Project report - White Knight Chess

CASPER ANDERSSON, Mälardalen University
NICK GRANNAS, Mälardalen University
PHILIP KARLSSON, Mälardalen University
ALEXANDER ANDERSSON, Mälardalen University
EMIL ELFSTRÖM, Mälardalen University

1 INTRODUCTION

This is the project report for the chess web application developed by the authors for the course "Development of web applications", DVA231, at Mälardalen University. The application is called White Knight Chess, and aims to be a online hub for chess players to play each other, communicate via social features, and have their data handled ethically and securely. We developed the application with the goal of receiving grade five, and were guided by project supervisor Rong Gu and examinator Afshin Ameri. This report details the broad strokes of the course of development and the reasoning behind some important decisions. The authors and major editorial contributors for each paragraph are indicated by the initials at the end of each paragraph. (PK)

2 BACKGROUND

Related websites which inspired this work are chess.com and lichess.org[4, 9]. These sites feature the ability to view completed games and start new games with friends from your friends list. Some of the features these sites contains are: leaderboards, different game modes, highlighting for all possible moves for a selected piece and the possibility of playing against a computer. (EE, PK, AA)

White Knight Chess (referred to as WKC from now on) is a web application for playing chess against friends or other people. One core feature is a secure account system. Users can manage their account by changing password on the profile page, and view a game-archive for completed games. WKC supports a large player base, because the software is scalable. The landing page shows the lobby, containing all users looking for an opponent. While playing (fig. 1), users are informed when they are checked and if it is their turn. WKC have a chat for each game for those players and has a friend system as well. From the search page, users can search for and add other users as friends. There is also a sidebar available throughout the site. It shows information about the user's active games, and shows both online and offline friends, through which new games can be initiated. The website is mobile and tablet friendly, and has different layouts for three different screen sizes. Several frameworks and design patterns were used to build WKC. (EE, PK)

Bootstrap is a popular front-end framework for HTML, CSS, and JavaScript for developing responsive, mobile friendly web pages[3]. Bootstrap takes the mobile-first approach to its design[16]. This means the framework is built to work on smaller devices and less feature-rich browsers first, that might not understand JavaScript or media queries. The framework is then designed to seamlessly adapt to larger devices and feature-rich browsers. (PK, EE)

The ASP.NET Core MVC framework is based of the MVC design pattern[12]. MVC stands for Model-View-Controller, which is a separation of concerns between the View, responsible for producing HTML markdown for the clients; the controller, responsible for handling HTTP requests from the client to the server (routing); and the Model, which contains the business logic and data that is used between the View and Controller. MVC allows
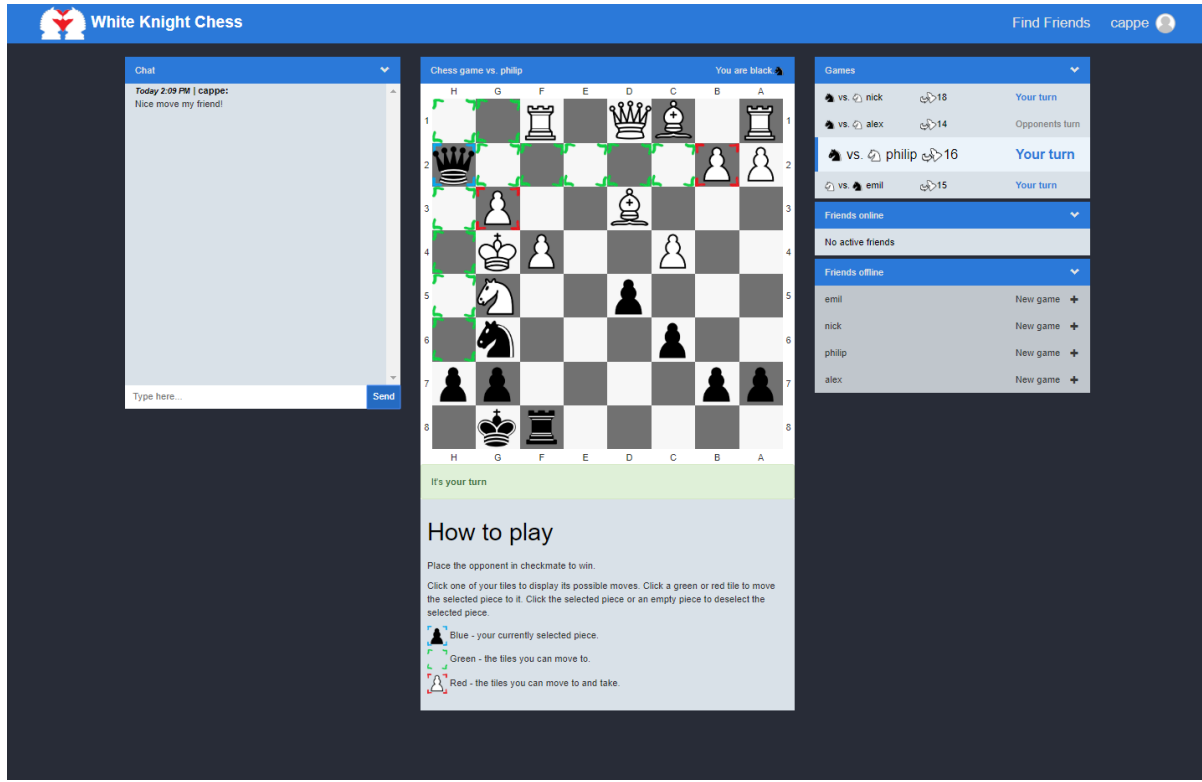
Fig. 1. The game view as seen by the user "cappe" engaged in a chess match against "philip". On the top is the nav-bar. On the left is the chat for the game, and the game itself is shown in the middle. Possible moves for the selected piece (highlighted in blue) are highlighted in green and red. In the sidebar, links to and information about other games "cappe" is engaged in can be seen, as well as his friends.

