****

***UNIVERSITY OF CRETE COMPUTER SCIENCE DEPARTMENT***

**DEGREE THESIS**

**A Scouting system for Chess Players**

Candidate : Samopoulos Georgios

Α.Μ. : CSD 4216

Supervisor : Yannis Tzitzikas

Co-Supervisor : Dimitris Plexousakis

Contents

1. Introduction ………………………………………………… 3
   1. General Description ……………………………… 3
   2. Motivation & Novelty …………………………… 3
2. General Thesis Information ………………………………… 4
   1. Timelapse ………………………………… 4
   2. Front-End Statistics ………………………………… 4
   3. BackEnd Statistics ………………………………… 5
3. System Design ………………………………… 6
   1. Tools ………………………………… 6
   2. Front-End Design …………………………………
   3. Back-End Design …………………………………
   4. Database Design …………………………………

**Introduction**

**1.1 General Description**

In this thesis, a novel web-based system tool is implemented, to assist chess players in organizing and analyzing data about their opponents. Its primary objective is to enhance the efficiency and speed of pre-game analysis by providing comprehensive reports of player’s games. Users can choose the player that they want to scout and the tool provides data such as player's performance based on opening or color, performance per move and graphs. Moreover, users gain the ability to effortlessly store and manage their games and profiles, eliminating the need for manual record-keeping. By these reports, players can gain valuable insights into their opponents' preferences for opening variations and ideas, and how accurate they are. Furthermore, the tool facilitates communication and collaboration among users. They can share comments, exchange ideas, and discuss game analyses with other users directly within the platform via chats. This fosters a collaborative environment where players can learn from each other's experiences and refine their own skills. Overall, the implementation of this web-based tool aims to change the way chess players approach pre-game analysis.

**1.2 Motivation & Novelty**

There is no online tool, which provides organized and analyzed data of his opponent to a chess player. The lack of this web-application is a great chance to develop a new automatization for chess players. This tool helps users to make their pre-game analysis faster and more efficient, by giving them rigorous game reports and a concluded profile to work via UI. Moreover, this system doesn't use its own database of games but downloads them when the user uses them. This approach ensures that the system remains up to date without requiring frequent updates to its games’ database.

**General Thesis Information**

The information below regarding the timelapse of the project , frontend and backend statistics.

**2.1 Timelapse**

Date received: 21 Oct 2022

Date submitted: 5 June 2023

**2.2 Frontend Statistics**

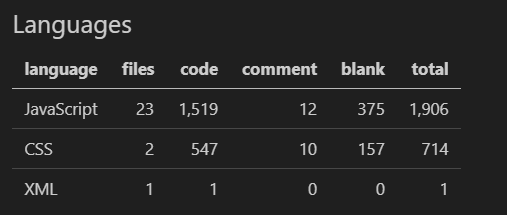
****

Figure 2.2.1: Number of files and code lines per language of frontend

****

Figure 2.2.2: Total number of files and code lines of frontend

**2.3 Backend Statistics**

**Controllers**

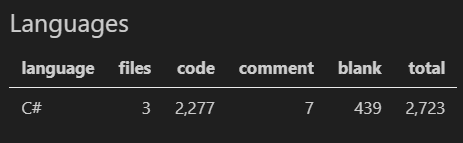
****

Figure 2.3.1: Total number of files and code lines of controllers’ folder

**Migrations**

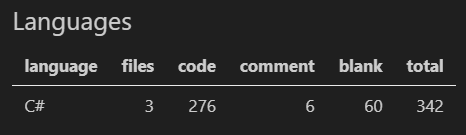
****

Figure 2.3.2: Total number of files and code lines of migrations’ folder

**Models**

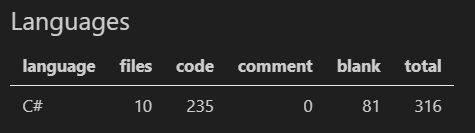
****

Figure 2.3.3: Total number of files and code lines of models’ folder

**System Design**

Chess Scout is a full-stack application and consists of 3 parts. The first part is the front-end , which is the client-side of a web application and contains everything that the user interacts with. The second part is the back-end, which is the server-side of a web application and contains all the components and processes that operate behind the scenes and handle the logic, data storage, and communication between the user interface and the database. The third and last part is the database , where our data is stored.

**3.1 Tools**

**Front-end**

**Programming Languages**

* HTML
* CSS
* JavaScript

**Libraries**

* Bootstrap
* React JS

**Back-end**

**Programming Languages**

* C#

**Programming Environment**

* ASP.NET

**Database**

**Programming Languages**

* SQL

**Software Tools**

* Visual Studio Code
* Visual Studio 2022
* SQL Server Management Studio 19
* Postman
* Google Chrome

**3.2 Frontend Design**

For the frontend development of this web app, HTML, CSS, and JavaScript are used, with React JS library. The decision to use React JS offers a number of positive aspects.

1. Component-Based Architecture: By adopting a component-based approach, React.js facilitates the organization of code into modular and reusable components. This approach enhances code maintainability and reusability, resulting in improved development efficiency.
2. Efficient Rendering: React.js employs an efficient rendering process by utilizing its reconciliation algorithm. This algorithm intelligently determines the most effective approach to update the user interface by comparing the previous and current states. As a result, React.js minimizes unnecessary DOM manipulations, leading to enhanced performance and improved efficiency.
3. Reusability: The component-based architecture of React.js promotes code reusability, allowing components to be easily reused throughout different sections of an application. This reusability significantly reduces development time and effort, as developers can leverage existing components to build new features or sections of the application without having to recreate them from scratch.
4. Virtual DOM: React.js employs a virtual DOM, a lightweight representation of the actual DOM. This virtual DOM allows React.js to efficiently update and render only the necessary components when there are changes in data. By minimizing the number of DOM manipulations, React.js achieves improved performance and responsiveness in rendering user interfaces.
5. Rich Ecosystem and Community Support: The React.js community is vast and dynamic, offering abundant documentation, tutorials, and a plethora of third-party libraries. This thriving ecosystem provides developers with a wide array of tools and resources that boost development productivity and address common challenges. With such extensive support, developers can leverage the collective knowledge and expertise of the community to overcome obstacles and streamline their React.js projects.

These are some of the reasons that React JS is a very powerful and preferable library in order to develop the front-end of a web-app.

Moreover, Bootstrap library is utilized for frontend development of Chess Scout. Bootstrap library helps to create a more appealing web application by stylingand splitting UI components.

**3.3 Backend Design**

The server developed over ASP.NET framework, that uses C# programming language. There are many reasons that we choose ASP.NET . First of all, this framework is a very powerful framework for building APIs, because it provides several features and capabilities that make it well-suited for developing scalable APIs. Moreover, ASP.NET offers cross-platform capabilities for the applications in order to run on multiple platforms such as Windows , macOS and Linux so we can create a bigger target group of users. Furthermore, Microsoft provides a huge set of built-in tools, libraries, and components that a developer can use to make the process of development more efficient. In our project, one tool that helped was the caching, which ASP.NET offers in the newest version.

**3.4 Database Design**

The database that we created is a simple SQL database. SQL databases are recommended due to their flexibility, scalability, and ability to handle complex relationships between data. SQL allows you to define the structure of your database, insert, update, and delete data, and perform complex queries to retrieve the desired information. We generated the following 6 tables in order to organize and retrieve our data.

The **Users** table which contains the following data :

* uid (user’s unique number) [primary int NOT NULL PRIMARY KEY]
* username (user’s username) [varchar(255) NOT NULL]
* pass (user’s password) [varchar(255) NOT NULL]
* email (user’s email) [varchar(255) NOT NULL]
* country (user’s country) [varchar(255) NOT NULL]
* birthday (user’s birthday) [date NOT NULL]

| **uid** | username | password | email | country | birth date |
| --- | --- | --- | --- | --- | --- |
| **#0** | giorgos123 | csd123! | csd4216@csd.uco.gr | Greece | 22/12/2000 |

The **ScoutProfiles** table which contains the following data :

* Pid (profile’s unique number) [int NOT NULL PRIMARY KEY]
* Uid (user’s unique number that owns the profile) [int NOT NULL]
* PlayerName (player’s name that we scout) [varchar(1000) NOT NULL]
* Elo (player’s elo ranking) [int NOT NULL]
* JsonGames (the json of the player’s games that we use) [text NOT NULL]
* JsonOpeningPreferted (the json of the player’s preferred openings) [text NOT NULL]
* JsonOpeningTree (the json of the player’s opening tree) [text NOT NULL]

| **pid** | uid | player’s name | Elo | JsonGames | JsonOpeningPreferted | JsonOpeningTree |
| --- | --- | --- | --- | --- | --- | --- |
| **#1** | #0 | Carlsen Magnus | 2882 | ….. | ….. | ….. |

The **ChessGame** table which contains the following data :

* Gid (profile’s unique number) [int NOT NULL PRIMARY KEY]
* Uid (user’s unique number that owns the game) [int NOT NULL]
* WhitePlayer (white player’s name) [varchar(1000) NOT NULL]
* BlackPlayer (white player’s name) [varchar(1000) NOT NULL]
* Result (game’s result) [varchar(20) NOT NULL]
* JsonGame (the game’s json ) [text NOT NULL]

| **Gid** | Uid | WhitePlayer | BlackPlayer | Result | JsonGame |
| --- | --- | --- | --- | --- | --- |
| **#1** | #0 | Carlsen Magnus | Anish Giri | 1-0 | ….. |

The **FriendRelations** table which contains the following data :

* uid1 (user1’s unique number) [int NOT NULL]
* uid2 (user2’s unique number) [int NOT NULL]

## 

| uid1 | uid2 |
| --- | --- |
| #0 | #1 |

The **FriendRequests** table which contains the following data :

* Sender (sender’s unique number) [int NOT NULL]
* Receiver (receiver’s unique number) [int NOT NULL]

## 

| Sender | Receiver |
| --- | --- |
| #0 | #1 |

The **Message** table which contains the following data :

* uidSender (sender’s unique number) [int NOT NULL]
* uidReceiver (receiver’s unique number) [int NOT NULL]
* MessageText (message sent) [text NOT NULL]
* seen (if the message has been seen or not)[int NOT NULL]
* Date (time that sent)[datetime NOT NULL]

## 

| **uid (sender)** | **uid (receiver)** | message | seen | Date |
| --- | --- | --- | --- | --- |
| #0 | #1 | text… | true | 12-12-2000 |

**Installation**

**4.1 Front-End Setup**

1. Install VS Code : https://code.visualstudio.com/download
2. Create npm project : write the command “npx create-react-app my-app ”
3. Replace the files public ,src and node\_modules with these inside the link : <https://drive.google.com/drive/folders/1xUTvj_OCsP1khZos7IOGjBpClWsvJp9f?usp=drive_link>
4. Replace the package.json and package-lock.json files

**4.2 Back-End Setup**

1. Install Visual Studio 2022 : <https://visualstudio.microsoft.com/downloads/>
2. Download and extract zip file ChessScout : <https://drive.google.com/drive/folders/1xUTvj_OCsP1khZos7IOGjBpClWsvJp9f?usp=sharing>
3. Open the project above with Visual Studio 2022

**4.3 Database Setup**

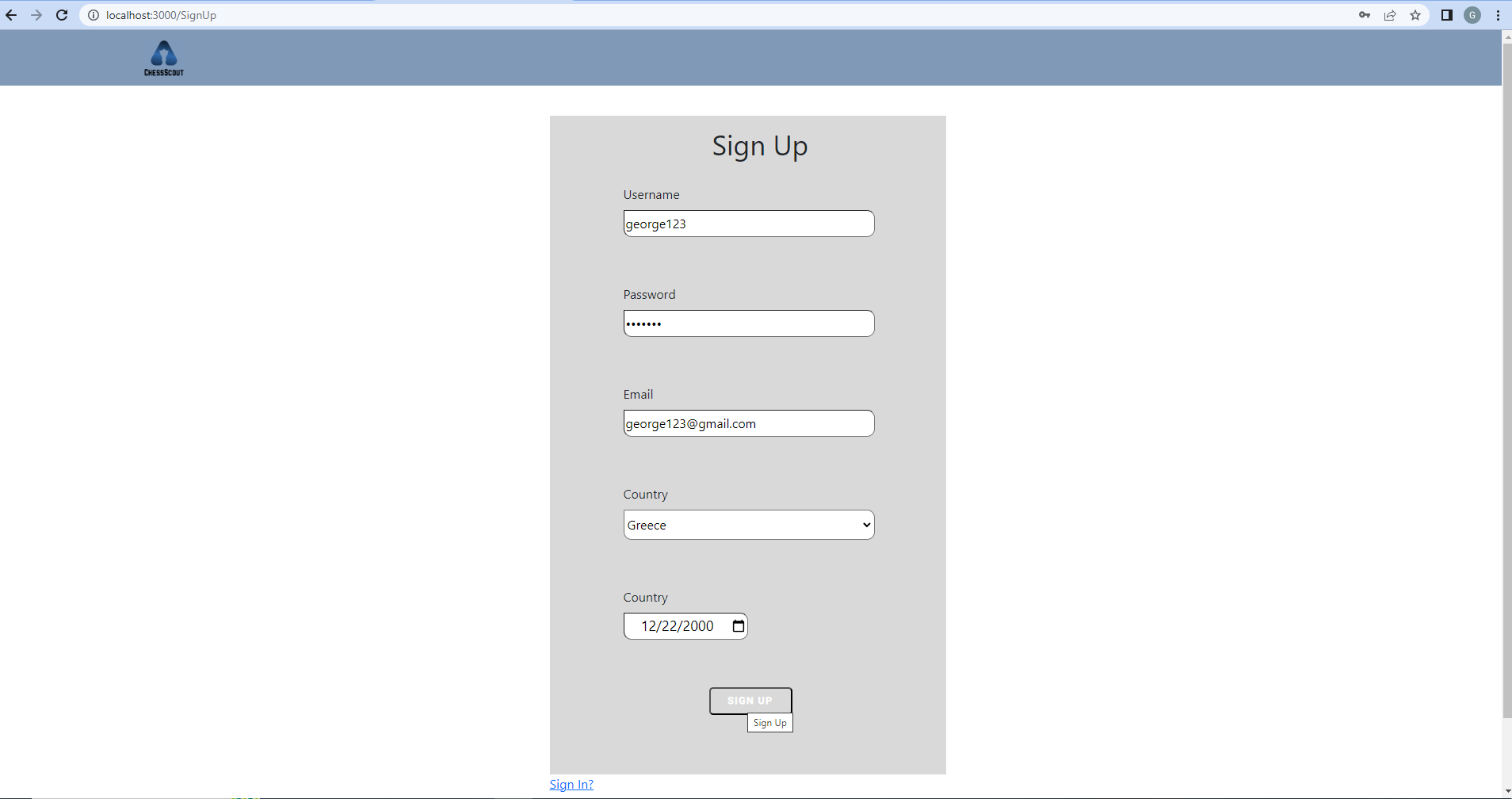
1. Install SSMS (SQL Server Management Studio) : <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16#download-ssms>
2. Connect your SQL Server with Windows Authentication
3. Create a Database with name ChessScoutDB
4. Click New Query , paste and execute the code from the Database\_init file : https://drive.google.com/drive/folders/1xUTvj\_OCsP1khZos7IOGjBpClWsvJp9f?usp=sharing

**System Tests**

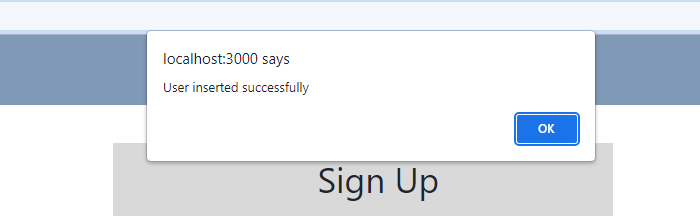
It's important for our system to undergo testing. To ensure this, we have created a series of tests that thoroughly check all the features provided by the system. Here are the tests :

* Sign up a new user.
* Sign in and create a new profile for a player of our choice.
* View profile’s graphs.
* Apply filters on the profile and use the opening tree.
* Open and save a game
* Save and open a profile
* Delete profile and game
* Search a user and send a message

**1) SIGN UP USER**

For the first test we will create a new user for our system. So we open the app and click on “Sign Up ?” link. Then we fill the form.

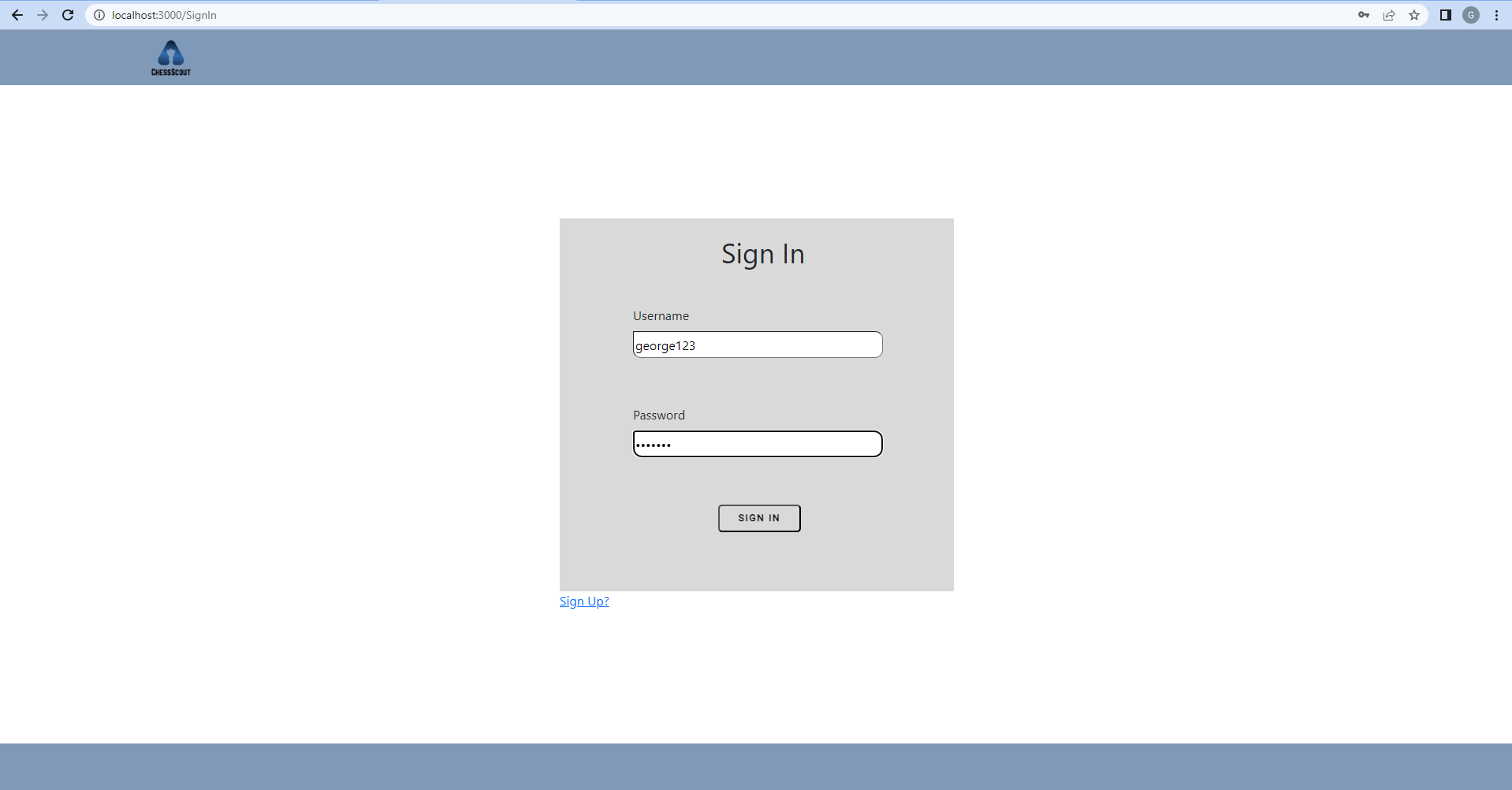
After that we click on the Sign Up button.



