



ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

по професия код 481020 „Системен програмист“
специалност код 4810201 „Системно програмиране“

Тема: Система за провеждане на игра на шах
между човек и алгоритъм

Дипломант:
Кристиян Михайлов Петров

Дипломен ръководител:
Николай Станишев

СОФИЯ

2022



ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Дата на заданието: 14.12.2021 г.

Утвърждавам:.....

Дата на предаване: 14.03.2022 г.

/проф. д-р инж. Т. Василева/

ЗАДАНИЕ за дипломна работа

ДЪРЖАВЕН ИЗПИТ ЗА ПРИДОБИВАНЕ НА ТРЕТА СТЕПЕН НА ПРОФЕСИОНАЛНА КВАЛИФИКАЦИЯ

по професия код 481020 „Системен програмист“

специалност код 4810201 „Системно програмиране“

на ученика Кристиан Михайлов Петров от 12 А клас

1. Тема: Система за провеждане на игра на шах между човек и алгоритъм

2. Изисквания:

2.1 Обучаване на системата, как се играе шах с неговите правила и движения.

2.2 Създаване на алгоритъм, който намира най-оптималния ход в конкретната ситуация, като знае, че ходовете му са през един.

2.3 Създаване на учеб апликация, която комуникира с алгоритъма и визуално представя играта на шах.

3. Съдържание 3.1 Теоретична част
 3.2 Практическа част
 3.3 Приложение

Дипломант:.....

/ Кристиан Петров /

Ръководител:.....

/ Николай Станишев /

Директор:.....

/ доц. д-р инж. Ст. Стефанова /



Отзив на дипломния ръководител

Настоящата дипломна работа отговаря на всички изисквания без едно. Това изискване е заменено в хода на разработката с такова, което ще допринесе по-голяма стойност на проекта.

Темата и технологиите са нови за дипломанта и той е успял да навлезе в дълбочина в тях.

По мое мнение дипломната работа може да бъде допусната до защита.

Увод

Ежедневно напредва развитието в областта на информационните и комуникационни технологии. Благодарение на него популяризирането и употребата на Интернет пространството се увеличава.

В края на 2015г. в световен мащаб активните потребители на Интернет са около 3 млрд., докато през 2021г. тази бройка расте до близо 5 млрд.[1] Хората са си изградили ежедневен навик, който представлява използване на десетки сайтове за добиване на информация, без да се замислят за процеса, който стои зад всичко това. Много от потребителите не осъзнават какво всъщност достъпват и по какво се различава от останалите. WEB сайтът не позволява възможност за интеракция от страна на своите потребители, тоест той просто им предоставя информация, картички, видео и аудио, те са статични и не се обновяват динамично. WEB приложенията са изключително функционални и интерактивни, в повечето случаи очакват някакво действие от потребителя и обновяват своята информация динамично, като доста често са свързани с база данни, където съхраняват необходимите за тях неща. [2]

С раждането на толкова много възможности за създаването на креативни идеи и употребата им от други потребители, се появяват и игрите. През 2021г. бройката на хората, които играят видео игри е около 3 млрд. [3] Това е една много популярна и бързо нарастваща област, която намира все повече последователи от всички възрасти.

С настоящата дипломна работа ще бъде изградена система за WEB базирано приложение за игра на шах. За реализацията на тази идея, тя може да бъде разделена на 3 основни части.

Първата част представлява графичния потребителски интерфейс, който визуално представя на потребителя шахматна дъска и фигурите. Играчът има възможността да мести пionките си с натискането на бутон на мишката. WEB

приложението съдържа няколко страници. Начална, от която е възможно да се достъпят другите две. Страница за влизане в стая, в която трябва се вписва даден код, за присъединяване към чужда игра. Третата страница е самата игра на шах, в която се играе играта, дава възможност за копиране кода на стаята с натискането на бутон или обновяване на дъската с начално наредени фигури.

Втората част е създаването на алгоритми, които проверяват изиграните от играча ходове, вярността и възможността им спрямо правилата на играта шах. Обновяване на информацията за шахматната дъска, която бива променена от потребителския интерфейс или базата данни, чрез изпращане на заявки, към създаденият API, за запаметяване или взимане на данни.

Третата част е свързването на графичния потребителски интерфейс, WEB контролер и базата данни. Създаването на API, който позволява манипулирането на данни свързани с играта на всеки играч. Поддържането на различни адреси, от които могат да се достъпят различните страници на WEB приложението. Отваряне на WebSocket канал, чрез който играчите в дадена игра да сигнализират помежду си за извършения от тях ход.