for hiding the underlying structure of data to the client. The framework aims to provide a clean separation of concerns. (PK, CA)

ASP.NET Identity is an extensible user account system that plugs into any .NET application[1]. It offers a template website for prebuilt secure login and management pages. It comes with a prebuilt login/register system and a "remember me" functionality. This includes hashing the passwords and saving them to a database, and setting a persistent cookie, that expires after two weeks, to remember that you were logged in. (PK, CA, NG)

ASP.NET Identity makes use of Microsoft's ADO.NET Entity Framework (EF Core) by default[17]. EF Core is an Object-Relational-Mapping database (ORM), meaning it automatically translates C# classes into database tables and the other way around when querying data. Using an ORM comes with the cost of understanding the actual structure of the database and schema[19]. On the other hand, C#'s LINQ library automatically protects against SQL injection[6]. LINQ makes it easier to write SQL queries, without a need to convert the received data. (CA, PK)

A Repository is an object or component that acts as an intermediary between the model layer and data layer by centralizing all data access and encapsulating the logic required to access data sources[11]. By encapsulating objects stored in the database together with operations that can be performed on them, the data source is

decoupled from the model. This can result in better maintainability and easier testing, as the repository could easily be changed. (AA, PK)

Dependency injection allows for easy module replacement[5]. By pre-defining the controllers behaviour towards the different modules, using interfaces, any class that implements the interface can be injected. This allows for data hiding, and testing of different implementations or temporary mock-data. (PK, CA)

Data Transfer Objects (DTO) are used to transfer data in a single object. They can limit the amount of calls that is done to the server[15]. Instead of calling the server for each atomic data, the server can collect the information into one DTO and send it to the client. DTO's are therefore convenient when handling large amounts of data or nested structures.(NG, PK)

The observer pattern consists of one subject object and several observer objects which are notified when something changes in the subject[13]. This is commonly used on a higher level of abstraction with a standard library. (PK)

3   CODE STRUCTURE

Our project consists of three main components: Data, which handles the database; Game, for the gamelogic; UI, which handles client requests and contains all web pages.

The Game and Data components are built as standalone projects with their own namespaces. This allowed for easy separate testing. The UI component is the actual web server that references the other two components. Each component contains the models and entities for the data required for its internal operations. We decided the Game and Data component should not reference any other projects, in order to minimize problems when different people worked on different code. The UI however, uses data models from Game and Data.

We injected both the gamelogic and the database repository as dependencies into the controllers which needed to use them (fig. 2). UserManager is part of the ASP.NET Identity System and is also injected. It provides methods for working with users, for example, getting the user ID for the client sending a request. (CA, PK)

4   METHOD

4.1   Setting up the project

We first had a design meeting, discussing how the end result should look and feel. After reviewing related websites we formulated a basic design for the layout of the necessary visual artifacts. We chose to use the ASP.NET MVC framework because we had used it before. We aimed to build a strong foundation to later expand upon. In that way we can focus on core features our users wants. After the general course was set, we split the project into three major components. Game, data, and user interface, as described in the previous section. We agreed to use Git as version control and proceeded to set up a Git repository for our project. We then made a weekly schedule so that each group member would partake in at least 16 hours of development sessions weekly. We decided who would be responsible for which part of the project. (PK, NG, CA)