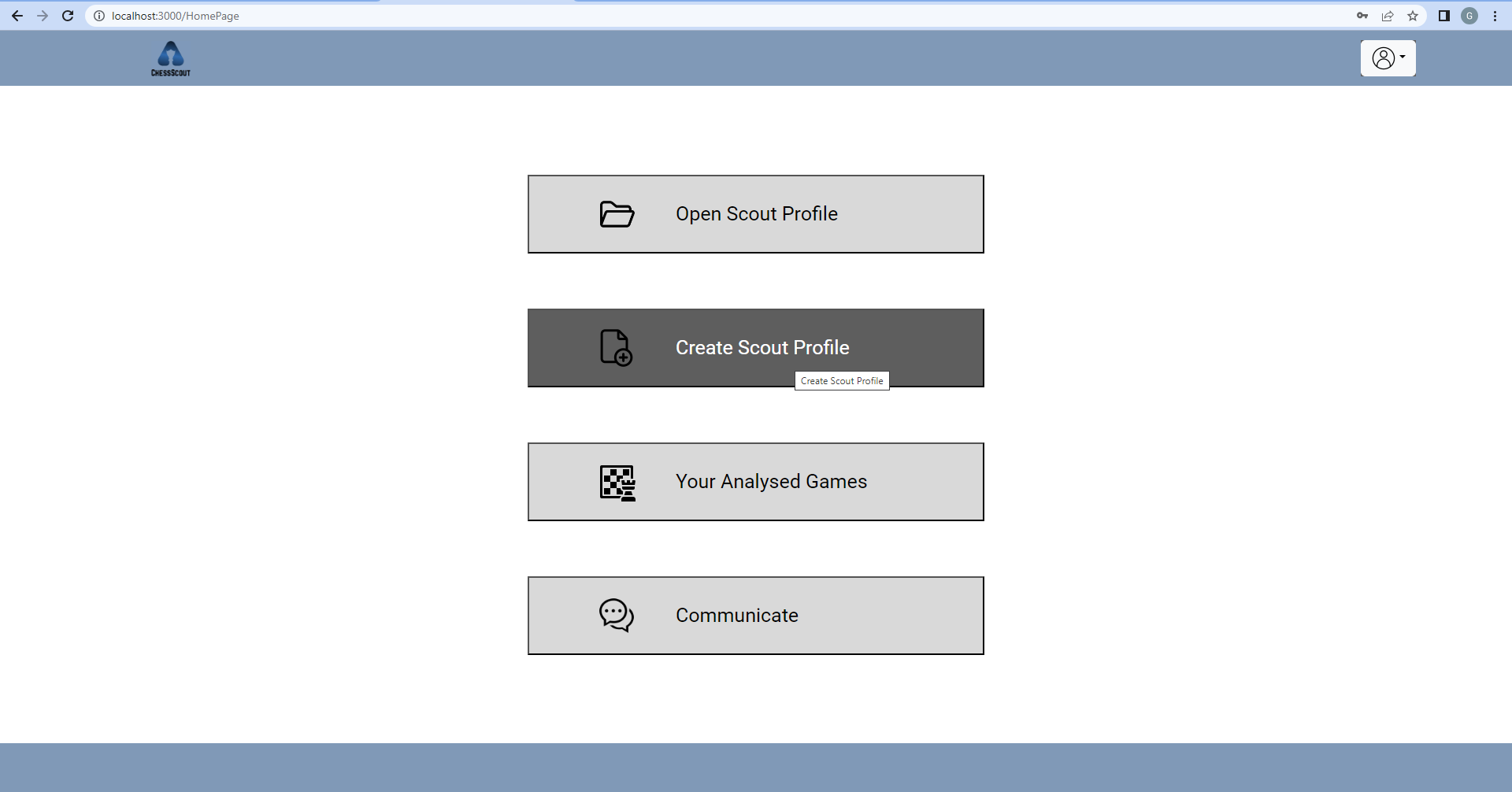
Our user is created successfully.

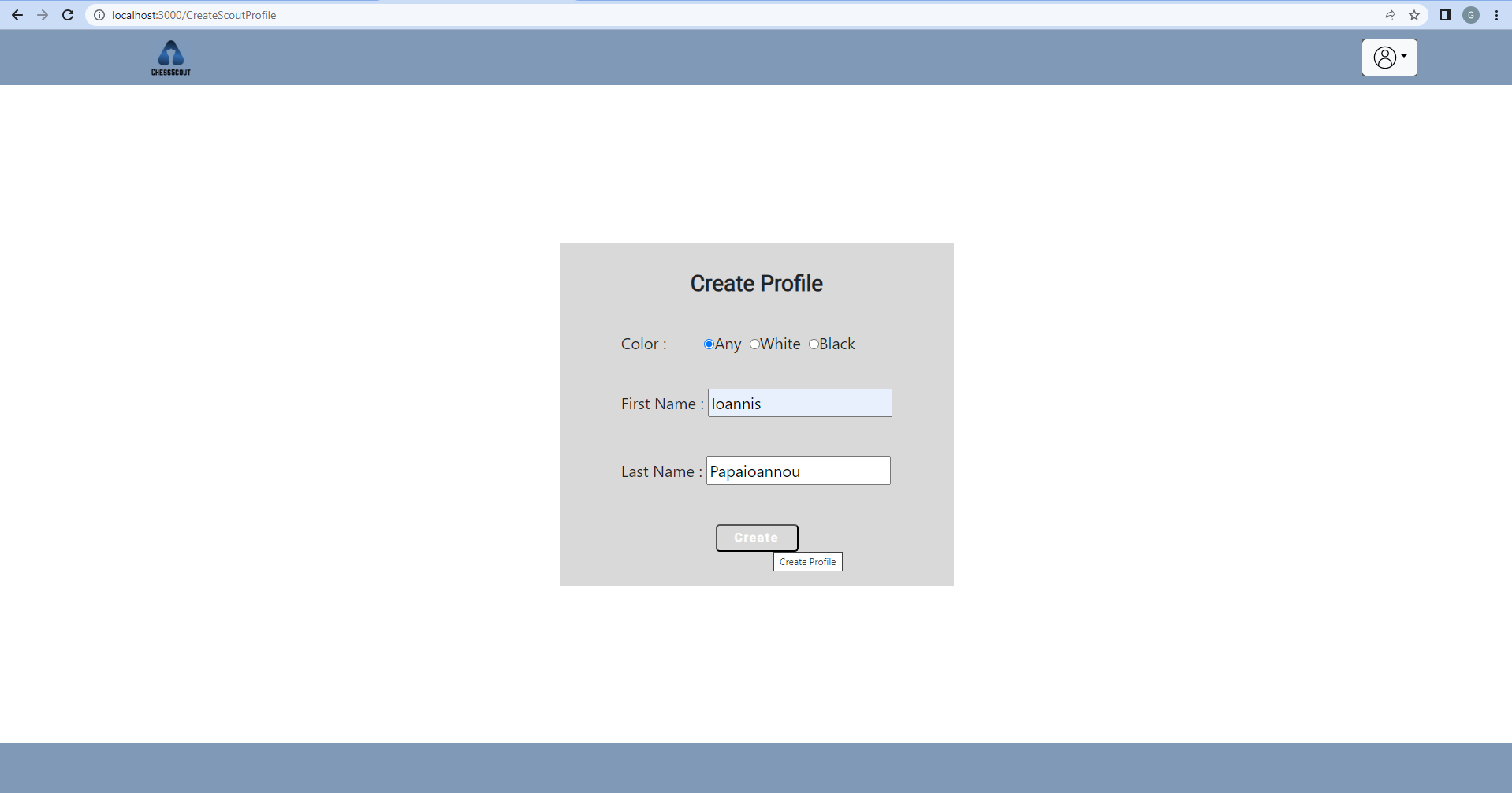
**2) SIGN IN & CREATE A NEW PROFILE**

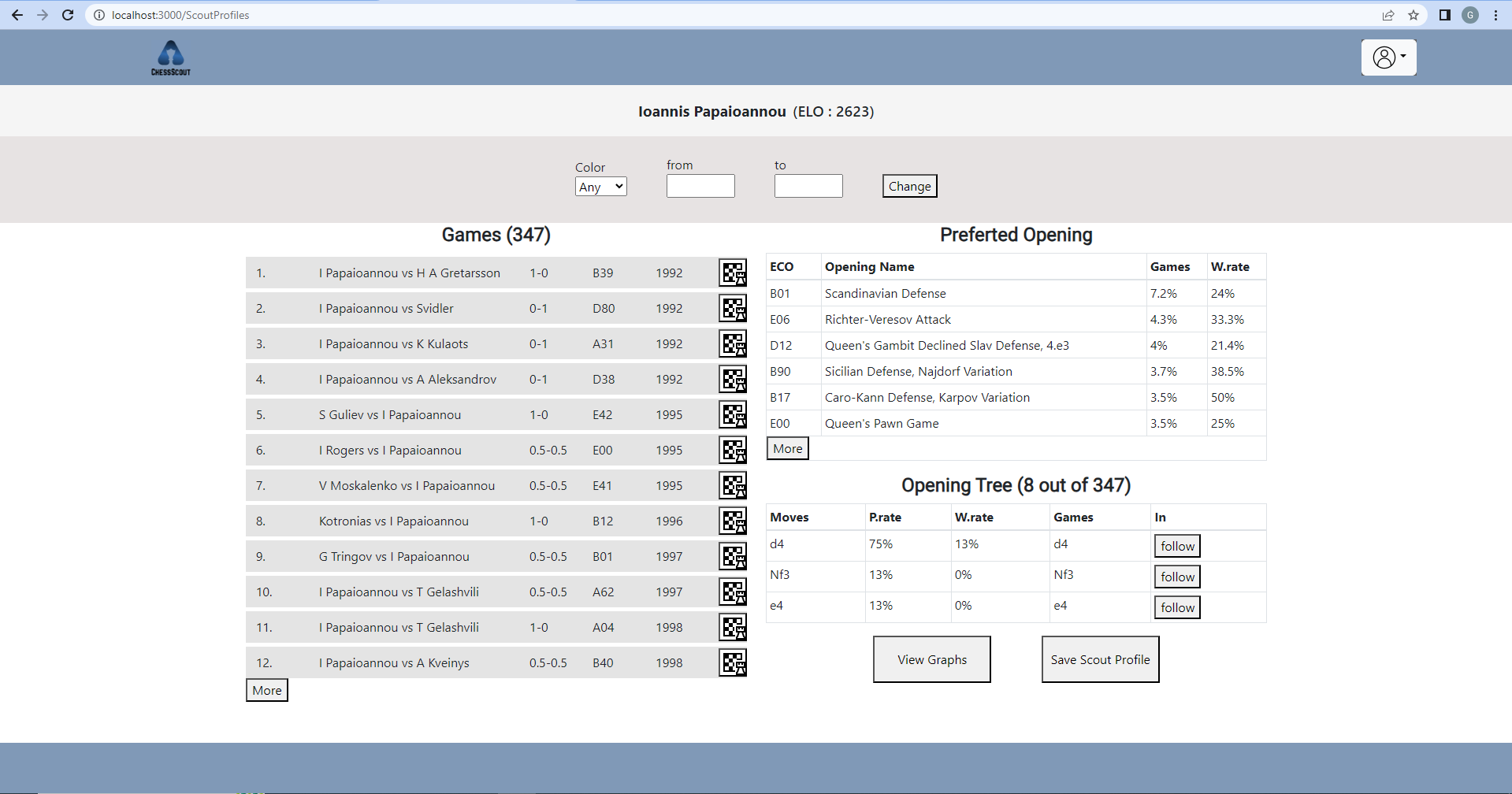
For this example we will use our new user that just created. In the sign in page we fill the form and click on the Sign In button.



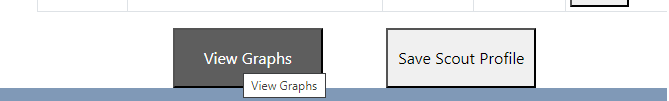
The application navigates us to the Home Page. To create a new profile we click on Create Scout Profile.



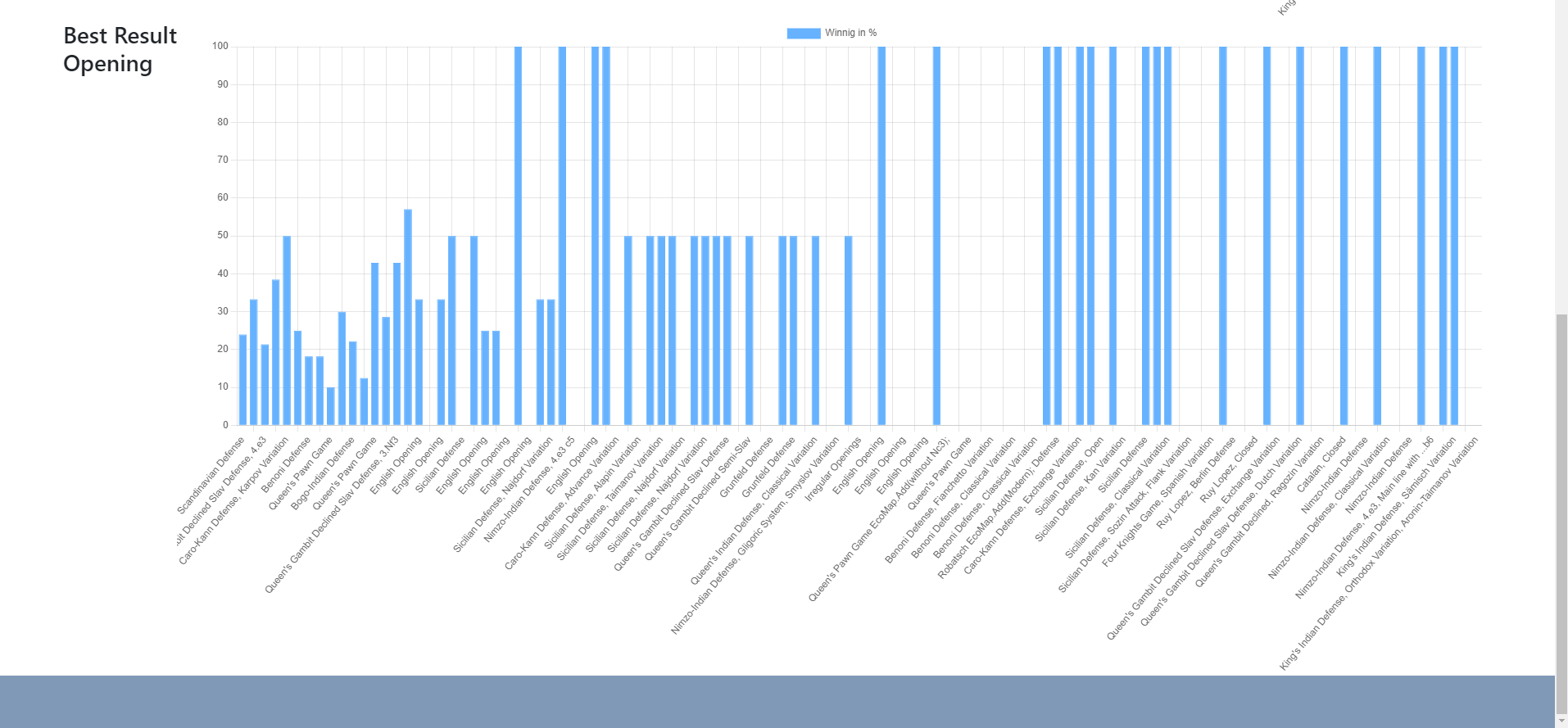
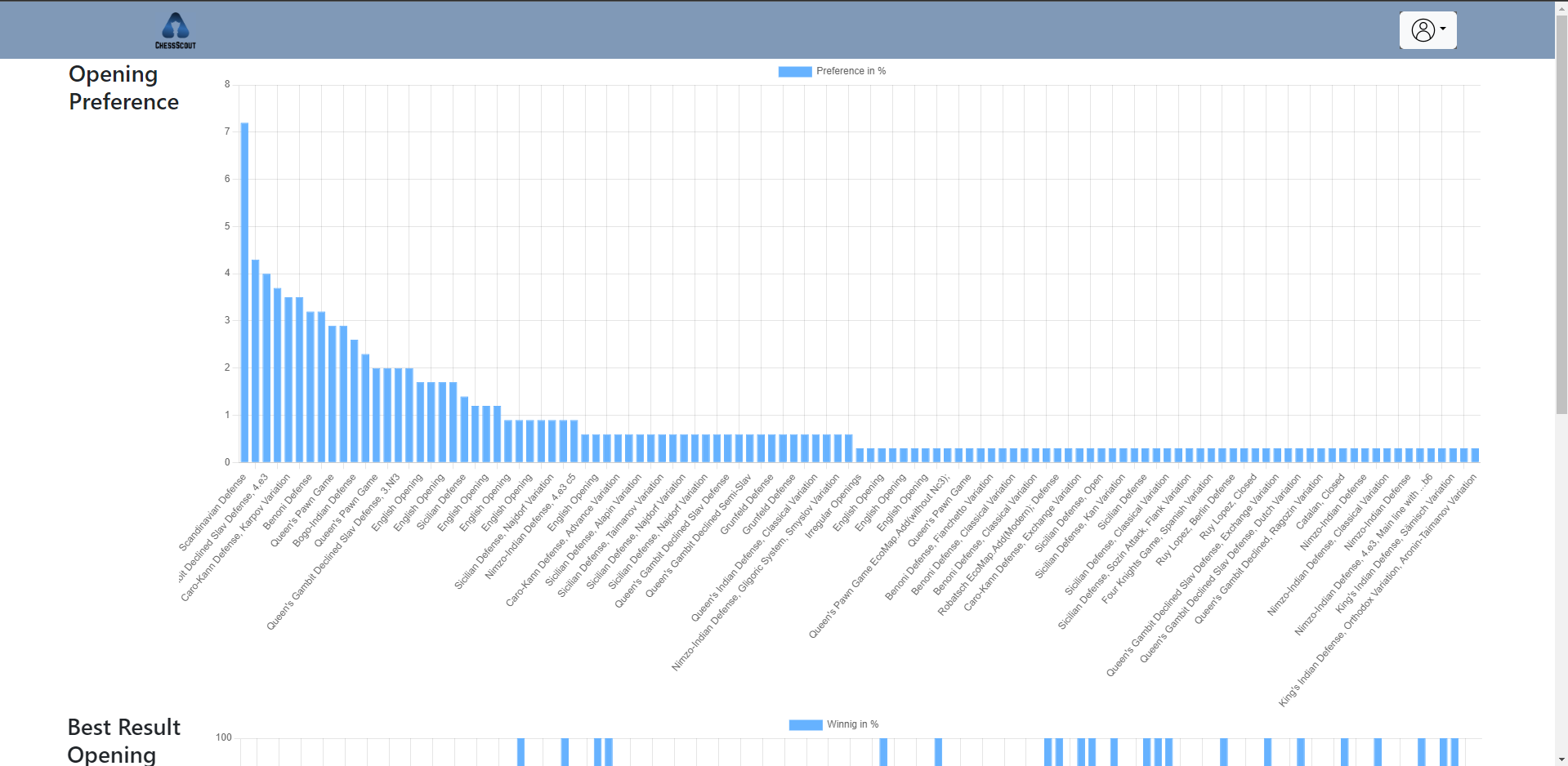
In the Create Profile form that appeared we created a profile for the current No.1 player in Greece Ioannis Papaioannou for any color. 