Проблемът, който решава системата е улеснена и приятна за използване платформа за възпроизвеждане на игра на шах между двама опоненти, която не изисква никаква автентикация.

Използвани термини

WEB - The World Wide Web – Световната мрежа

API - Application Programming Interface – Интерфейс за програмиране на приложения

SQL - Structured Query Language – Език за съхранение, манипулация и получаване на данни от таблици

NOSQL - Non-Structured Query Language – Нерелационен и неструктурриран език за обработка на данни

IDE - Integrated development environment – Среда за разработване на приложения

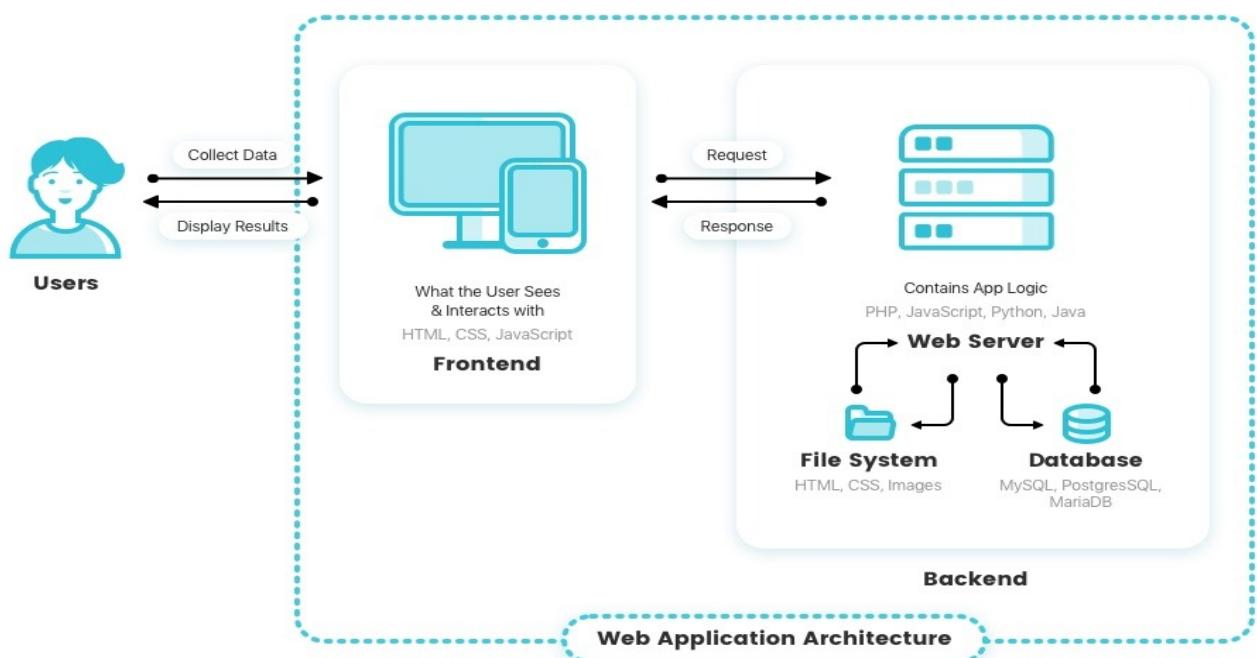
ПЪРВА ГЛАВА

МЕТОДИ И ТЕХНОЛОГИИ ЗА РЕАЛИЗИРАНЕ НА WEB БАЗИРАНО ПРИЛОЖЕНИЕ ЗА ИГРА НА ШАХ

1.1 Основни принципи, технологии и развойни среди за разработване на WEB приложение

1.1.1 Обзор на WEB приложение

За създаването на едно WEB приложение са необходими контролер и графичен потребителски интерфейс(ГПИ). В най-честия случай, потребителят изпраща заявка към контролера, през ГПИ, която се обработва и извлича нужната информация от базата данни, след което данните се изпращат обратно и се изобразяват пред потребителя. Доста често е необходимо да се въведе и WebSocket връзка, която представлява отваряне на канал, чрез който сървърът комуникира с един или няколко клиента едновременно.