Nick and Emil were assigned to work on the user interface. Nick has previously redesigned the KronoX app, and Emil has experience with web design from previous education[8]. Alexander opted for the data component, since he has used EF Core in the laborations of this course. Casper was a good fit for the game component, since he is productive independently. Philip became the project manager, because he has had similar roles previously. We wanted to have a project manager in order to circumvent the tardiness problems of direct democracy that we experienced earlier. It became Philip's responsibility to review the code that is pushed to the master branch, and to make sure that we followed the project plan and followed good practice. (PK, NG)

Initially, we worked in one branch for each of the three groups and made pull requests to the master branch on Github. After the first week, we switched to the Github Flow workflow[7]. This allowed us to work on feature branches, which after review can be merged to master with the newly implemented feature.(NG, PK)
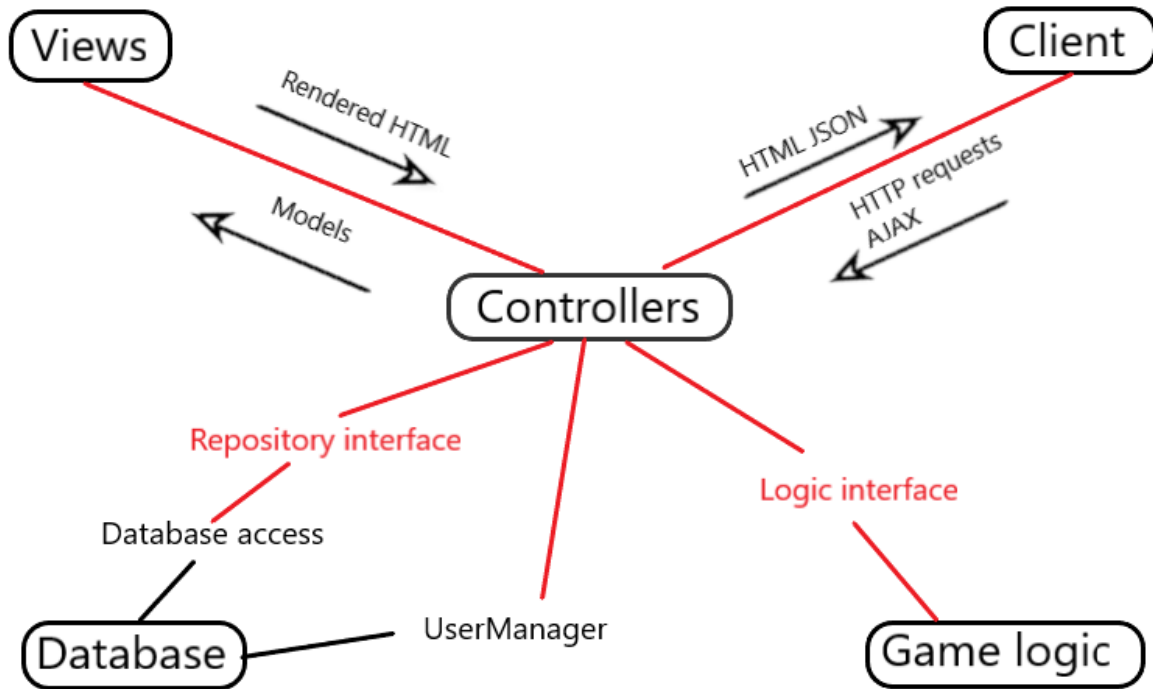
Fig. 2. Illustration of the project layout and interaction between different components. The MVC pattern is present in the form of the Controller processing requests and passing Models to the Views, which are then sent to the client.

For our database, we used Microsoft SQL together with EF Core because of its ease of use and our previous experience with it. EF Core facilitate the code first approach to generate the database schema. This reduced our workload to set up a database by abstracting the database design. This choice to use EF Core removed the necessity to manually write SQL queries which is both error prone and vulnerable to SQL injection. We could instead use LINQ to write strongly typed queries. We used the repository pattern in our project to abstract and encapsulate the database and make the data accessible through an interface. It also gives the project better maintainability and helps achieve a higher state of modularity and low coupling. Using a repository makes it easy to change database provider. For example, Casper worked on a Linux computer and used SQLite. (AA, PK)

4.2 Implementing features

We then started implementing the lobby. When a user requests to join the lobby a method is called which changes the user's lobby status to "Registered" in the database. To leave the lobby another method is called that changes the status again. A user will also be automatically removed from the lobby when another user starts a game from the lobby with them. The lobby is regularly refreshed with AJAX so that changes in the lobby is visible to the user.(NG, EE, PK)