Our application downloads all player’s games and calculates the preferred openings (P.rate) and winning rates (W.rate) for each opening. It also generates an opening tree, showing the P.rate and W.rate for every move. Finally, the application displays all this vital information to the player.

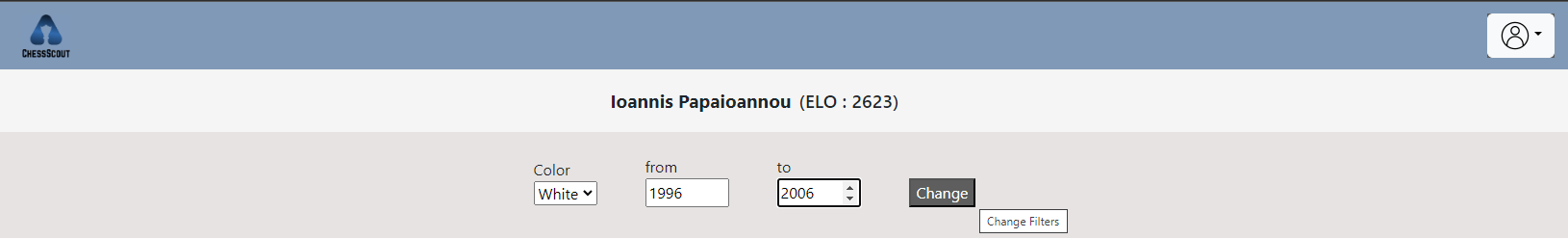
**3) View profile’s graphs**

In the profile that we created before we will display the graphs. In order to display the graphs we click the button View Graphs right in the center.

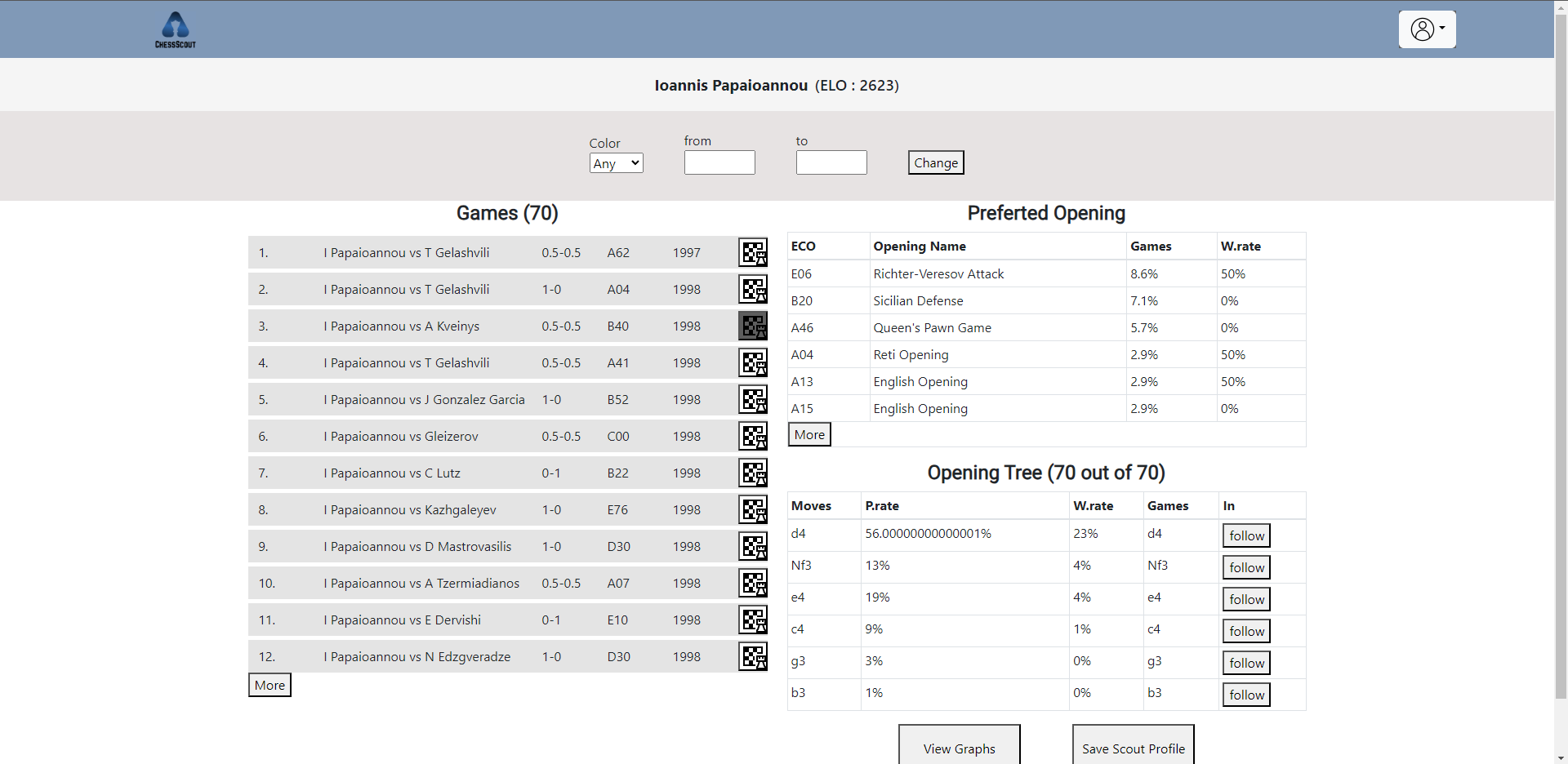
View Graphs page of the profile contains 2 graphs. The first one is the Opening Preference which displays the P.Rate for every opening. The second one is the Best Result Opening which displays the W.Rate for every opening.

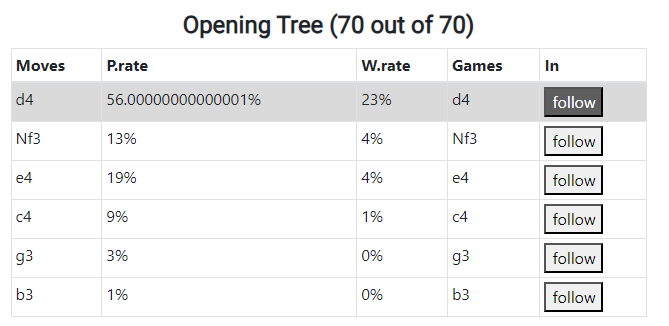
****

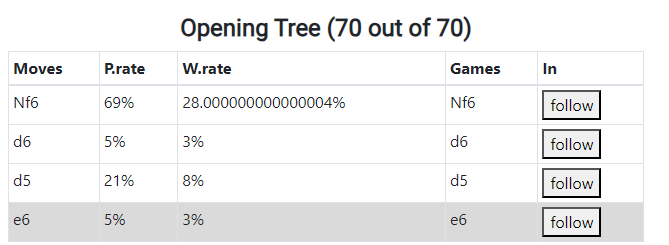
**4) Apply filters on the profile and use the opening tree**

For this test we will use the profile from test no.2 . At the top of the page, there is a filter bar that allows users to modify the games they want to analyze. Let's get the games that player played as white from 1996 to 2006. To apply the filters click on change. 

After the change, the system modifies the games and all the data (Preferred Opening and Opening Tree).

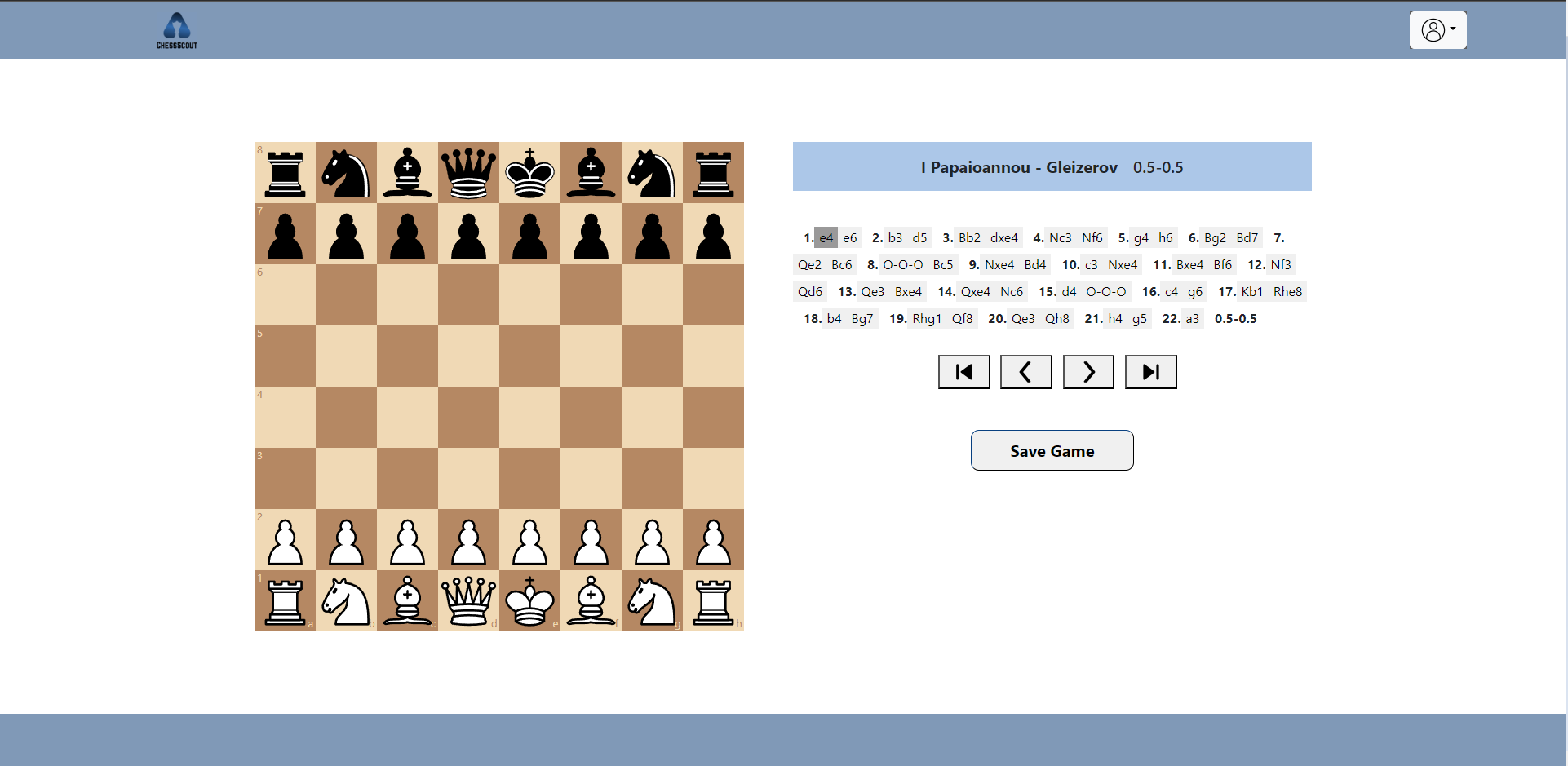


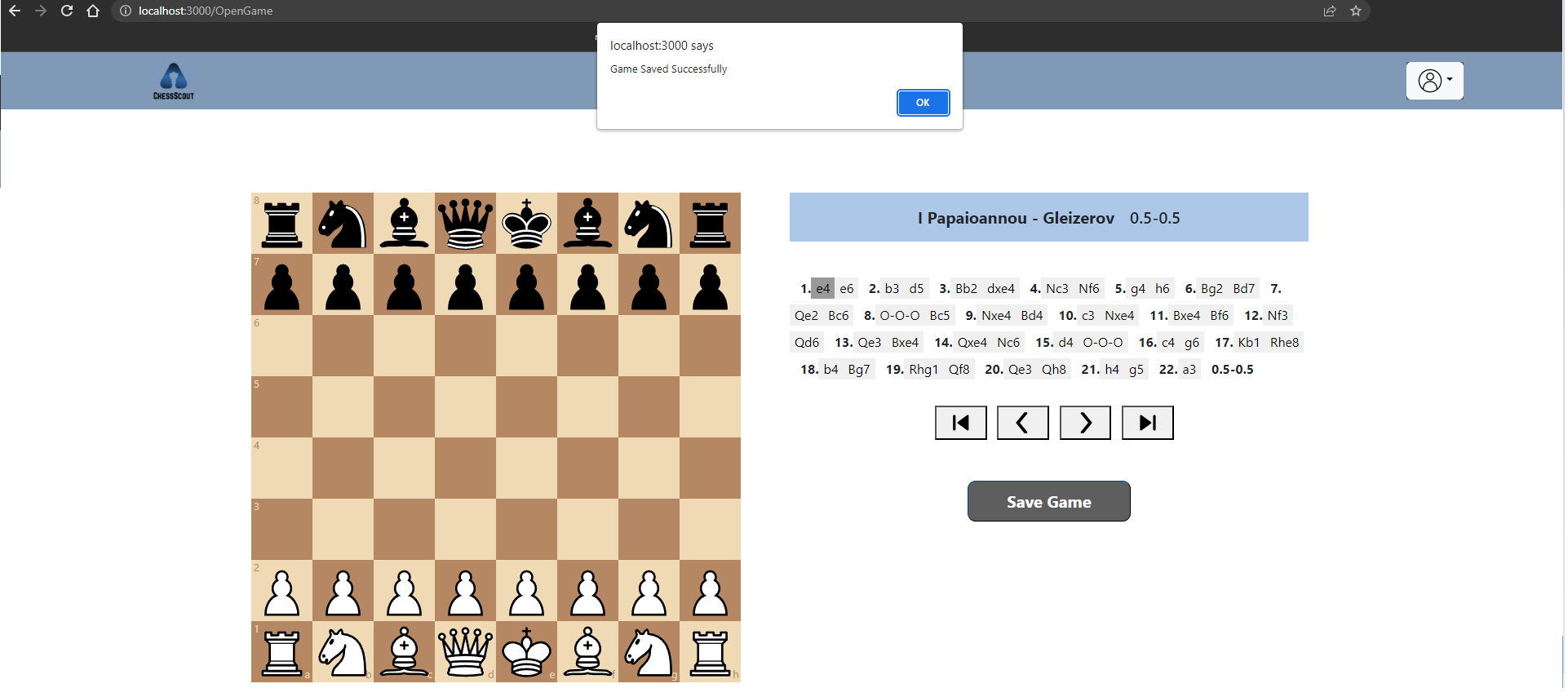
Next we will use the Opening Tree. In our example, we want the most popular move that is played after the move d4 against mr.Papaioannou. So, we click on the follow button next to move d4.

As we can see below the most popular move against d4 is Nf6 (P.Rate 69%).