Фиг. 1.1 Архитектура на WEB приложение

1.1.2 Принципи за разработване на WEB приложение

Разработката на проект е дълъг и сложен, изиска доста стъпки, за да се получи качествен продукт. [4]

- **Създаване на идея** - Идеята трябва да бъде породена от проблем, който ще бъде решен от продукта.
- **Съществуващи системи** - Трябва да бъде уникално и различимо от другите решения.
- **Технологии** - Избор на релевантни технологии за създаване на WEB приложение, съобразно идеята и тяхната поддръжка на библиотеки и и използваемост.
- **Планиране на основата** - Планирането на основните функционалности на проекта е важно, за да се създаде приоритизация.
- **Менажиране** - Използване на среда за менажиране на апликацията, в която се следи прогрес, приоритетни задачи и т.н
- **Скициране на дизайн** - Важно е да съществува ясна представа за това как ще изглежда приложението, за да бъде разработено лесно след това.
- **База данни** - В повечето случаи необходима, за запазване на данни, които се използват за визуализация или съхраняване на важна информация използвана от алгоритми.
- **Контролер** - Той най-често съдържа базата данни, сървъра, логиката и методи за обработка на заявки.
- **ГПИ** - Създаване на графичен потребителски интерфейс, на база скиците, изчистен, лесен за употреба и интерактивен. По възможност поддържащ мобилни устройства.
- **API-ГПИ** - Необходима връзка, чрез която се визуализират необходимите от потребителя данни, чрез заявки към базата данни.

- **Интернет** - За да бъде използван проекта от други хора, трябва да бъде качен в интернет пространството, като бива пуснат на машина поддържаща го работещ.

1.1.3 Технологии за разработване на WEB приложение

Избирането на технологиите за разработка на WEB приложение е важен процес, който трябва да бъде съобразен с идеята и скалируемостта на проекта. Те могат да бъдат разделени в няколко подкатегории:

- **Технологии за създаване на ГПИ** - Това са тези технологии, които са използвани за направата на красив и удобен графичен потребителски интерфейс, чрез който потребителите да управляват приложението. Такива са “HTML” (Език за създаване на структурата на една страница), “CSS” (Език за украсяване на визуално представените елементи в страница), “JavaScript” (Език използван за добавяне на допълнителна функционалност на ГПИ).
- **Технологии за създаване на контролер** - Това са програмни езици, които позволяват създаването на сървър, приемане на заявки, управление на базата данни и поддръжката на библиотека за разширяване на функционалността. Често използвани програмни езици са “Python”, “Java”, “Javascript”, “PHP” и т.н.
- **Технологии за създаването на база данни** - Базата данни е място за съхранение на важна, ключова информация. Съществуват две основни разновидности “SQL” и “NOSQL”. Важните разлики са, че SQL базите данни са релационни и структурирани, използващи таблици за запазване на информация. При NOSQL не се използват таблици, а ключови стойности, графи и т.н. Те нямат определена структура и не са релационни. Най-разпространени за SQL са “Db2”,

“MySQL”, “PostgreSQL”, “SQLite”, “Microsoft SQL Server” и т.н., а за NOSQL са “Redis”, “FaunaDB”, “MongoDB”, “Cassandra” и други.

- **Framework** - Това е вече изготвена основа за изграждане на WEB приложение. Употребата им е широко разпространена, защото улесняват процеса, понеже не се изготвя апликацията от нищото, ами вече има изградена структура. Съществува голяма разновидност, според избрания програмен език. Примери са “Django”, “Express.js”, “Vue.js”, “Laravel”, “Angular”, “Flask” и т.н.