The board is first rendered when a specific game session is requested by the client. Razor code, which is C# code compiled on the server but written inline with HTML markdown, is used to draw the board state of the game session[2]. Information about the user is also accessed by the Razor code in order to draw the board from

each players point of view. This is done by generating 64 css-grid styled div-tags which display images for the pieces. To make the board responsive, we implemented a naive version of the observer pattern in AJAX[14]. Each web-client is an observer, waiting for the state of the board to change. This is more efficient than constantly updating the board manually, regardless if it is changed or not. An event loop (implemented as a timed javascript function) makes a request to the event provider (a POST request to the back-end)[18]. If changes have been made, another javascript function is called to "dispatch the event", by updating the affected tiles' div-tags. In a non distributed application an event loop might not have a timer, but on a web application it conserves bandwidth. (PK)

We built a chess engine to separate the game logic into its own component that is ignorant of the rest of the application. Therefore it could well be used for a non-web related project. It needs to know the state of the board, who's turn it is, and which tile is selected or what move the player wants to do. This is provided by DTO parameters. The game component can...

- ...decide if a move is valid.
- ...perform a move and return the new state of the board.
- ...decide if a player is checked or in checkmate.
- ...provide a list of valid moves for a piece.

We implemented functions to get the basic valid moves for every type of chess piece, relative to its current position. Using this we could get all the opponent's valid moves. If the king is in one those positions the player is checked. To see if the player is in checkmate every possible move for the player is evaluated to decide if the player is still checked. If no move can remove the king from being checked the player is in checkmate and the game is over. (CA, PK)

After our second meeting with Rong, we were informed that we required to implement a chat-system for grade five. First we researched a chat system with a hub connection that was part of the ASP.NET SignalR library[10]. However, we would not have time to learn it and adapt it to the project. We instead implemented a new chat system with AJAX, which we knew how to do. In broad terms, the chat messages would be stored in a relation with a foreign key to a particular gamesession, which meant that a chat message and a gamesession have a many-to-one relationship. The message and game ID is sent with a POST request to a controller, which adds a timestamp and a username to the object and saves it to the database. Our javascript periodically polls the server if the chat has been updated with a POST request. If it has the partial view of the chat is reloaded asynchronously to suit our vision of a fluency and responsiveness. Without AJAX the user would have to refresh the page every time they wanted to see if the chat is updated. That is why this feature was deemed more important than features mentioned in the next session. (NG, CA, PK)

We realized the users might expect to be able to access their games after they are finished Since they will no longer appear in the sidebar of active games, therefore we implemented an archive. The archive uses a partial view of the game board to display the state that the game was in when it ended. This reuses much of the code used to draw the board. The repository also provides a list of finished games for the signed in user of which one can be shown at a time. (NG, PK)

4.3 User study

To evaluate our website a user study was performed by two anonymous participants. All participants were informed beforehand of their rights to withdraw consent, and that the results would be used for a project report. The participants were asked to perform tasks that test the core functionality and features of our website. Then the participants were asked about their experience. These questions allowed us to receive feedback on both the functionality and the design of our website, and highlight areas in need of improvement.

These are the tasks that they were asked to complete:

(1) Create a new account.
(2) Start a new game from the lobby page.
(3) Make the first move in the game.
(4) Add a user as a friend.
(5) Start a new game with that user
(6) Log out.

They were then asked these questions:

(1) Did you have any problems performing the tasks? Was it unclear how to execute any of the tasks?
(2) Do you have any other feedback or opinions of the website?
(3) If there is one thing you could change about the website, what would it be?

No user found the tasks difficult to complete. They found the website was aesthetically pleasing. One user asked for a help button to display what certain things do and how they work. One user experienced that it was odd to see an empty chat when entering a new game. (EE, NG, PK)

5 DISCUSSION

During the user study no user had any problems navigating the website. The information provided on each page seemed to be sufficient as all the users quickly understood the core functionality of the features on each page. One user said that they wanted a help-button that provides useful information or tips when clicked. This tells us that the user experience might be improved by giving first time users access to more information if they want it. The chat window on the game page starts out as an empty window. An improvement might be to make the window smaller at first, only expanding with new messages. (EE, PK)

WKC can be developed further by improving existing features, as well as adding new features. Possible new features include customizable appearance, better notifications, move history, a leaderboard, extended user statuses, persistent friend-to-friend chats and more efficient game logic. (EE, PK)