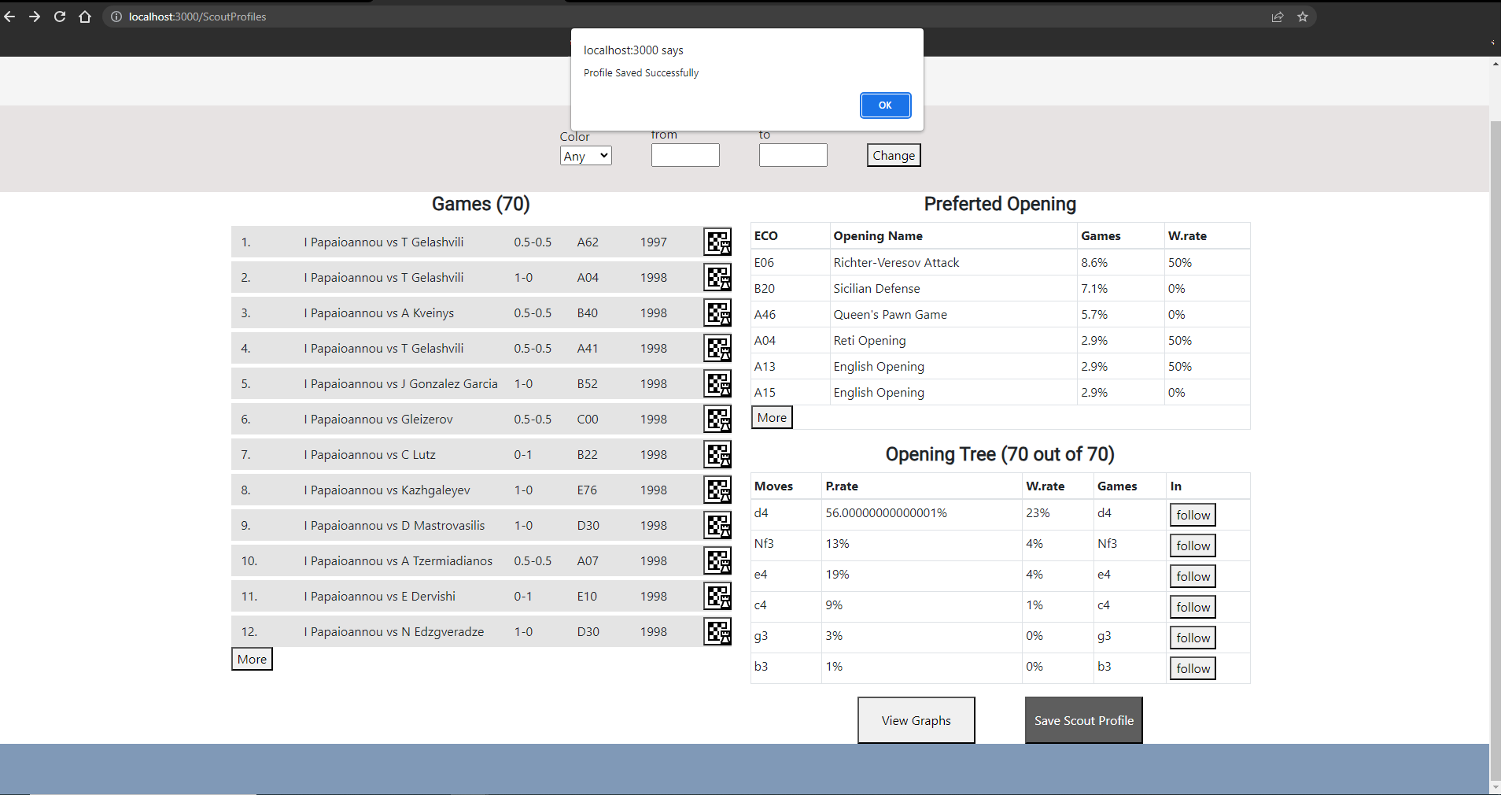
**5) Open and save a game**

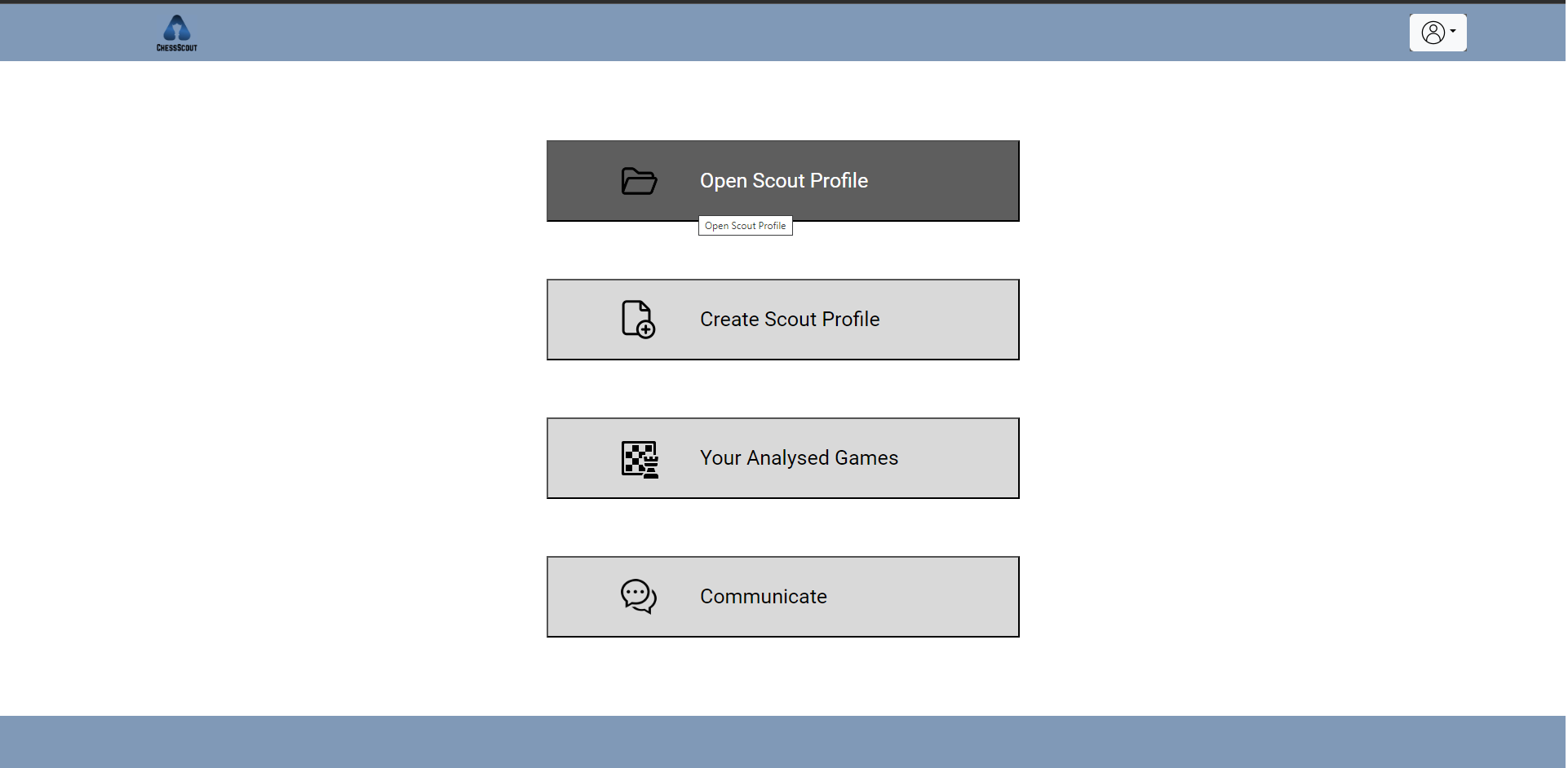
Users can choose and save any game they want from a profile. From the profile above, we select a random game and open it by clicking the icon next to it.

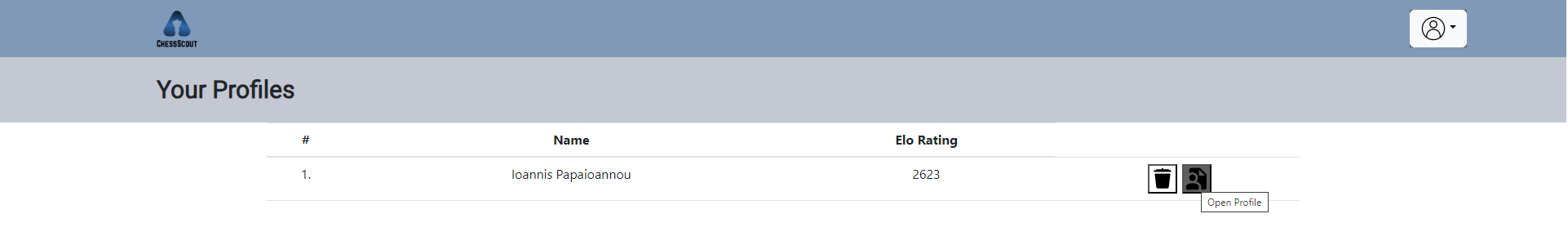
The game opens in the starting position that we can with the arrows in the screen or by clicking a move in the screen.

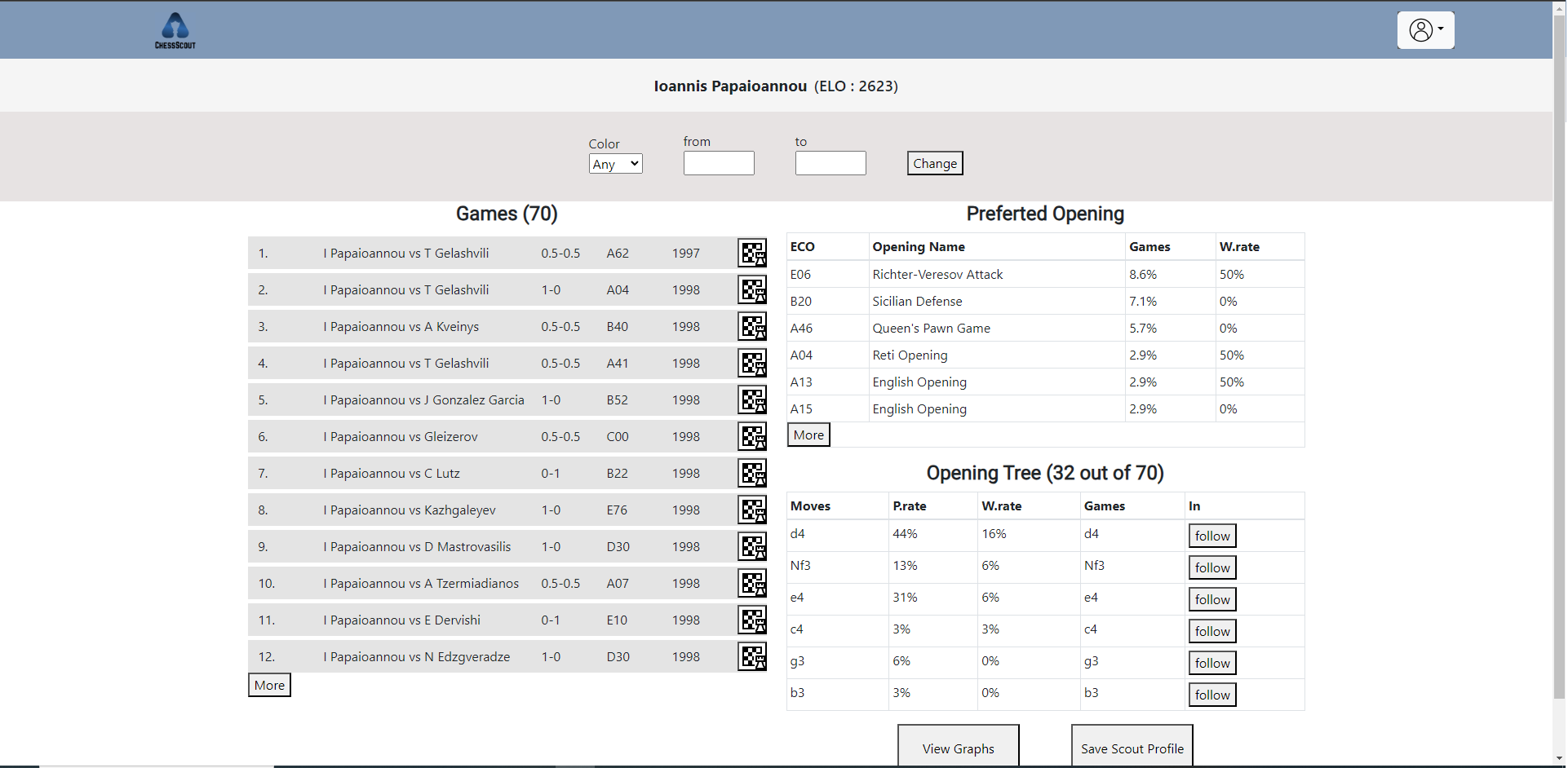
In order to save this game we simply click on the Save Game button under the arrows.

**6) Save and open a profile**

Now we need to save the profile we created. To do this we click the Save Profile button right down the profile page.

Our profile just saved. Next we have to open this profile again. In system’s Home Page we click on the first option (Open Scout Profile), which contains all the saved profiles.

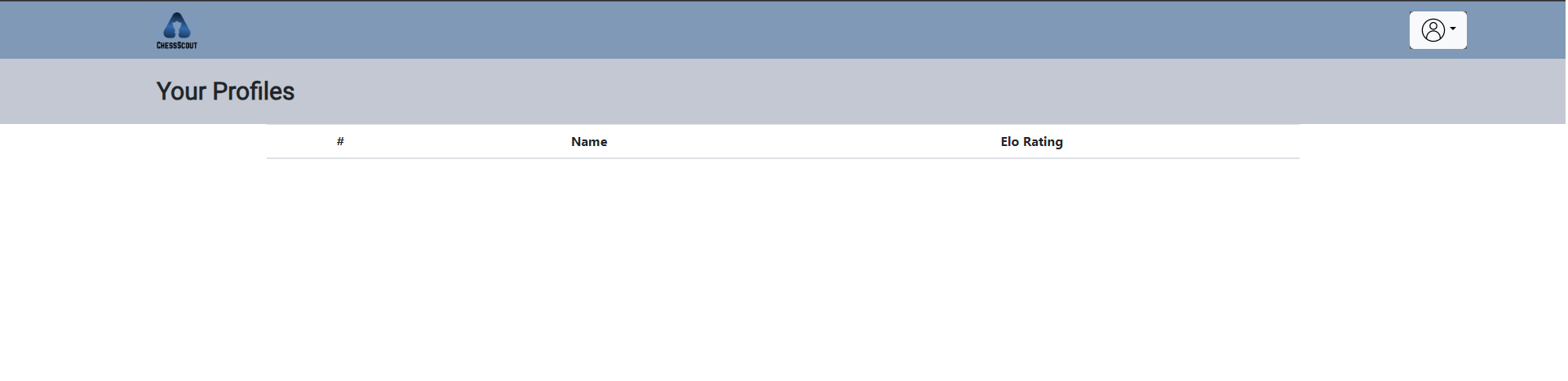
In the Open Scout Profile page we can see all the saved profiles. To open one of them just click the open profile button.

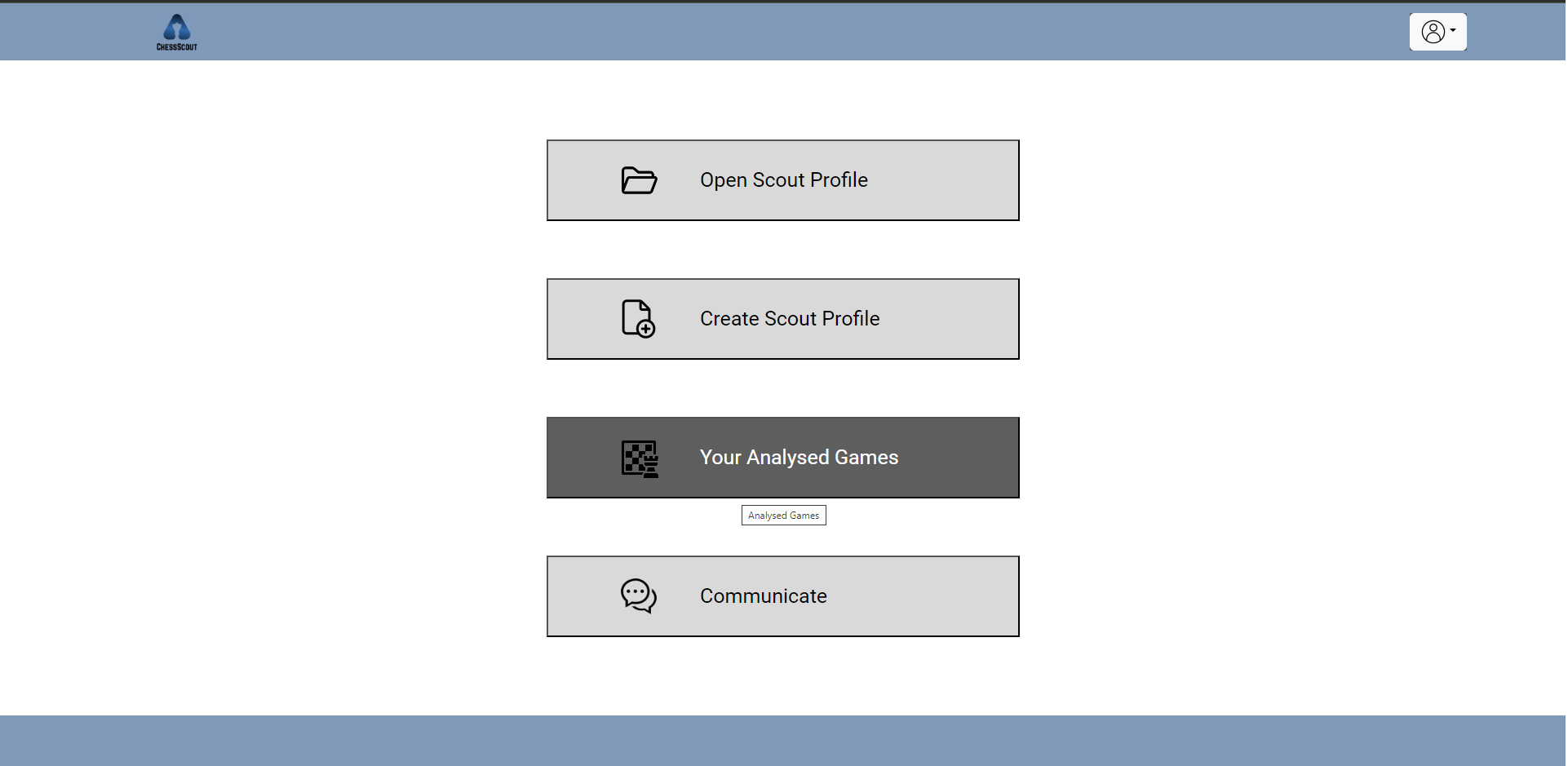
The profile opens again.

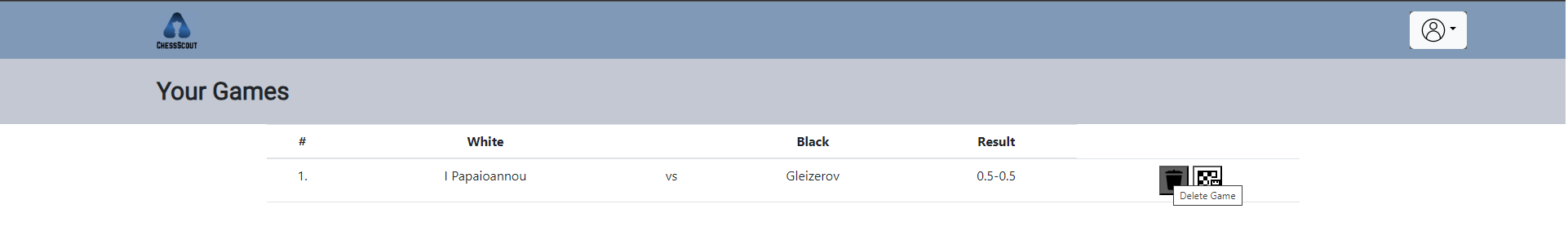
**7) Delete profile and game**

Users can not only save a profile or a game, but they can also delete them too. When we want to delete a profile, the first thing we have to do is go to the open scout profile. Then we simply click on the delete profile button. 

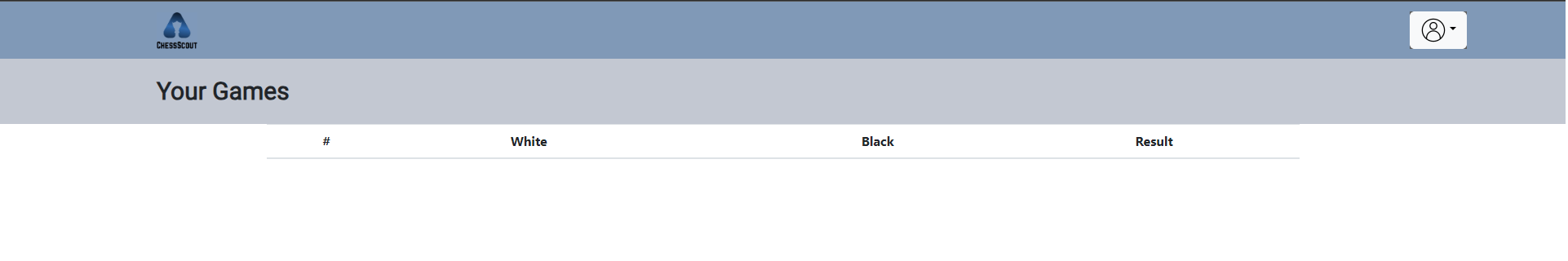
As we can see the profile is deleted.



In order to delete a saved game, we have to go back to the home page and click on the 'Your Analyzed Games' button. 

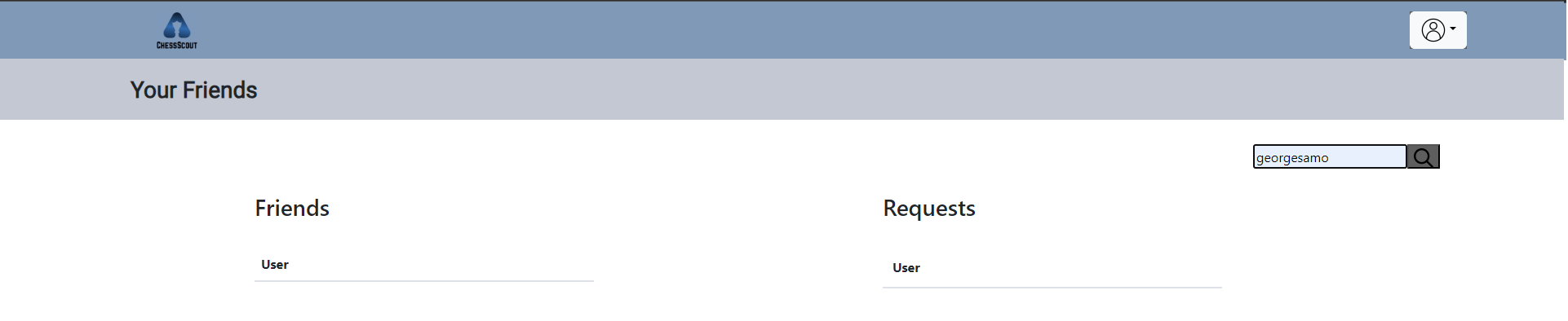
After we click on the Delete Game button to delete the game.

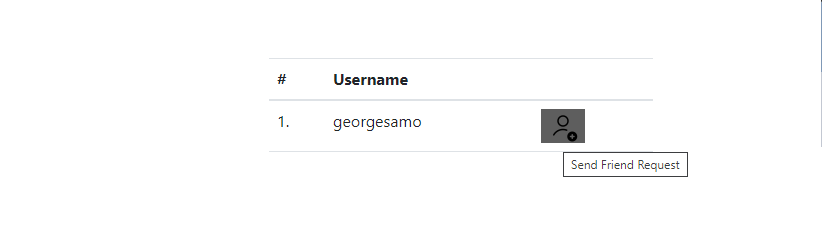
The game is deleted.