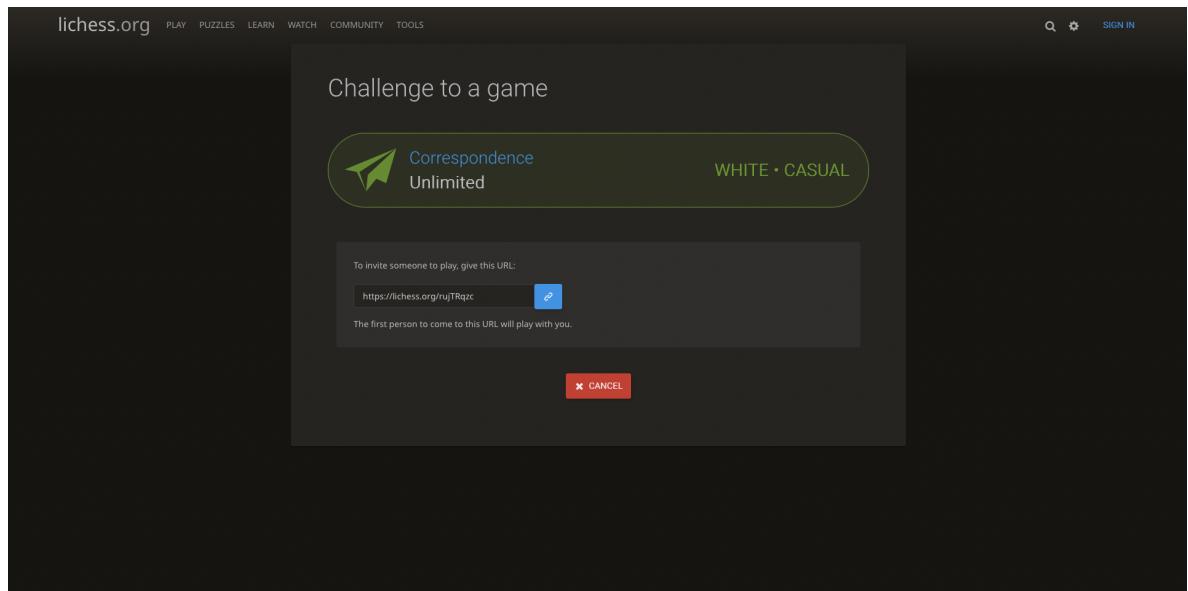
1.1.4 Развойни среди за разработване на WEB приложение

За разработването на WEB приложение са необходими помощни апликации, като например IDE. Това е апликация, която събира няколко инструмента на едно място. Един от тях е текстов редактор, в повечето случаи поддържа различни функционалности, като например подчертаване на синтактични грешки, автоматично допълване на думата и т.н. Друг инструмент, който намира място в IDE е Debugger, то представлява програма за тестване на кода, като проследява всеки един от зададените му параметри, в хода на действие. Примери за популярни IDE са “Visual Studio”, “IntelliJ IDEA”, “PyCharm”, “Eclipse”, “Komodo IDE”.

1.2 Съществуващи решения и реализации

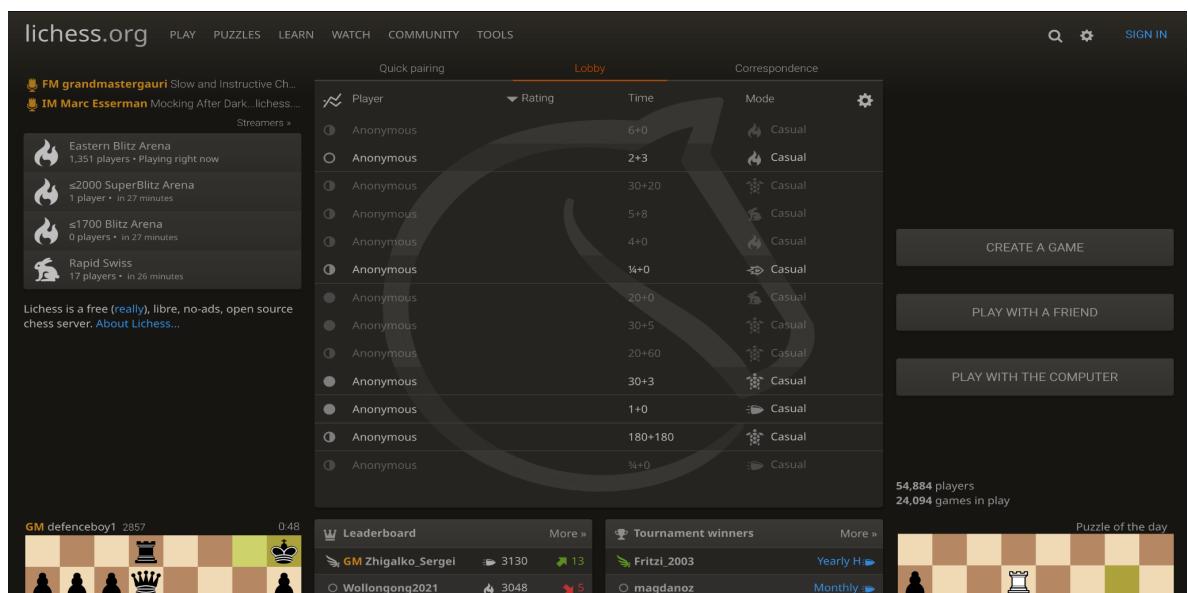
Поради старинния си произход, играта шах е една от най-играните игри по цял свят. Не съществува метод за проследяване на точната бройка играещи играта, но според проучване тя е приблизително 600 мил. [5] Със своята популярност и навлизането в технологичните времена се създават и много компютърни версии на играта. Едни от тях са “Chess.com”, “Lichess”, “Play Magnus”, “Chess24” и т.н

