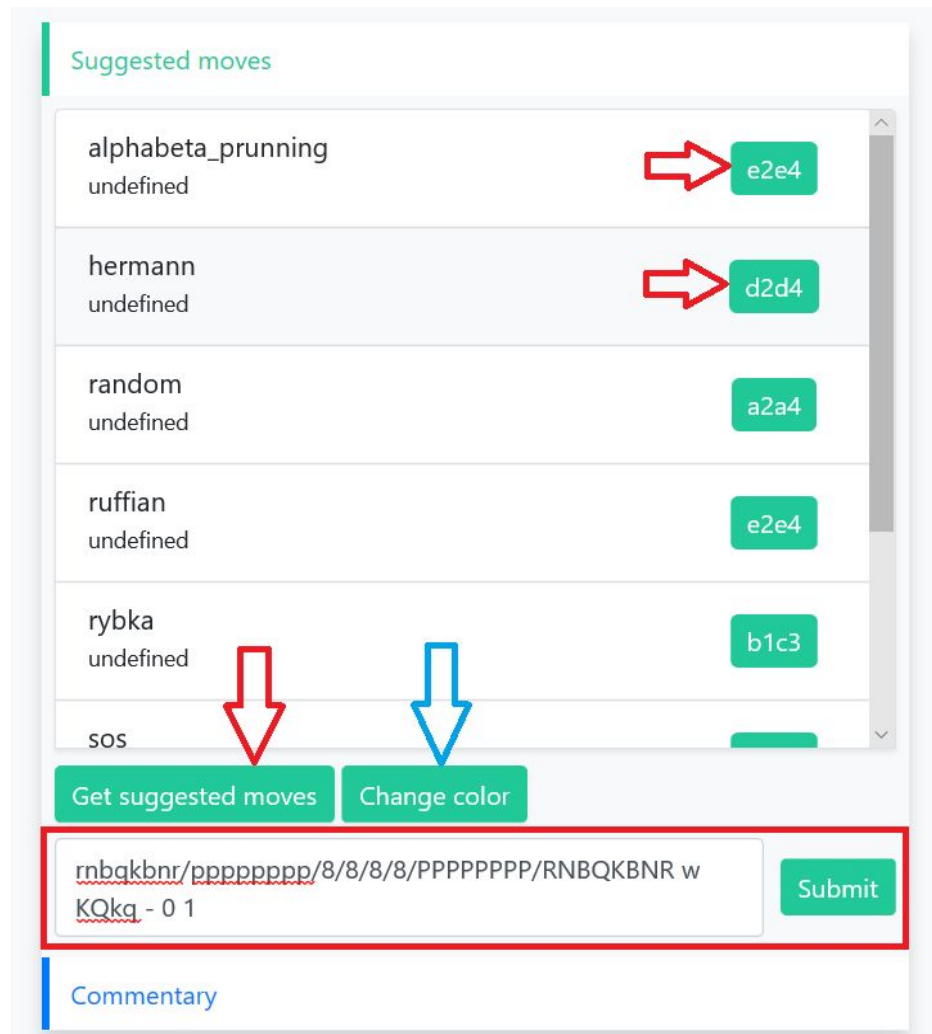
It receives the current FEN string as a parameter and updates the container in the application accordingly:

3.2. Suggested moves

In the right upper side of the application there is a “Suggested Moves” green button. Clicking on it will reveal a dropdown with a number of algorithms that the AI can apply in order to establish the next move, along with a number of functionalities.

Here you can do a number of things:

- You can ask the AI to help, by hitting the ‘Get suggested moves’ button. More on this in the ‘Get suggested moves’ section bellow.
- You can change the color of your chess pieces from white to black and vice versa, by hitting the ‘Change color’ button
- You can bring the board to a desired state by introducing a FEN string into the input container and clicking on the ‘Submit’ button



3.2.1. Get suggested moves

As is explained in the above section, the *Get suggested moves* button will prompt the AI to make a decision regarding the next move. There are a number of strategies for it to do this, and each of them may or may not produce the same result as the others (see the above picture). It is up to the player to choose what move to make next.

```
export let strategies = [];  
  
export function getStrategies() {  
  const STRATEGIES_URL = '/strategies';  
  $.getJSON( `${constants.BASE_URL}${STRATEGIES_URL}`, function( data ) {  
    $.each(data, function( key, val ) {  
      if ( "strategy" in val ) {  
        strategies.push(val.strategy);  
      }  
    });  
  
    // render strategies details  
    render.renderStrategiesDetails( data );  
    render.renderSuggestedMovesMarkup( data );  
  });  
}
```