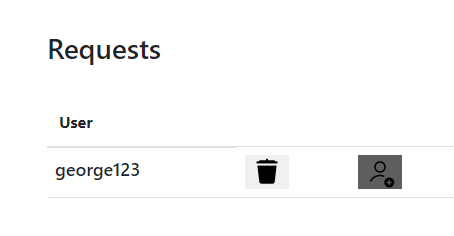
**8) Search a user and send a message**

For our last test, we will try to add a new friend and start a conversation with him/her. First, we click on the Communicate button on the home page.

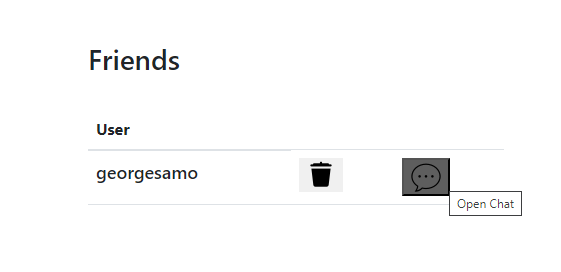
Next, we will write the username that we want in the search bar located at the top right of the page. After entering the username, we click on the search button. (for this example we will use the user “georgesamo”)

A list of users appears. So, we will send a friend request to the user that we want. 

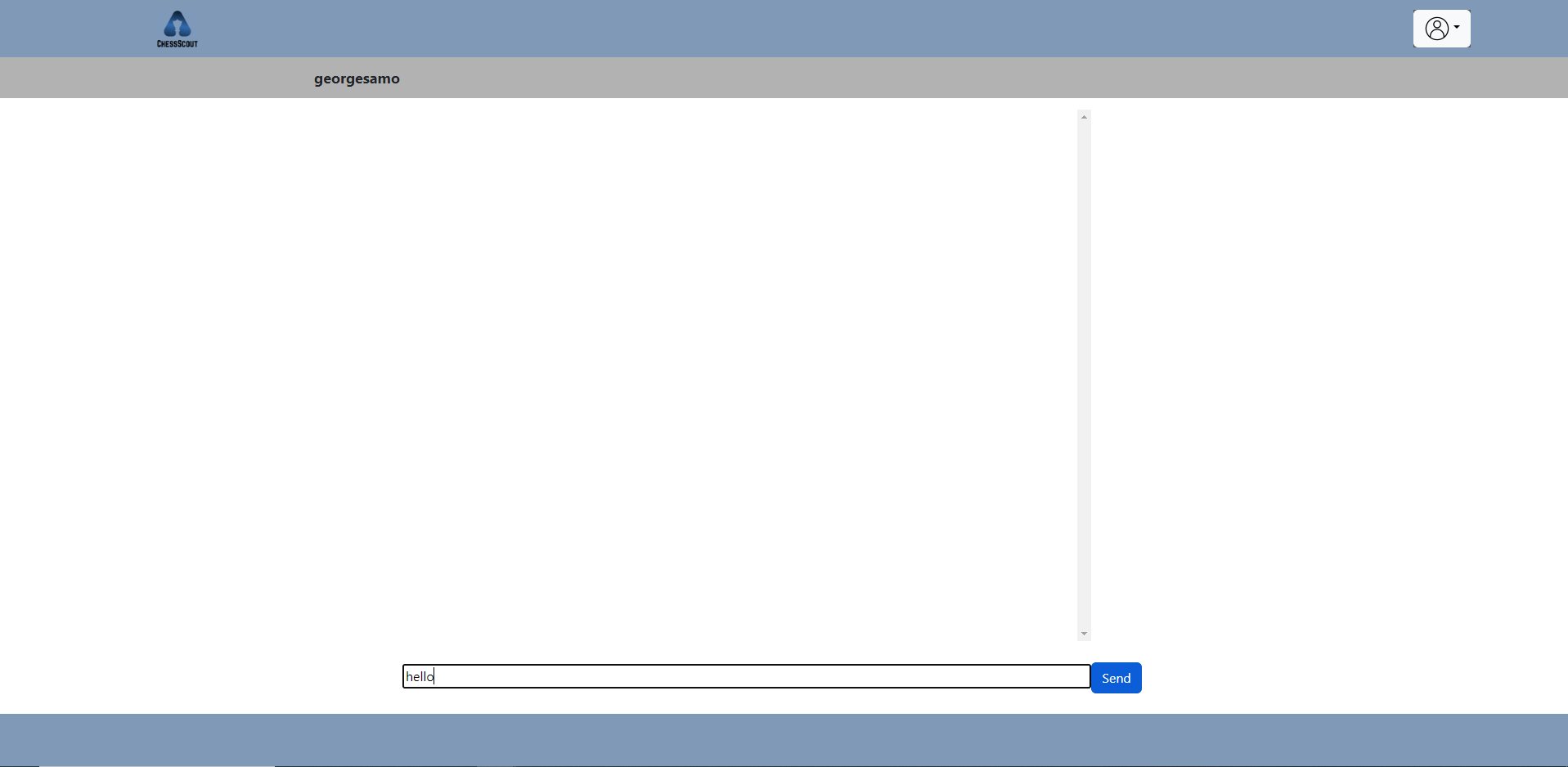
The other user “georgesamo” has to accept the request that received from “george123”.



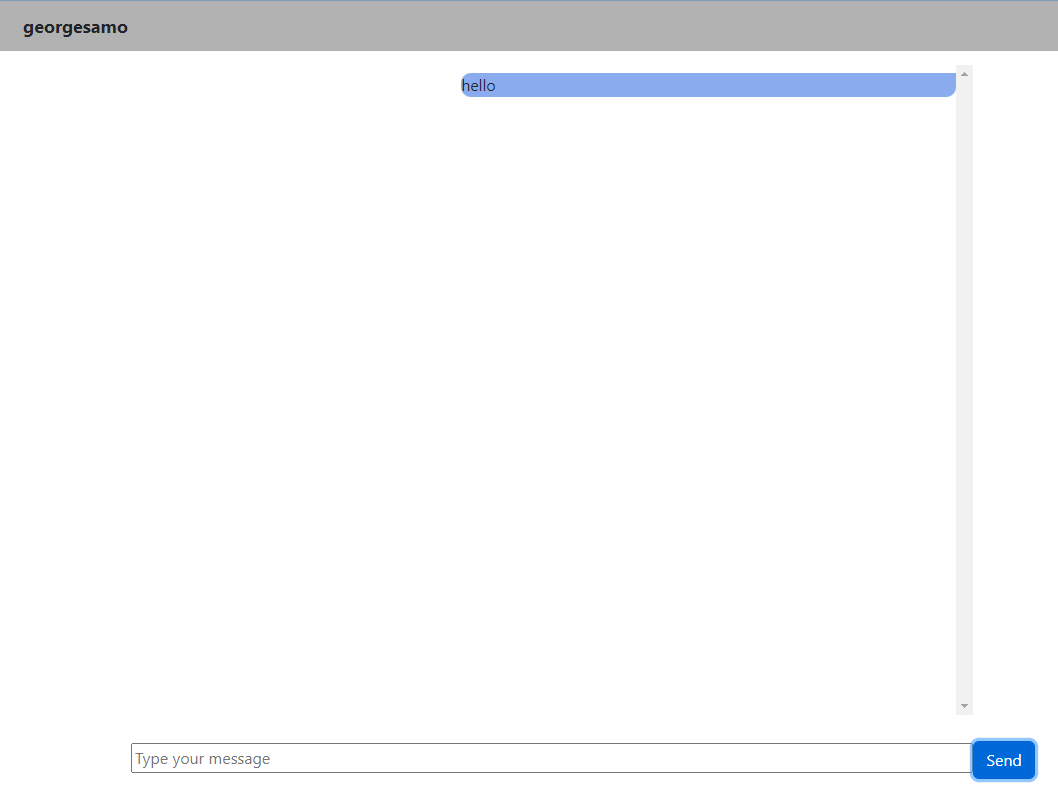
After accepting the request, the user will appear in our friends list. To open the chat with him we click on the Open Chat button.

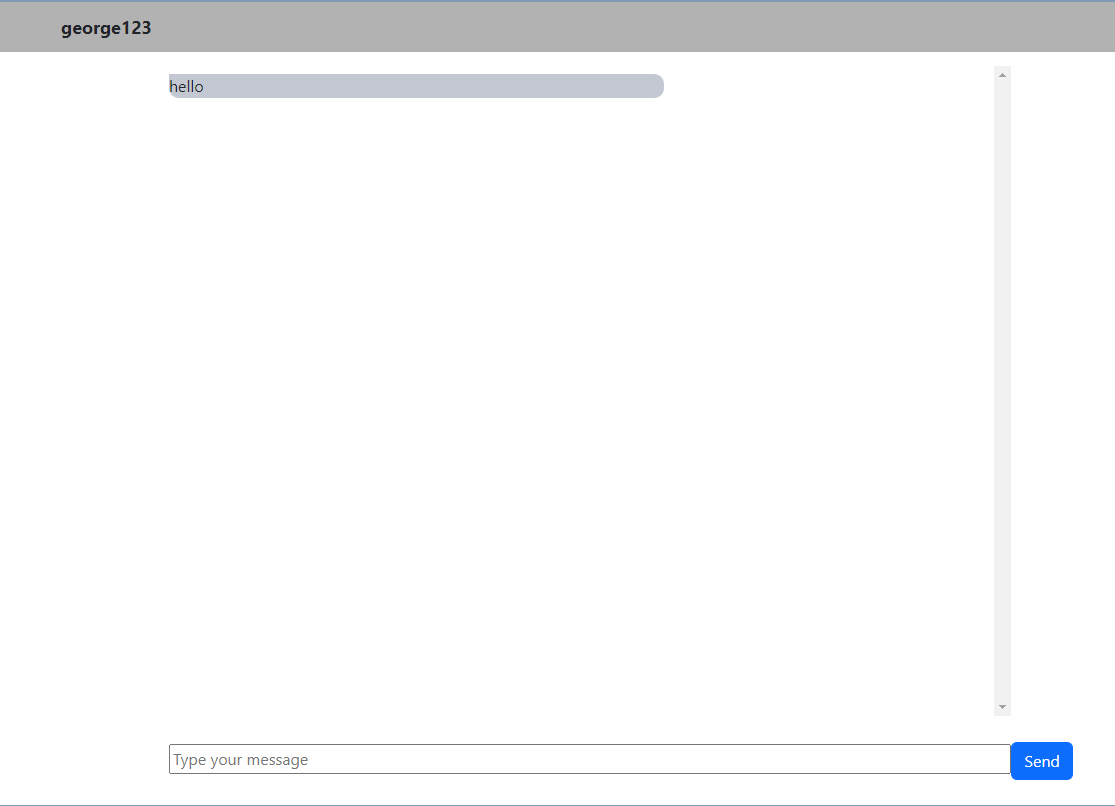


Now we send a message to “georgesamo”.



Message looks sent.



. The last thing we need to check is whether the other user (georgesamo) received the message successfully. The other received the message successfully as we can see..

**Implementation Details**

**6.1 ReactJS Components**

As mentioned above, our UI is developed using the ReactJS library. It is comprised of 3 different components, of which 2 are static and remain the same on every page, while the other is dynamically changed by a navigator.

**Static Components** :

1. Header
2. Footer

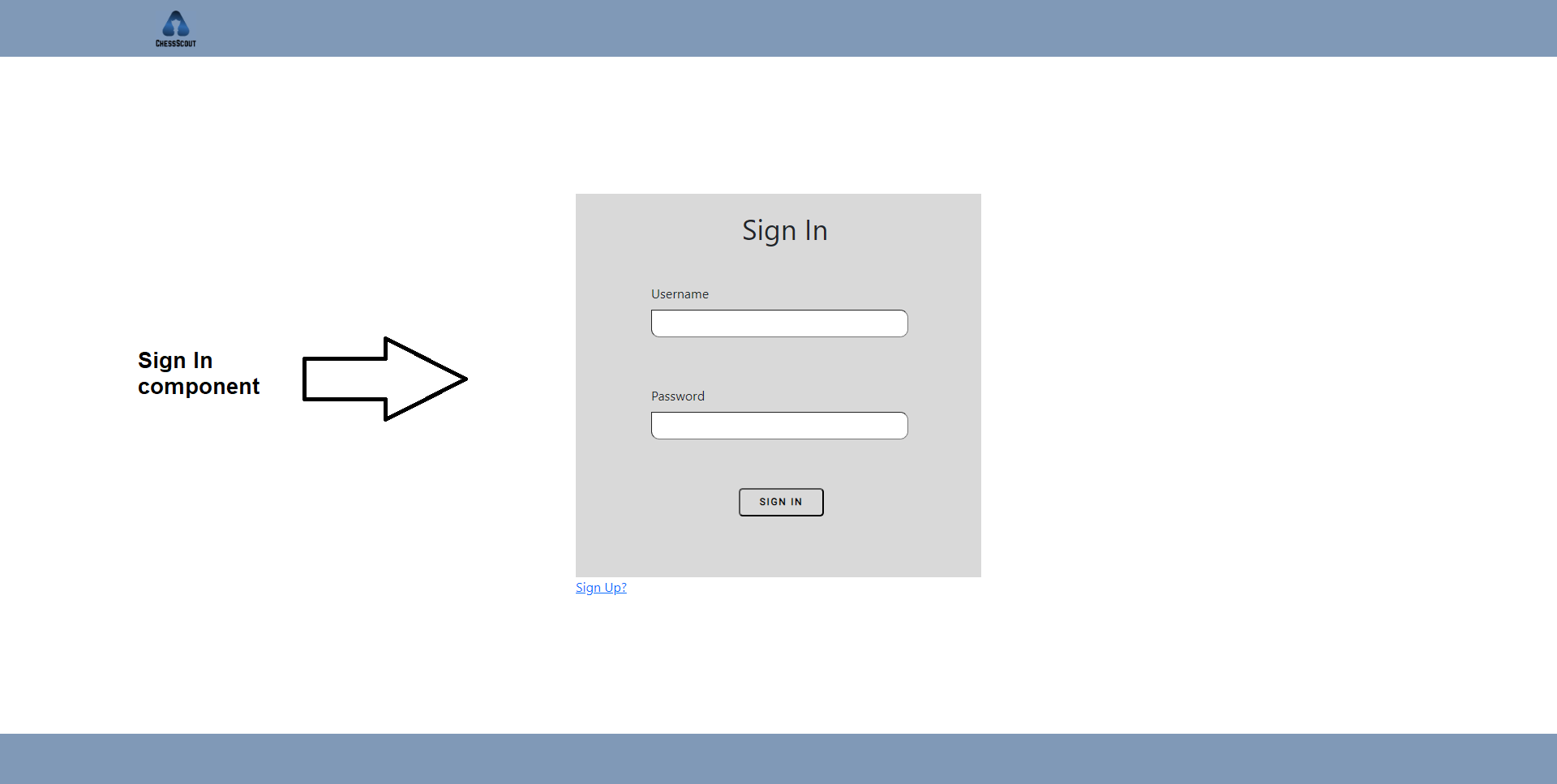
**Dynamic Components** :

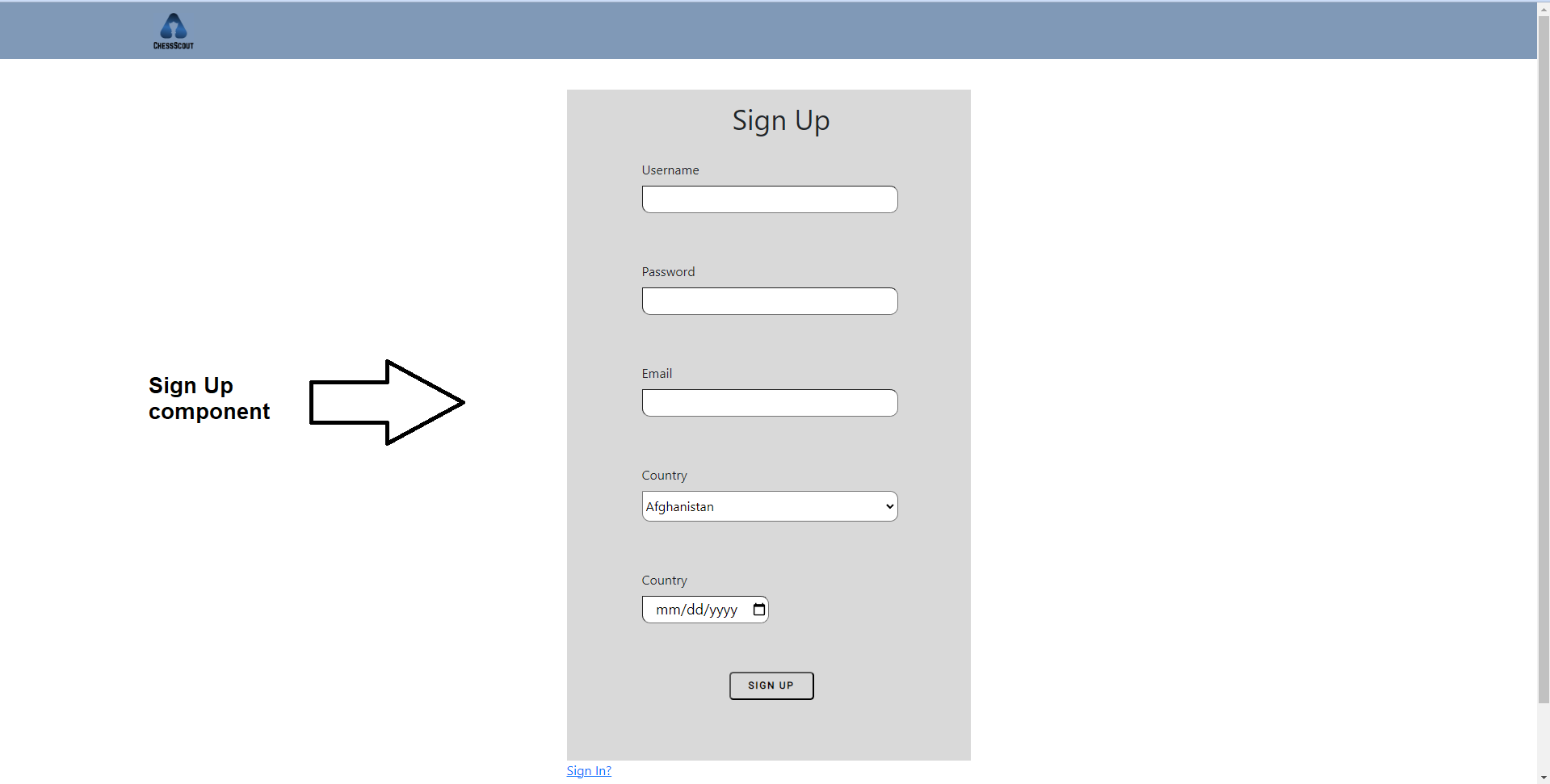
1. Body

The **static components**, which remain the same on all pages, are the header and the footer. Here is an example of these 2 components.