- Lichess е един от най-популярните сайтове за игра на шах. В него се намират много функционалности, като игра с приятели, срещу робот, уроци по шах, турнири и т.н. Една от отличаващите черти е, че позволява създаването на игри между потребители без създаването на профил, а през прашане на персонален адрес. [6]



Фиг. 1.2 Създаване на игра между приятели в Lichess

Според много от потребителите на сайта, ГПИ е неприятен за използване, заглавната страница притежава ненужно много информация и не изглежда професионално направено приложението.



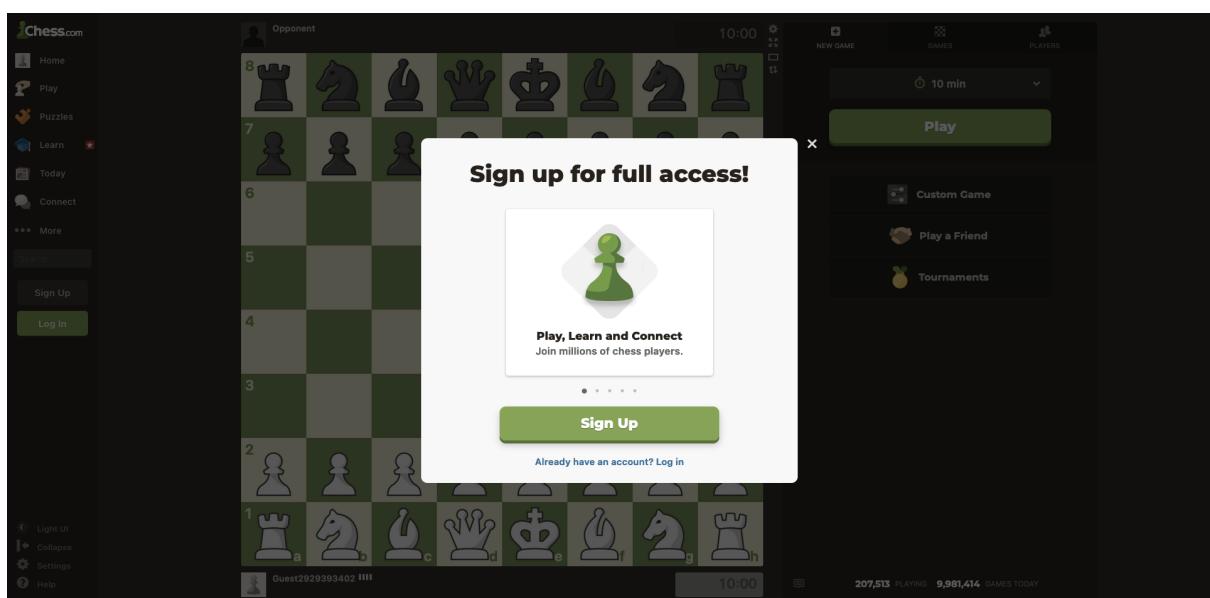
Фиг. 1.3 Заглавна страница на сайта Lichess

- Chess.com според своите потребители е доста приятен за употреба сайт, с много функционалности, разновидни игри на шах, уроци и приятни статии за четене. Спрямо други приложения предоставя лесен и удобен ГПИ.



Фиг. 1.4 Заглавна страница на сайта *Chess.com*

Един от недостатъците на сайта е, че не позволява създаването на игри между приятели, без създаването на профил, както и на много други функции.



Фиг. 1.5 Липса на достъп в *Chess.com*

1.2.1 Алгоритъм за описание на достигната позиция по време на игра на шах (FEN)

Нотацията на Форсайт-Едуардс или FEN е стандартна нотация за описание на позициите на фигуите по време на игра на шах. [8]

FEN е важен и необходим на едно приложение за игра на шах, защото улеснява пренасянето на състоянието на дъската, като го превръща в един ред. Използва се при запаметяване и вземане на информация, като използва 8 части отделени със символа “/”, които представляват един ред от дъската. Всяка една част съдържа съкратена версия на наименованията на фигуите и празните полета на този ред. Буквите, които използва репрезентират фигуите на дъската:

- “p” е пешка
- “r” е топ
- “n” е кон
- “b” е офицер
- “q” е царица
- “k” е цар

Разграничаването на това дали дадена фигура е черна или бяла става с използване на главната буква на дадената пиона, това показва, че тя е от бял цвят. Празните полета на реда се представят с число от 1 до 8, в зависимост от това колко последователни празни полета има.