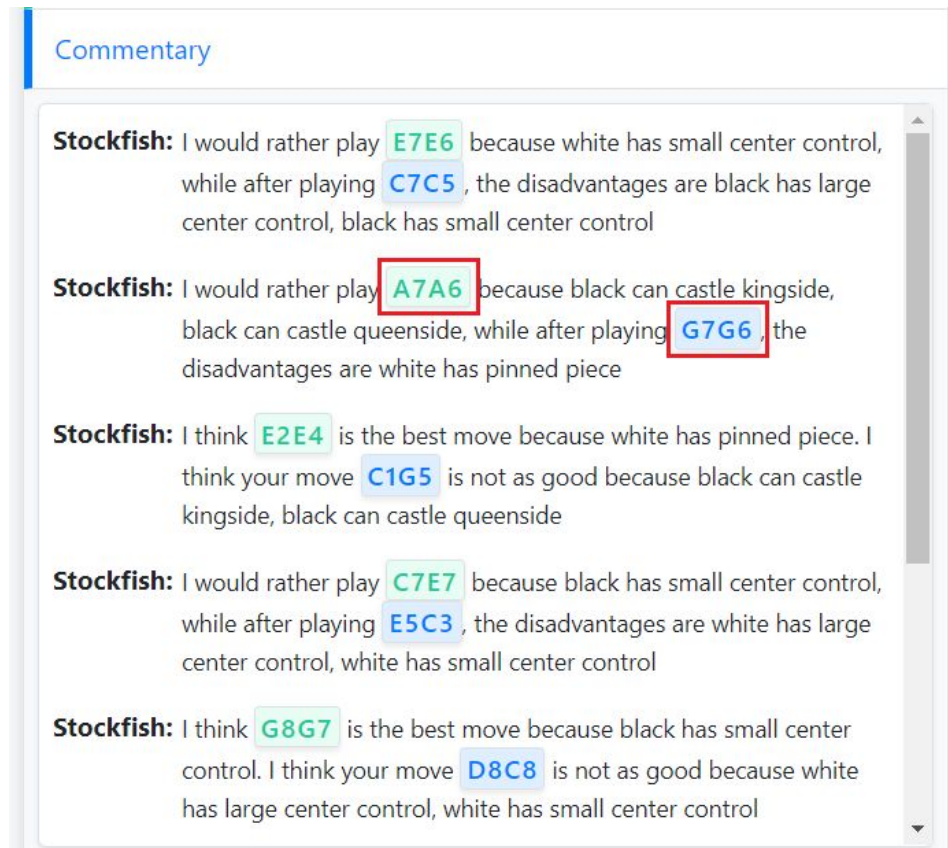
As far as the implementation is concerned, the flow is the following:

1. A call is made to the server to retrieve all the strategies.
2. The 'strategies' variable stores the strategy names and details. Inside the 'getStrategies' function the containers are also rendered, to avoid their redundant recreation everytime the suggested moves change.
3. Next, the 'getSuggestedMoves' function is called to retrieve and render the actual suggestions for each strategy.

```
export function getSuggestedMoves() {  
  let currentFEN = CHESS_COMPONENT.CHESS.fen();  
  let url = `${constants.BASE_URL}/moves?fen=${currentFEN}&strategy=`;  
  
  for (let i = 0; i < strategies.length; i++) {  
    let strategyName = strategies[i];  
    let moveEl = document.getElementById(`js-move-${strategyName}`);  
    let loader = document.getElementById(`js-loader-${strategyName}`);  
    // hide move element  
    moveEl.style.opacity = 0;  
  
    if( loader.classList.contains('in-view') ) {  
      loader.classList.remove('in-view');  
    }  
    // show loader  
    loader.classList.toggle('in-view');  
  
    $.getJSON( `${url}${strategyName}`, function( data ) {  
      let response = data[0];  
  
      // hide loader  
      loader.classList.toggle('in-view');  
      // show move element  
      render.renderSuggestedMove(strategyName, response.move);  
    });  
  }  
}
```


3.3. Commentary

The *Commentary* component is used to display a commentary about a finished match. The user can either play until checkmate or use the *Load Match* submit field to insert a string of concatenated *from, to* pairs of moves. The text is generated on the Back-End and is fetched for the Front-End via a request to the */commentary* route, using the string notation aforementioned as a query parameter.

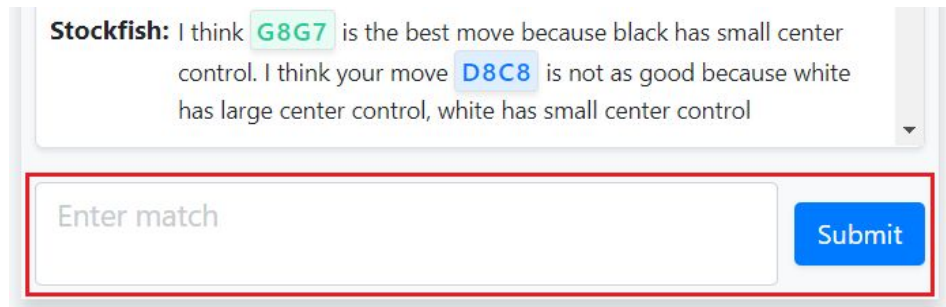


On the left there are the labels of the *strategy* that produced the comment. Inside the text of the comment there are some moves highlighted in either green or blue. The green ones are suggestions made by a certain *algorithm/engine* that the application is using. The blue ones are what the user played at a certain turn or was found in the string submitted using the *match loader*.

If either one of the moves present in the same comment is clicked the chessboard will change it's configuration to reflect the turn before the move highlighted in blue. This happens in order to show the user the difference of advantage gain between his move and the one suggested by the application.

3.3.1. Load match

The *Load match* component allows the user to submit a match using a string notation. The format for the said string is concatenated strings of *from*, *to* pairs of moves, describing a match from start to checkmate.



The screenshot shows a user interface for submitting a match. At the top, a text box contains the following text: "Stockfish: I think G8G7 is the best move because black has small center control. I think your move D8C8 is not as good because white has large center control, white has small center control". The moves "G8G7" and "D8C8" are highlighted in green and blue respectively. Below this text box is a red-bordered container. Inside this container, on the left, is a text input field with the placeholder text "Enter match". On the right side of the container is a blue button with the text "Submit".