During our presentation we received feedback from Afshin that users might want more noticable indications of when a message is received or a game status is updated by way of a notification. The sidebar of the website currently shows the status of every game that a user is playing. When the opponent makes a move the sidebar dynamically updates to show that it is your turn. The upgraded system could show a notification as well, and prompt users to accept or decline friend- and game requests. This would improve the user experience by making relevant information more accessible, in a manner that most users will be familiar with from their experience with smartphone operating systems. (EE, NG, PK)

Another extension of the archive would be a history of every move in the game. This would also show the state of the game for each move. The move history would be displayed next to the game board so a user can see the history of the game, both in the archive view and while playing an ongoing game. This implementation would help players master chess by going back to analyze their moves. (EE, NG, PK)

Rong suggested during a meeting that an improvement of the chat, in general the dynamic parts, could be made by using a publish-subscribe pattern. This can be achieved with ASP.NET SignalR library and allow us to remove the AJAX that runs at timed intervals, and replace it with publishing when a change is made on the database[10]. The client would not have to rely on an interval to wait for the next fetch attempt, and there would not be constant requests to the server. (PK, NG)

Other than "online" and "offline, there could be another status for when a user has been inactive for some time, but are still logged in. This status could be displayed as "away", and would indicate if a friend is available. (EE, PK)

The current game logic uses a naive and inefficient method for finding checkmate. There are duplicate checks done. These unnecessary calculations use processing power that can be used for other computations. This could become a severe problem if there are many users. Also, if a scalable hosting service is used for the server, this

could lead to higher operating costs. Instead of attempting every move for each piece, we can use a process of elimination. By starting with a collection of every tile the enemies that are checking the king are standing on and can move to, removing every tile that did not solve the problem; thus reducing the amount of computation. Instead of creating a copy of the original board every time, it would be better to use one board to do a move and then backtrack it after each attempted move. (CA, PK)

REFERENCES

[1] Rick Anderson et al. 2019. Introduction to ASP.NET Identity. Retrieved November 8, 2019 from https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity

[2] Rick Anderson et al. 2019. *Razor syntax reference for ASP.NET Core*. Retrieved November 8, 2019 from https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-2.0

[3] Bootstrap. [n.d.]. Bootstrap. Retrieved November 8, 2019 from https://getbootstrap.com/docs/3.3/

[4] Chess.com. [n.d.]. Chess.com - Play chess online. Retrieved November 8, 2019 from https://www.chess.com/

[5] Android Developers. [n.d.]. Dependency injection in Android. Retrieved November 8, 2019 from https://developer.android.com/training/dependency-injection

[6] Entity Framework. [n.d.]. LINQ Prevent SQL Injection. Retrieved November 8, 2019 from https://entityframework.net/linq-prevent-sql-injection

[7] Github. 2017. Understanding the GitHub flow. Retrieved November 8, 2019 from https://guides.github.com/introduction/flow/

[8] Nick Grannas. 2019. Utvärdering av en enklare design med ett neutralt perspektiv. (April 2019). Improvements on existing UX assignment.

[9] lichess.org. [n.d.]. Lichess.com - Free online chess. Retrieved November 8, 2019 from https://lichess.org/

[10] Microsoft. [n.d.]. Real-time ASP.NET with SignalR. Retrieved November 8, 2019 from https://dotnet.microsoft.com/apps/aspnet/signalr

[11] John Parente and Maira Wenzel. 2018. Design the infrastructure persistence layer. Retrieved November 8, 2019 from https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design

[12] Steve Smith. 2019. *Overview of ASP.NET Core MVC*. Retrieved November 8, 2019 from https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.0

[13] Tutorialspoint. [n.d.]. Design Patterns - Observer Pattern. Retrieved November 8, 2019 from https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

[14] w3schools. [n.d.]. PHP - AJAX Introduction. Retrieved November 8, 2019 from https://www.w3schools.com/php/php_ajax_intro.asp

[15] Mike Wasson. 2014. *Create Data Transfer Objects (DTOs)*. Retrieved November 8, 2019 from https://docs.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5

[16] Wikipedia. [n.d.]. Responsive web design. Retrieved November 8, 2019 from https://en.wikipedia.org/wiki/Responsive_web_design#Mobile_first,_unobtrusive_JavaScript,_and_progressive_enhancement

[17] Wikipedia. 2019. ADO.NET. Retrieved November 8, 2019 from https://en.wikipedia.org/wiki/ADO.NET

[18] Wikipedia. 2019. Event loop. Retrieved November 8, 2019 from https://en.wikipedia.org/wiki/Event_loop

[19] Wikipedia. 2019. Object-Relational mapping. Retrieved November 8, 2019 from https://en.wikipedia.org/wiki/Object-relational_mapping