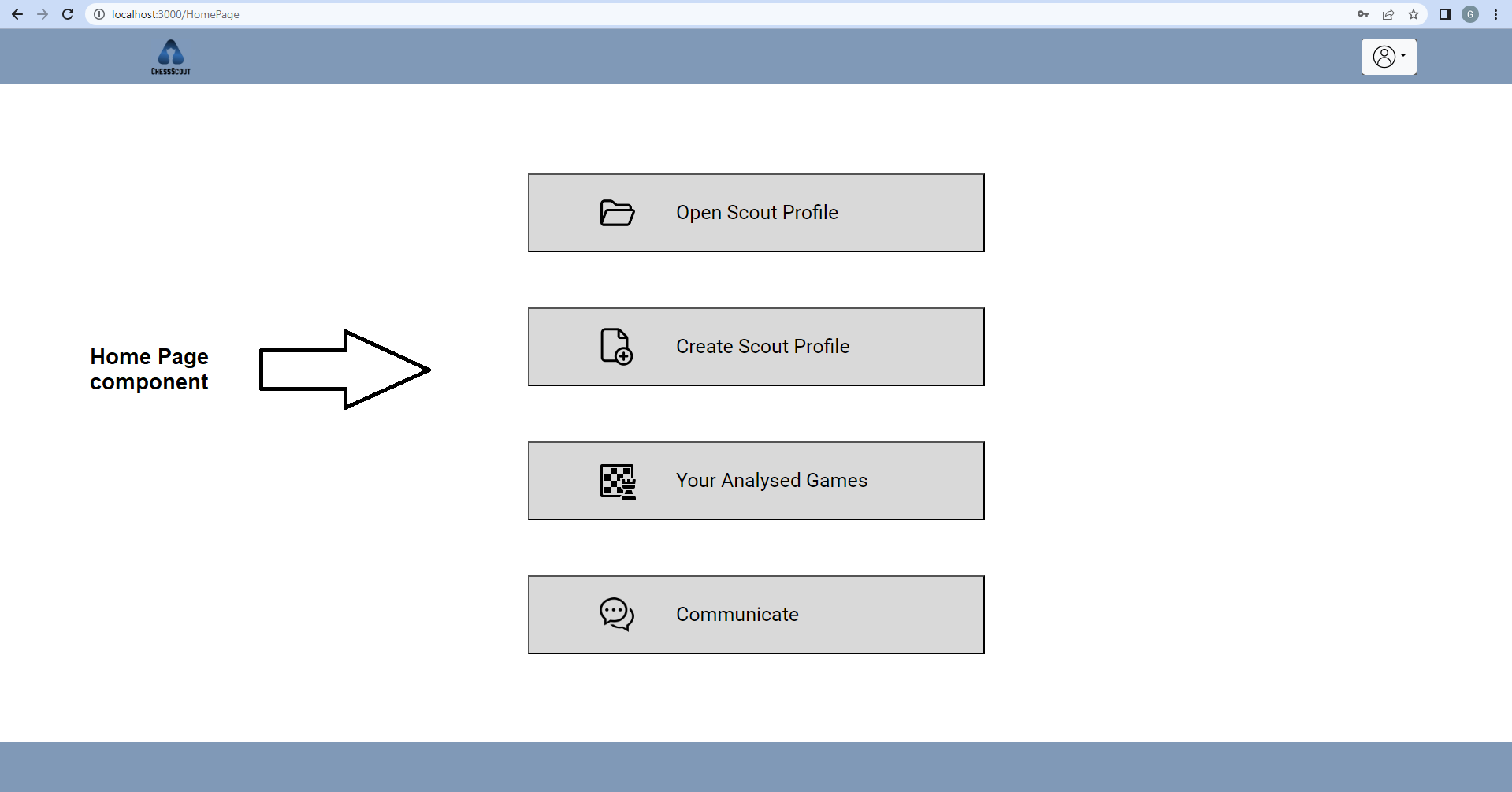
The **dynamic component** called 'Body' is the only component that changes when the user navigates from page to page or when data changes. This component is the one in the center. 

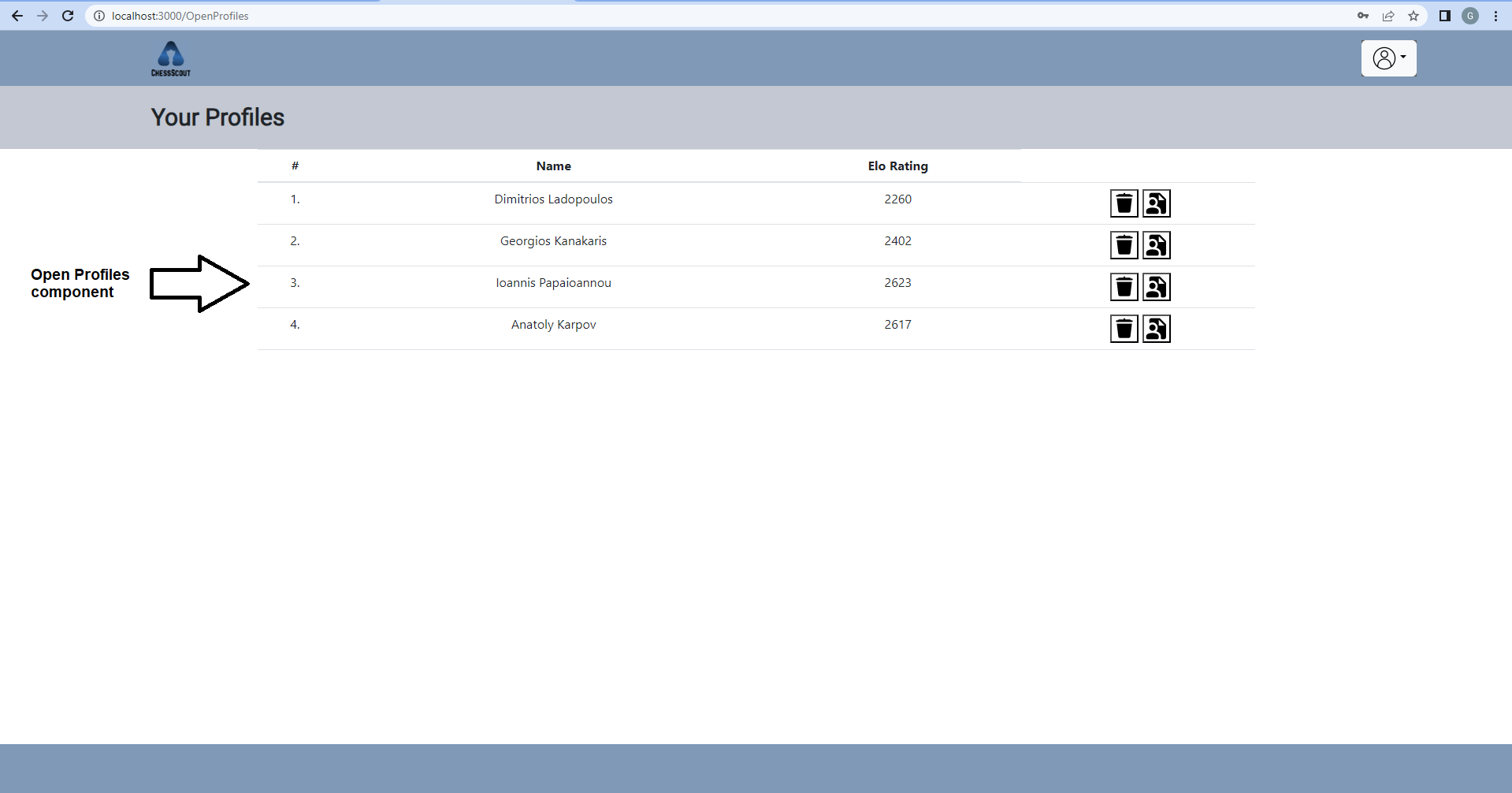
In order to navigate between pages we use the Router component from the “react-router-dom” library. There are 13 paths for navigation (pages).

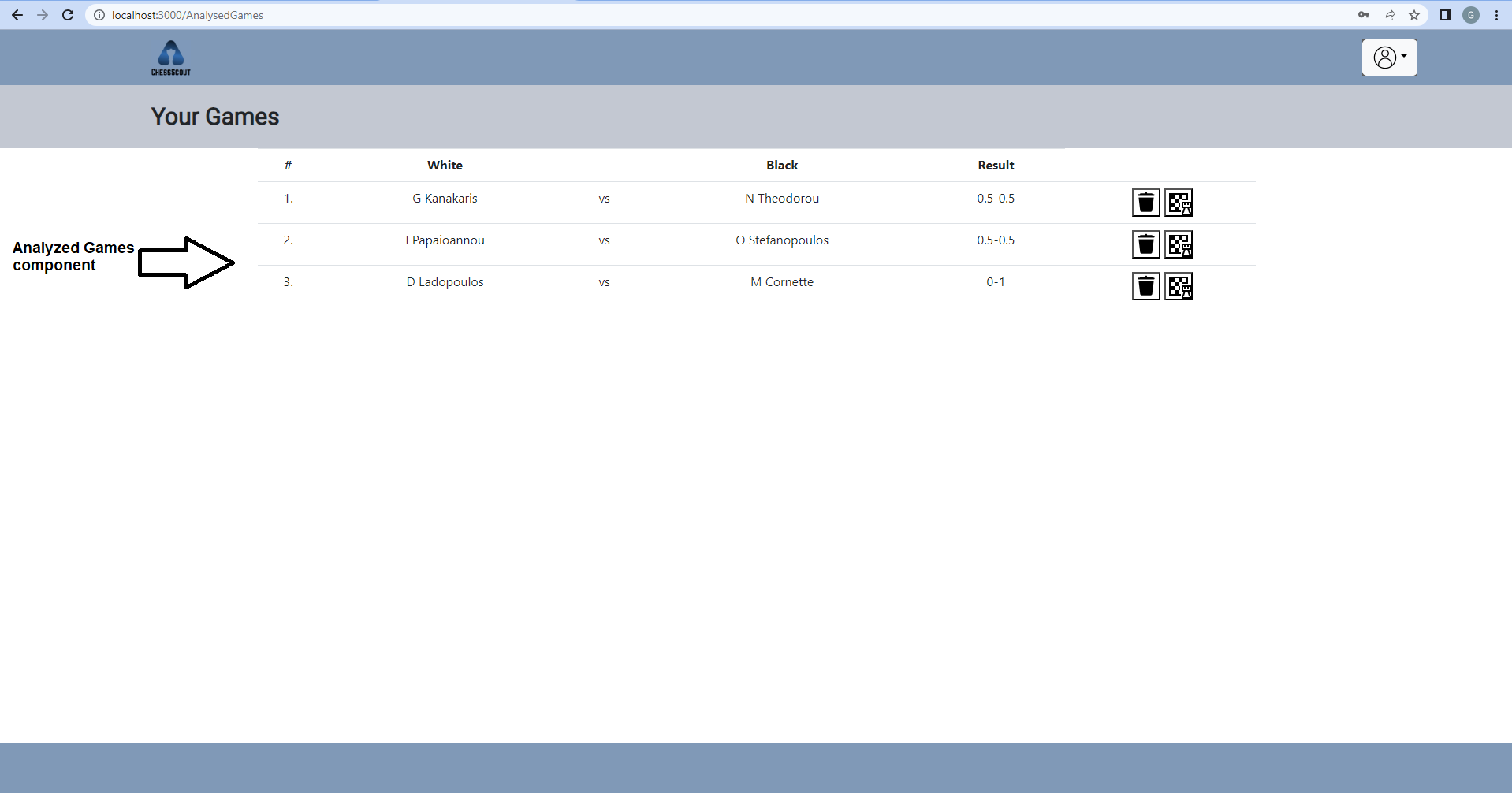
The <DefaultPage/> component is similar to the <SignIn/> and navigates the user to the Sign In page with the Sign In component. The Sign In component includes a Sign In form.

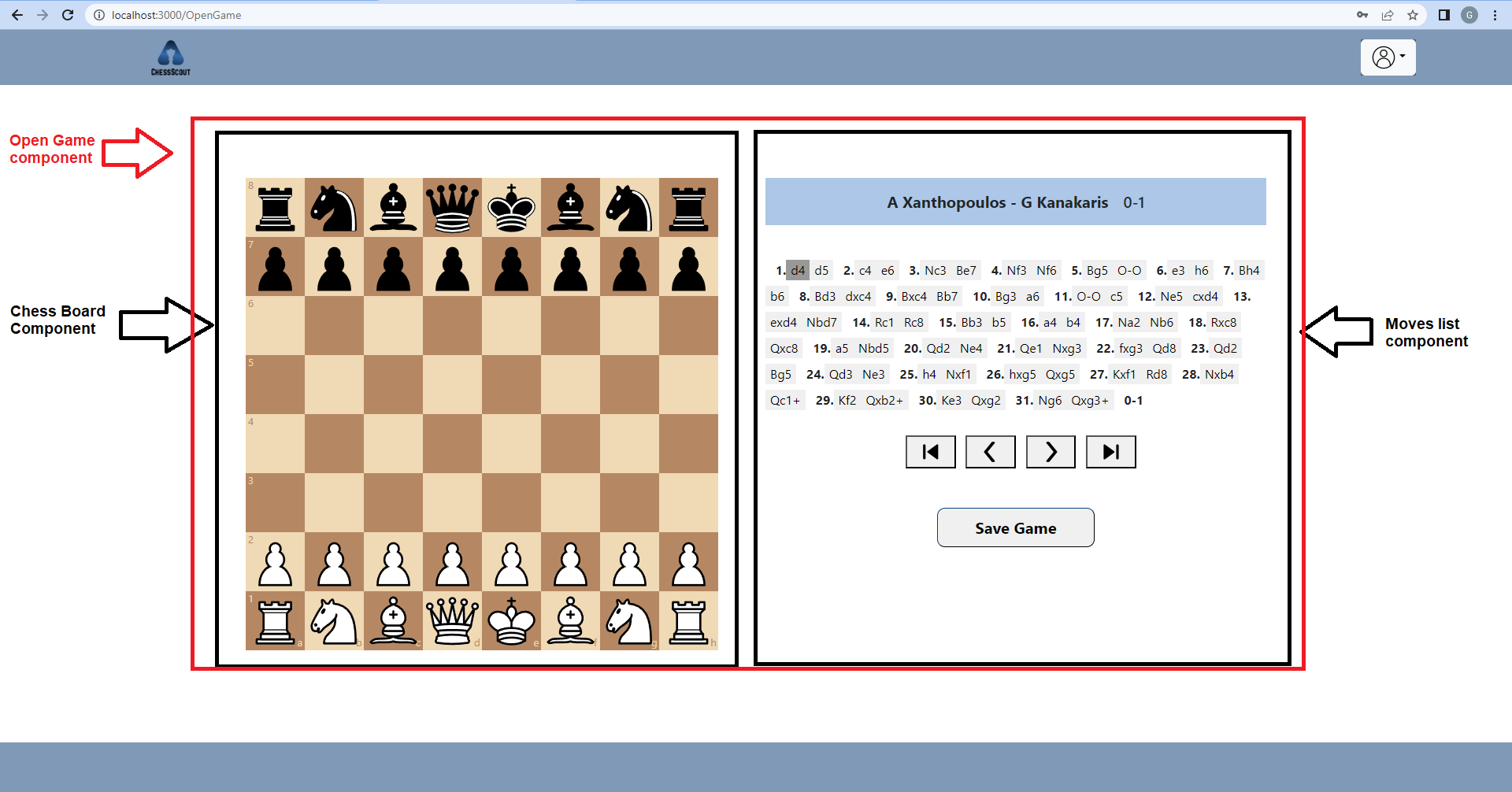
The <SignUp/> navigates the user to the Sign Up page with the Sign Up component. The Sign Up component includes a Sign Up form.

The <HomePage/> navigates the user to the Home Page page with the Home Page component. The Home Page component includes 4 buttons for 4 different pages.

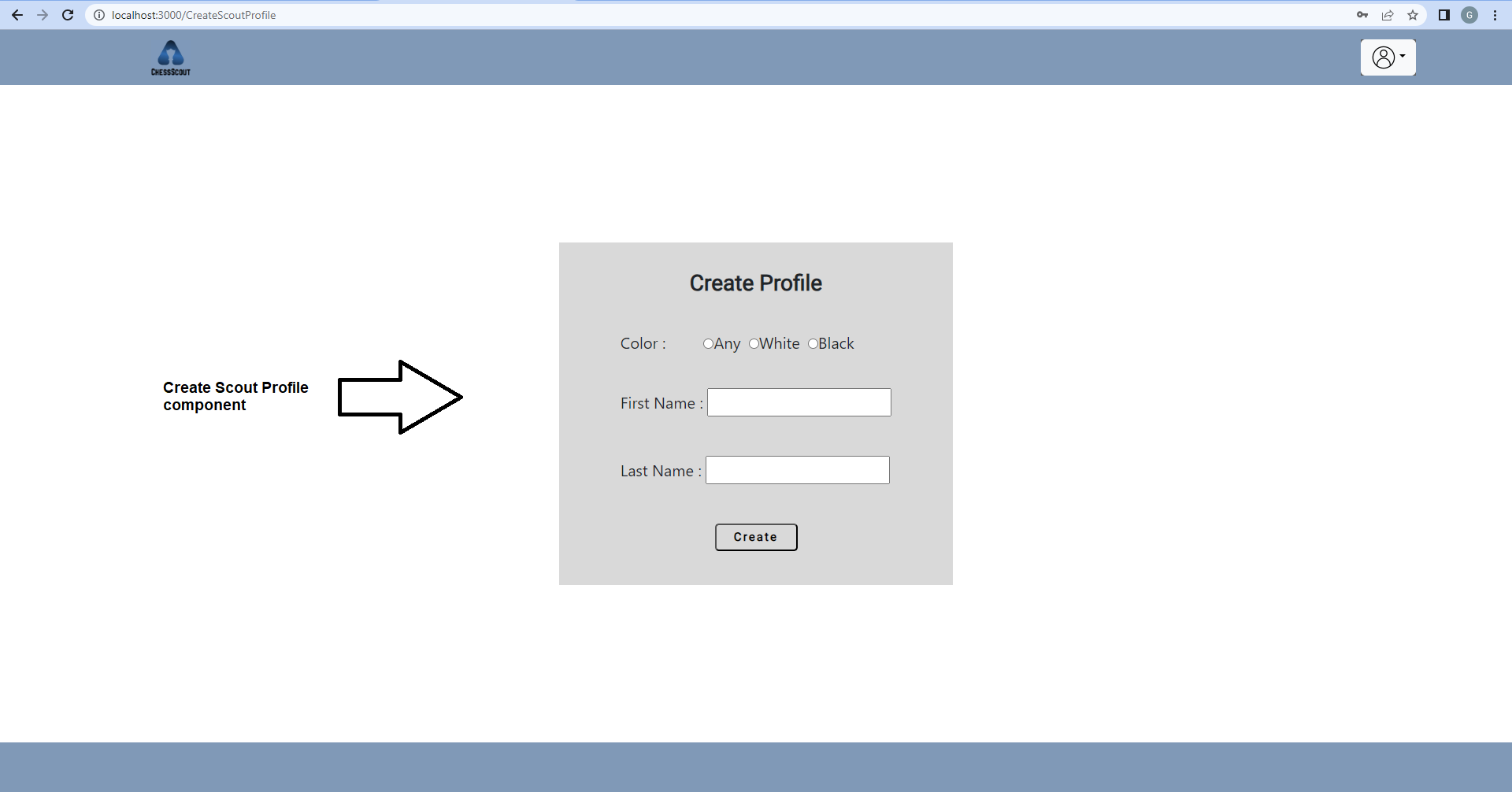


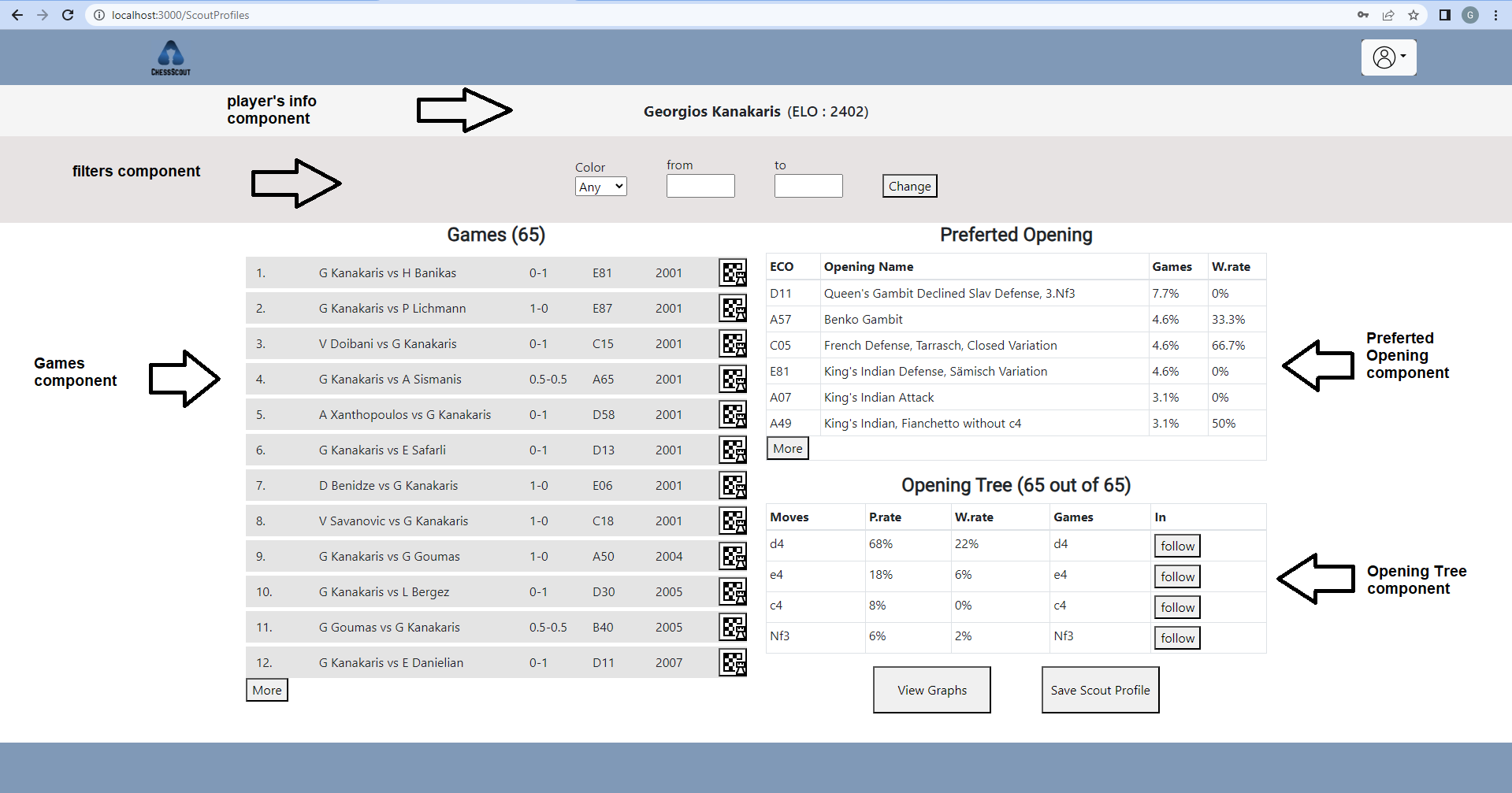
The <OpenProfiles/> navigates the user to the Open Profiles page with the Open Profiles component. The Open Profiles component includes the list of saved profiles (one div tag for every profile).

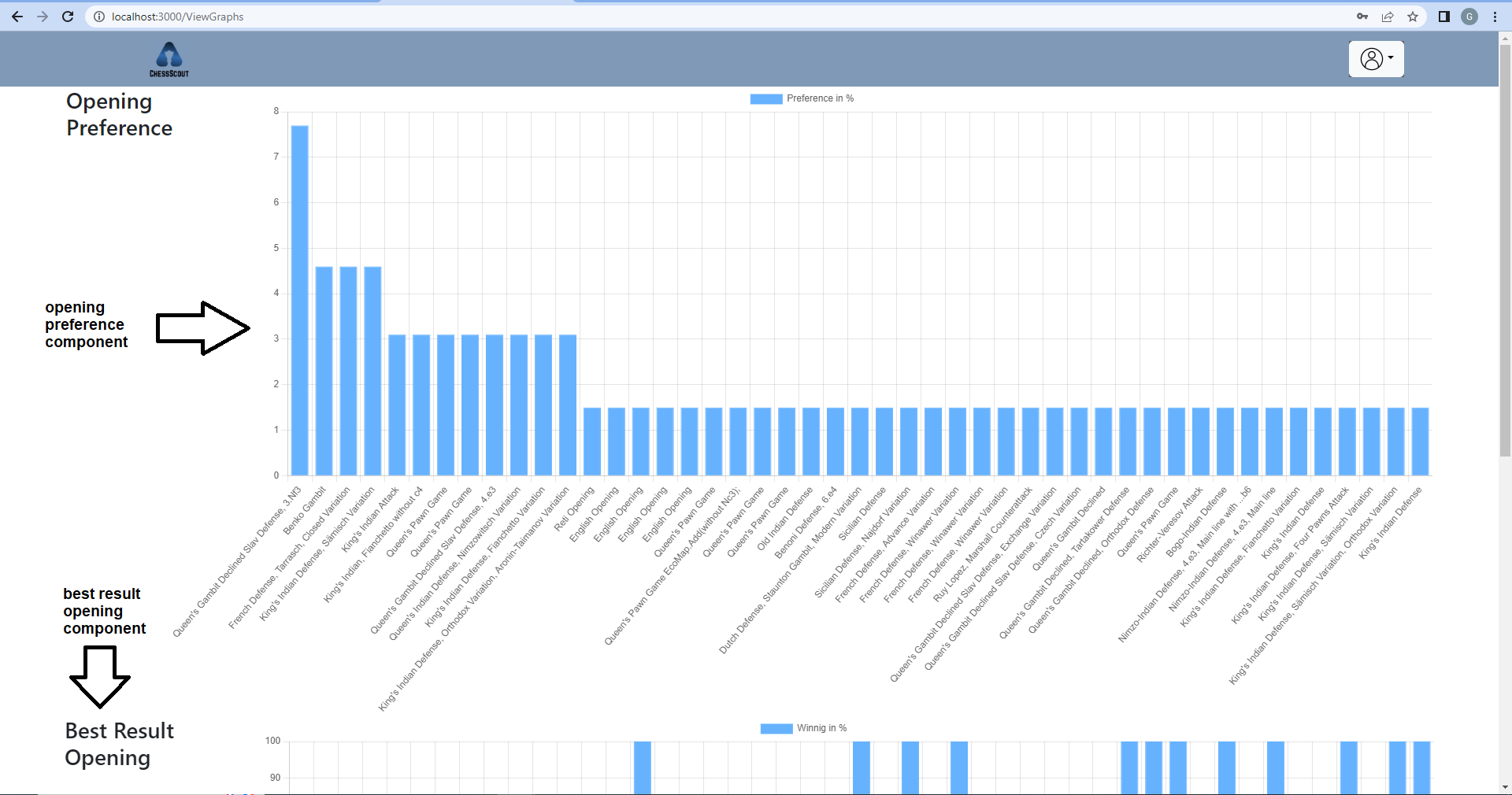
The <AnalysedGames/> navigates the user to the Analysed Games page with the Open Game component. The Analysed Games component includes the list of saved games (one div tag for every game). 

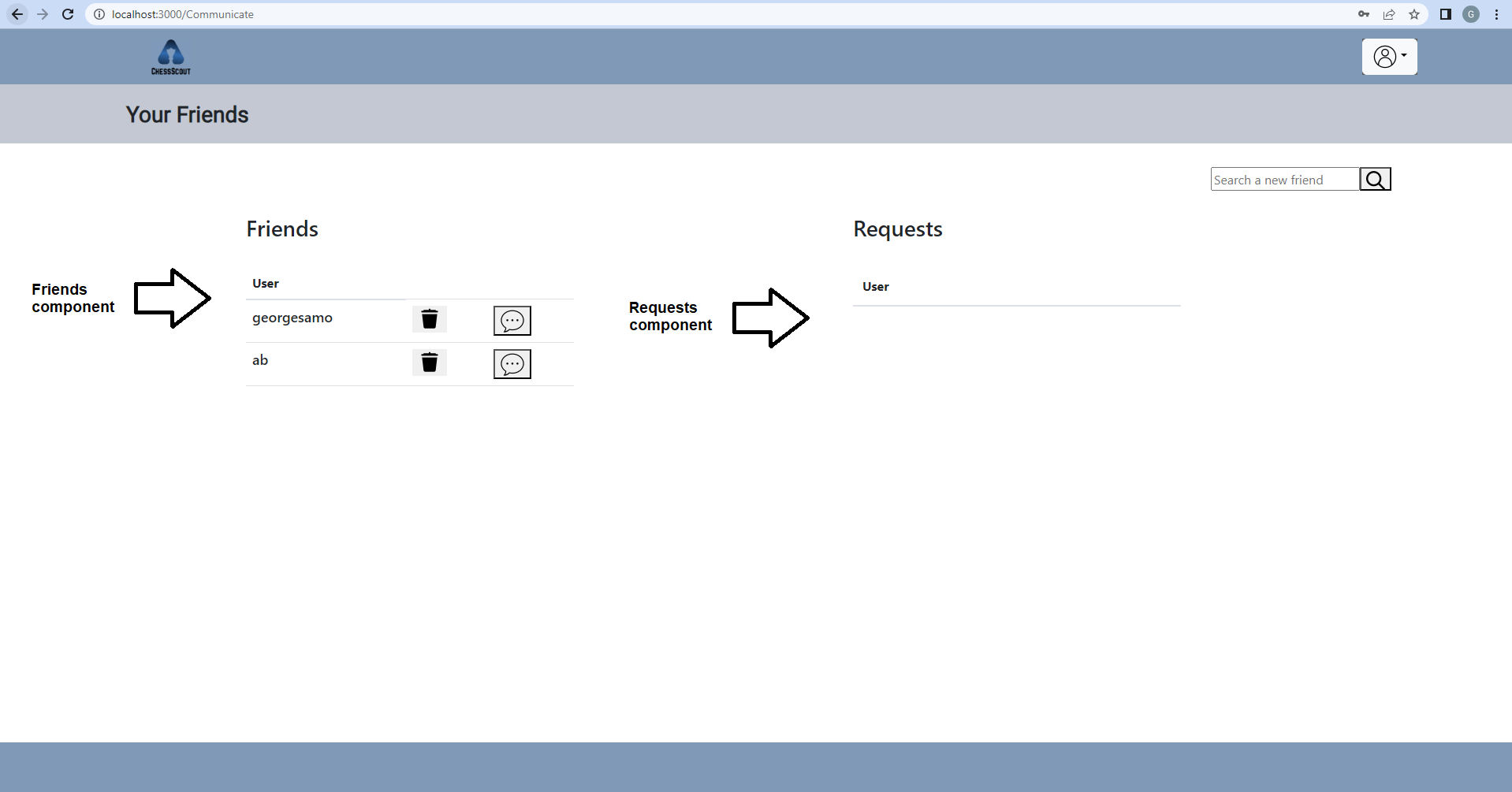
The <OpenGame/> navigates the user to the Open Game page with the Open Game component. The Open Game component includes two sub-components, the chess board and moves list component.

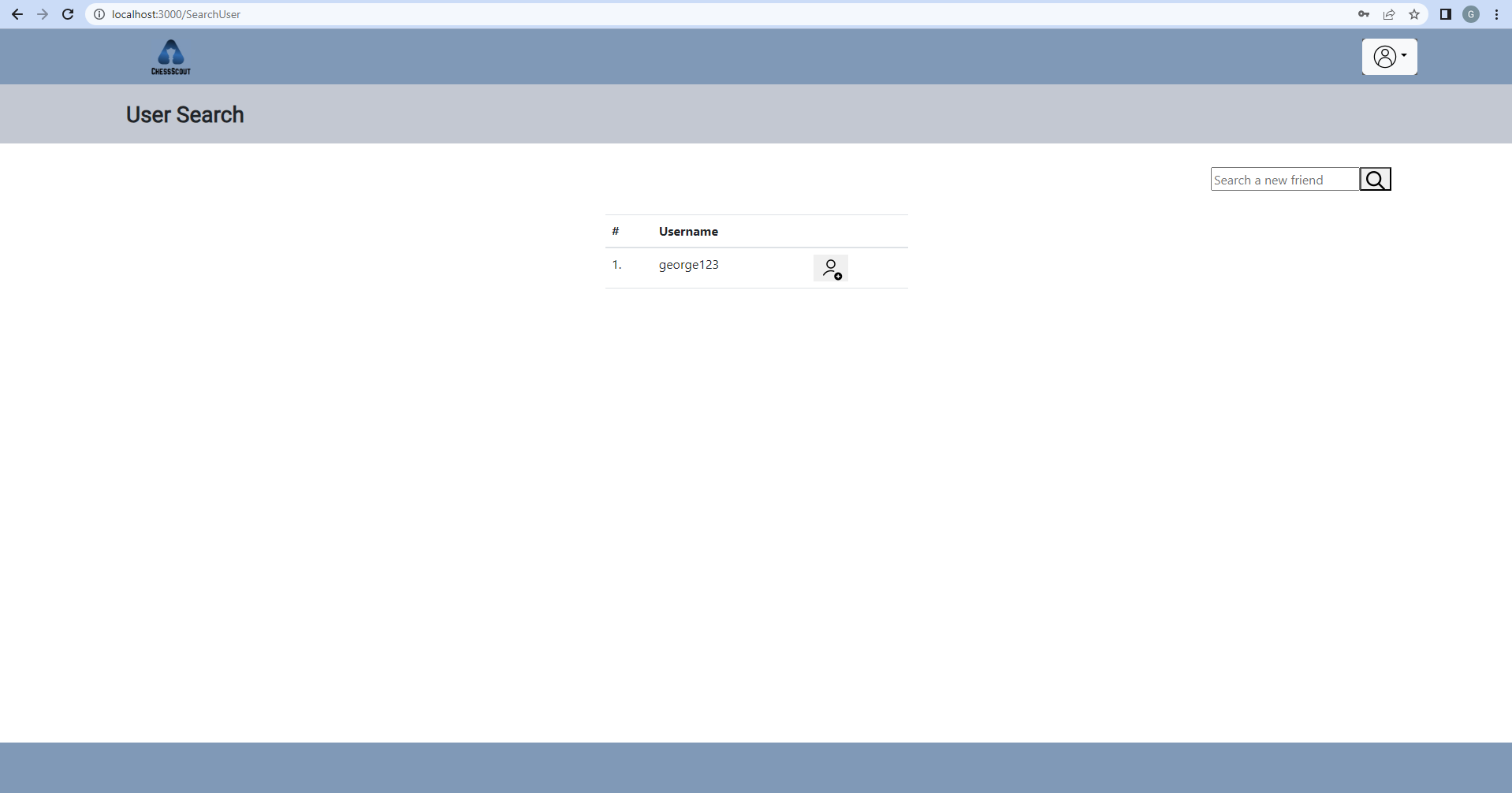
The <CreateScoutProfile/> navigates the user to the Create Scout Profile page with the Create Scout Profile component. The Create Scout Profile component includes Create Scout Profile form.

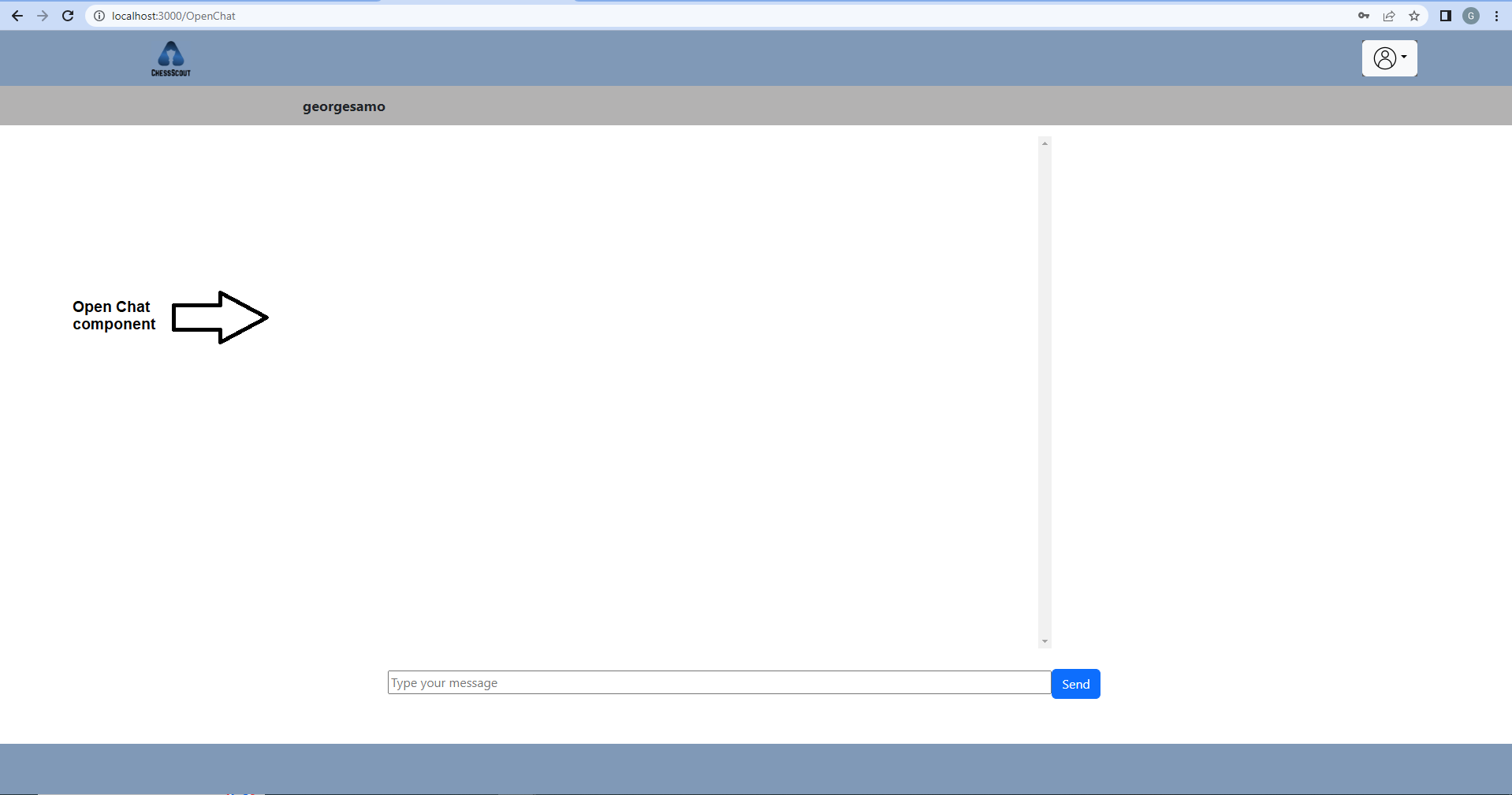


The <ScoutProfiles/> navigates the user to the Scout Profiles page with the Scout Profiles component. The Scout Profiles component includes 5 sub-components. The first one is the player’s info component that contains the player’s name and elo. The second component is the filter component that contains the filter’s form. The other 3 components are the games, the preferred openings and the opening tree.

The <ViewGraphs/> navigates the user to the View Graphs page with the View Graphs component. The View Graphs component includes 2 sub-components. The first one is the opening preference component that contains the player's most preferred openings for the player. The second component is the best result opening component that contains the player's best openings.

The <Communicate/> navigates the user to the Communicate page with the Communicate component. The Communicate component includes 2 sub-components. The first one is the friends component that contains the user’s friends. The second component is the requests component that contains the user’s pending requests.

The <SearchUser/> navigates the user to the Search User page with the Search User component. The Search User component includes the results of the user's friend search (one div for every result).

The <OpenChat/> navigates the user to the Open Chat page with the Open Chat component. The Open Chat component includes the chat with another user (one div tag for every message).

**6.2 Server’s APIs**

System’s backend is developed with the ASP.NET framework. Our backend consists of a number of APIs (controller’s folder) and models (model’s folder). In ASP.NET, models are implemented to represent the data types of our application. In this system the models that we created are :

* **DataOpeningName** (modelizes system’s preferred opening)
* **Friend** (modelizes system user’s friend)
* **FriendRequest** (modelizes system’s friend request)
* **GameDetails** (modelizes system player’s game)
* **Message** (modelizes system’s message between users)
* **MoveTree** (modelizes system opening tree’s move)
* **MoveTreeResponse** (modelizes system’s opening tree)
* **ScoutData** (modelizes system’s scout profile)
* **User** (modelizes system’s user)

Except for the models above, our system’s server contains a number of APIs that the front-end uses (client part). There are 27 different API :

**1) https://local host:7183/api/User/GetFriendRequests**

This API gets the user's username as a query parameter, verifies the user, gets the friend request that is sent to the user from others and returns a json of FriendRequest objects.

**2) https://local host:7183/api/User/GetFriends**

This API gets the user's username as a query parameter, verifies the user, gets the user’s friends and returns a json of Friend objects.

**3) https://local host:7183/api/User/DeleteFriend**

This API gets the users’ usernames (username1 and username2) as query parameters, verifies the users and deletes friends relation from the database’s table.

**4) https://local host:7183/api/User/SearchUsers**

This API gets the user's username and user’s username that we search for as query parameters, verifies the user, gets the users with the same username and returns a json of User objects.

**5) https://local host:7183/api/User/DeleteRequest**

This API gets the users’ uids (SenderUid and ReceiverUid) as query parameters and deletes the request from the database’s table.

**6) https://local host:7183/api/User/SaveGame**

This API gets the user’s username and game’s info as query parameters, verifies the user and inserts the game to the database’s table.

**7) https://local host:7183/api/User/AcceptRequest**

This API gets the users’ uids (SenderUid and ReceiverUid) as query parameters, deletes the request from the database’s table and inserts friend relation to the database’s table.

**8) https://local host:7183/api/User/SignIn**

This API gets the user’s password and username as query parameters, verifies the user and returns a User object.

**9) https://local host:7183/api/User/SignUp**

This API gets the user’s sign up info as query parameters, checks for duplications, inserts the user to the database’s table and returns a success or fail message.

**10) https://local host:7183/api/User/SaveProfile**

This API gets the profile’s info as query parameters, checks for duplications, inserts the profile to the database’s table and returns a success or fail message.

**11) https://local host:7183/api/User/GetProfiles**

This API gets the user’s password and username as query parameters, verifies the user, gets the user’s profiles from the database’s table and returns a json of ScoutData objects.

**12) https://local host:7183/api/User/DeleteProfiles**

This API gets the profile’s pid (profile id) as query parameters and deletes the profile from the database’s table .

**13) https://local host:7183/api/User/GetGames**

This API gets the user’s username as query parameters, verifies the user, gets the user's saved game from the database’s table and returns a json of GameDetails objects.

**14) https://local host:7183/api/User/DeleteGame**

This API gets the game’s gid as query parameters and deletes the game from the database’s table.

**15) https://local host:7183/api/User/GetMessages**

This API gets the users’ usernames (Sender and Receiver) as query parameters, verifies the users, gets all the messages between these 2 users and returns a json of Message objects.

**16) https://local host:7183/api/User/SendMessage**

This API gets the users’ usernames (Sender and Receiver) and user’s message as query parameters, verifies the users and inserts the message to the database’s table.

**17) https://local host:7183/api/User/CreateRequest**

This API gets the users’ usernames (Sender and Receiver) and user’s message as query parameters, verifies the users and inserts the message to the database’s table.

**18) https://local host:7183/api/CreateProfile/OpenProfile**

This API gets the profile’s pid (profile’s unique number) as query parameters and returns the profile with the same pid.

**19) https://local host:7183/CreateProfile**

This API gets the player’s first name , last name and color that plays as query parameters. Then, by using the first and last name crawls all the player games’ info from chessgames.com site (page by page) . After crawling it filters games by color (using GamesByColor function) and returns a json of GameDetails objects.

**20) https://local host:7183/DownloadGame**

This API gets the game’s gid (game’s unique number) as query parameters, crawls the game's PGN from chessgames.com site and returns a json of GameDetail object which is the game with this pid.

**21) https://local host:7183/CalculateByMove**

This API crawls 8 of the player's games (game’s PGN) every time that is called. Then, gets the depth that the player wants from the opening tree and the last move of the tree as query parameters and returns a json of MoveTree objects from the valid games.

**22) https://local host:7183/GetData**

This API returns the current player's name , elo , startYear and lastYear.

**23) https://local host:7183/NewFilters**

This API gets new filters that the user applied (color, last year and start year) as query parameters, finds the valid games and returns a json of GameDetails objects.

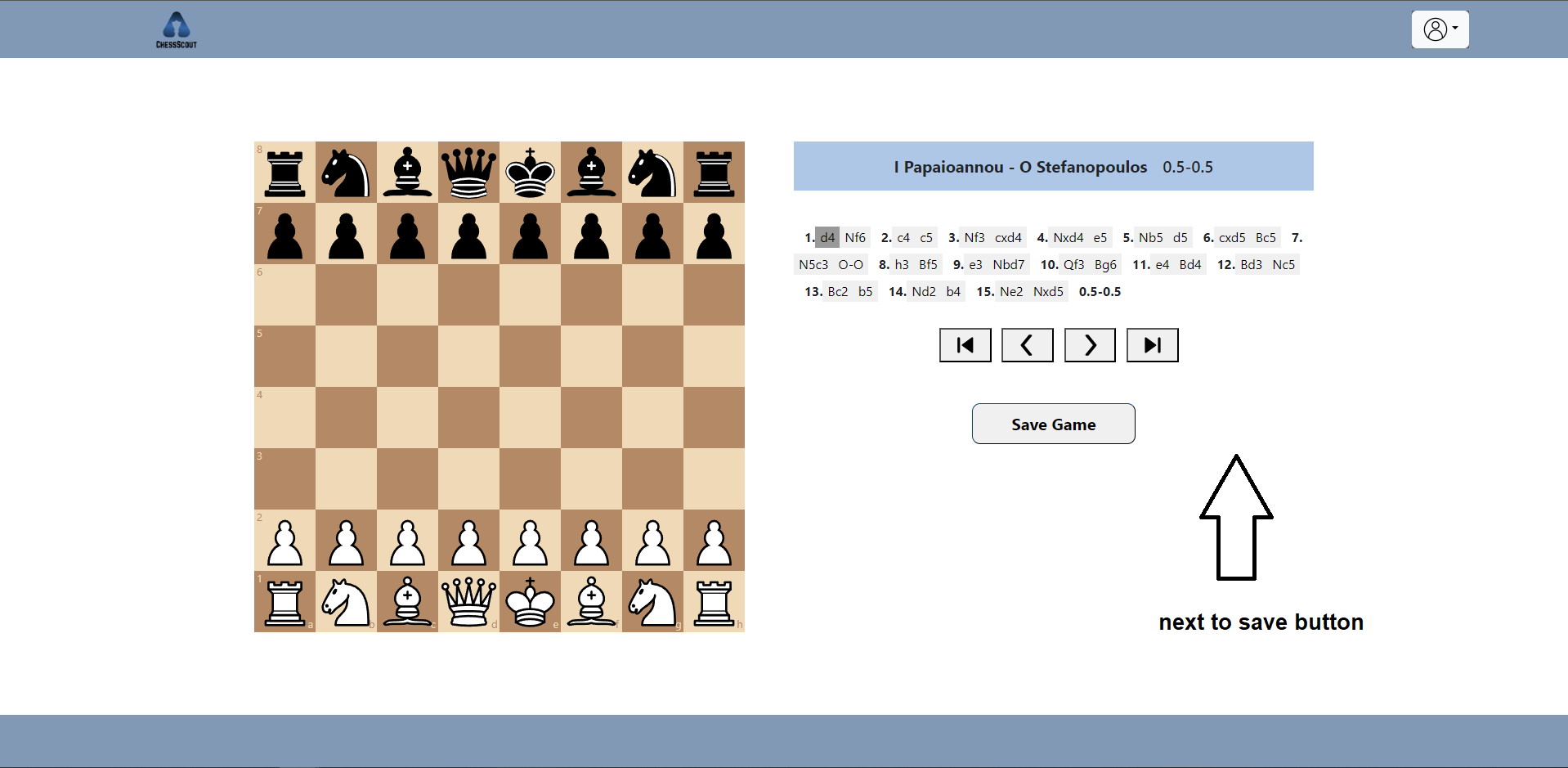
**24) https://local host:7183/CalculateNewFiltersByName**

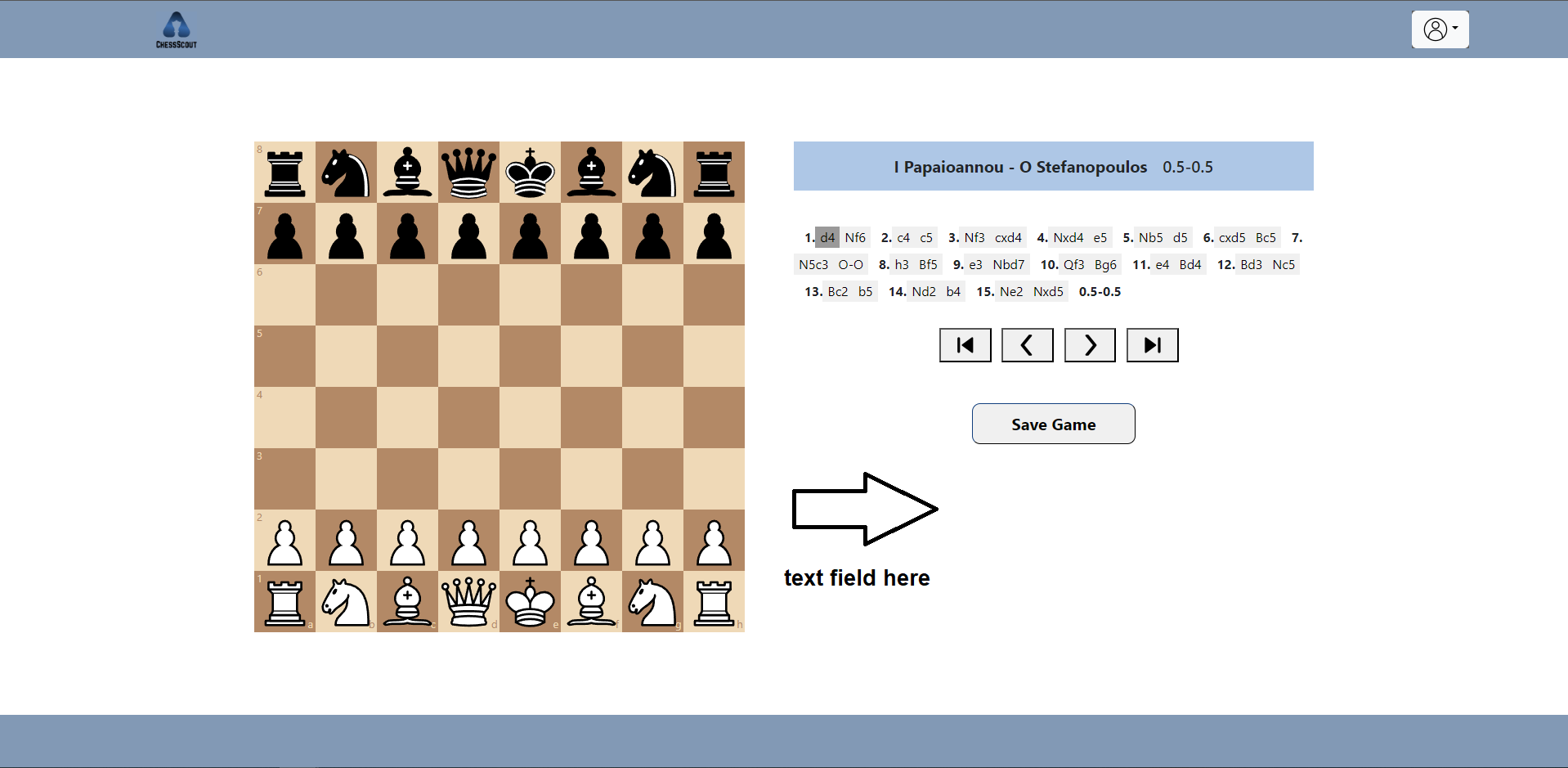
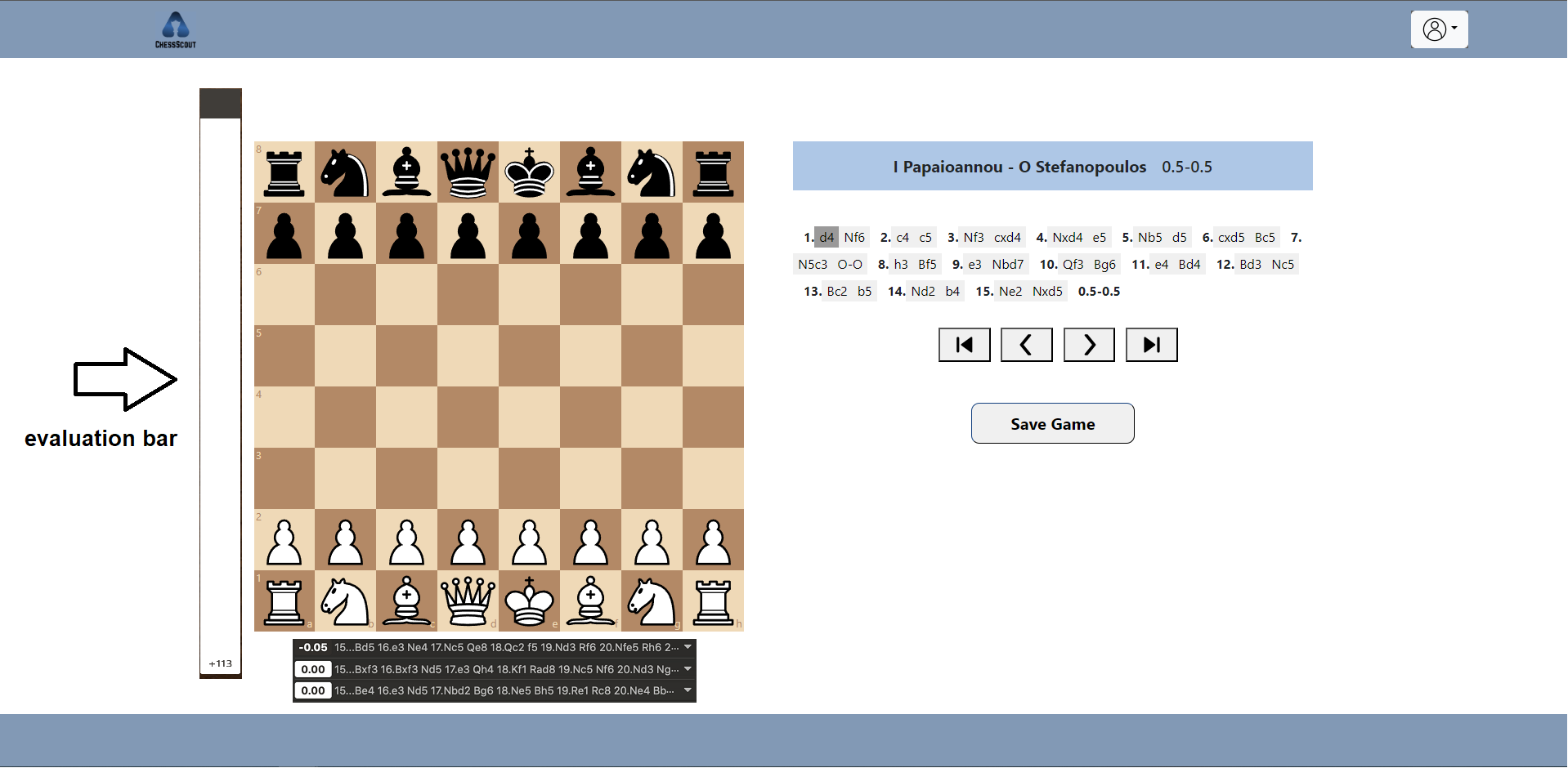
This API calculates the preferred openings data by ECO from the valid games (after filters) and returns a json of DataOpeningName objects.

**Future Extensions**

In terms of the future development of our system, there are several new features for the system’s extension. Here are some of them :

* Users download a game as a PGN file to their computer. PGN, which stands for Portable Game Notation, is a widely used plain-text format for recording chess games. It allows users to store and share games, along with additional information such as game details and annotations. PGN files are compatible with various chess software and databases, making it convenient for analysis and study. For these reasons, users need to have the ability to download the games when they access them.



* Users can add comments to games either before or after downloading them. Users who utilize the app for training, coaching, analysis, and study purposes are encouraged to write their own comments for each game they wish to annotate. This is why we should add a field below the buttons, allowing users to write their own comments.
* Sharing games between users is a vital feature for app upgrades as it facilitates faster learning and improvement, which are two primary goals for every chess player. Collaboration and idea sharing accelerate these processes, making game sharing an essential component of the app's enhancement.
* Adding the ECO (Encyclopaedia of Chess Openings) as a filter in profile creation would greatly benefit users by enabling them to find more specific games and access relevant data within the vast pool of available games.
* Expanding the pool of chess games to include a larger selection from online chess sites such as chess.com or lichess.org would be beneficial. Many professional players participate in online tournaments on these platforms, resulting in a wealth of games that contain vast amounts of valuable information.
* Offers the user the ability to add sub-variations in games before or after they download them. This feature makes the game's analysis easier because the user can modify the PGN with his own variations.
* Creating forums for users to build a community and share information is a valuable addition. Forums enhance communication through discussions between members, provide a platform to share information about FIDE tournaments for participation, improve game analysis, and enable users to provide feedback to the site.
* Integrating a chess engine into game analysis is crucial. Every game analysis should include a chess engine to evaluate and calculate the best moves and variations. This tool is instrumental in helping users create high-level analysis that aids in the improvement of their game.

**Conclusion**

Nowadays, there is a lack of chess scout engines in the chess community, leading to increased interest from companies involved in chess to develop tools that offer game preparation for players. An environment like this gives players a huge amount of organized data that can explore more efficiently in order to advance their ideas and playstyle .The objective of this thesis is to establish a foundation for creating an environment that closely emulates such tools.

**References**

1. <https://legacy.reactjs.org/>
2. <https://getbootstrap.com/>
3. <https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-7.0>
4. <https://www.chessgames.com/>
5. <https://code.visualstudio.com/>
6. <https://visualstudio.microsoft.com/>
7. <https://www.w3schools.com/>
8. <https://stackoverflow.com/>
9. <https://www.365chess.com/eco.php>
10. <https://www.npmjs.com/package/chess.js?utm_source=cdnjs&utm_medium=cdnjs_link&utm_campaign=cdnjs_library>
11. <https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>
12. <https://icons8.com/>