Фиг. 1.6 Подредбата е:

“rnbqkbnr/pppppppp/8/8/8/
8/PPPPPPPP/RNBQKBNR”

								r1b1k1nr/
								p2p1pNp/
								n2B4/
								1p1NP2P/
								6P1/
								3P1Q2/
								P1P1K3/
								q5b1

Фиг. 1.7 Подредбата е:

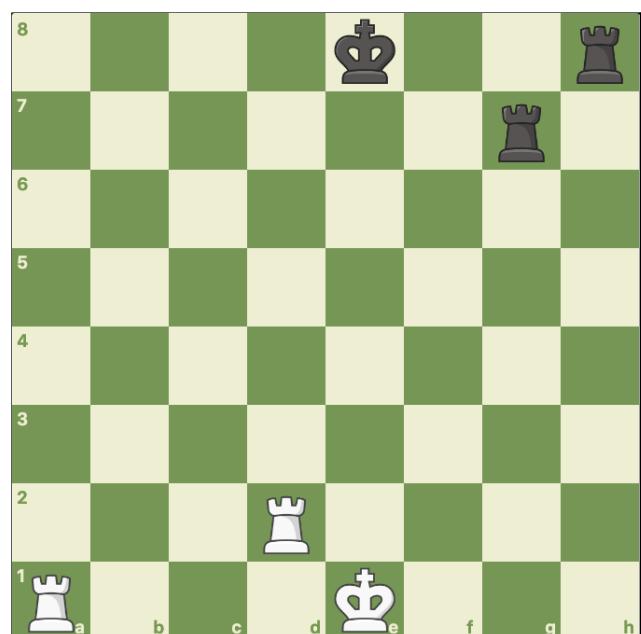
“r1b1k1nr/p2p1pNp/n2B4/1p1N
P2P/6P1/3P1Q2/P1P1K3/q5b1”

Нотацията също показва кой цвят е на ход. След подредбата на фигурите се изписва буквата “w”, която показва ще белите фигури трябва да местят или “b” аналогично за черните фигури.

Възможността за рокада също се включва. Буквата “q” се използва за рокада от страната на царицата, а буквата “k” показва, че е възможна рокада от страната на царя. Когато буквите са големи, рокадата са отнася за белите фигури, а малките букви, за черните. При използване на символа “-” се показва, че няма възможна рокада за нито една от страните.

Фиг. 1.8

FEN: “4k2r/6r1/8/8/8/8/3R4/R3K3 w Qk”



1.2.2 Алгоритъм за записване на всички изиграни ходове в една шахматна игра (PGN)

Нотацията за преносима игра е стандартен формат за записване на игра под формата на текст. Тя е създадена от Стивън Едуардс и подпомага зареждането на шахматната игра от компютри. [9]

Тя е по-различна от **FEN** (1.2.1), тъй като тя записва поредицата от ходове направени по време на игра, а не просто състоянието на шахматната дъска. Тази нотацията също записва подробна информация, като например имената на играчите, таймера, резултата и още много.

```
[Event "Live Chess"]
[Site "Chess.com"]
[Date "2020.03.25"]
[Round "-"]
[White "pdrpnht"]
[Black "ColinStapczynski"]
[Result "0-1"]
[WhiteElo "1543"]
[BlackElo "2241"]
[TimeControl "180"]
[Termination "ColinStapczynski won by resignation"]

1. e4 d5 2. exd5 Qxd5 3. Nc3 Qa5 4. d4 c6 5. Nf3 Bf5 6. Bd3 Bxd3 7. Qxd3 e6 8.
O-O Nf6 9. Bg5 Nbd7 10. Ne5 Qc7 11. Ne2 Nxe5 12. dxe5 Qxe5 13. Bxf6 gxf6 14.
Rfe1 Bd6 15. Ng3 Qd5 16. Rad1 Qxd3 17. Rxd3 O-O-O 18. Red1 Be7 19. Ne4 Rxd3 20.
Rxd3 Rd8 21. Rh3 f5 0-1
```

Фиг. 1.9

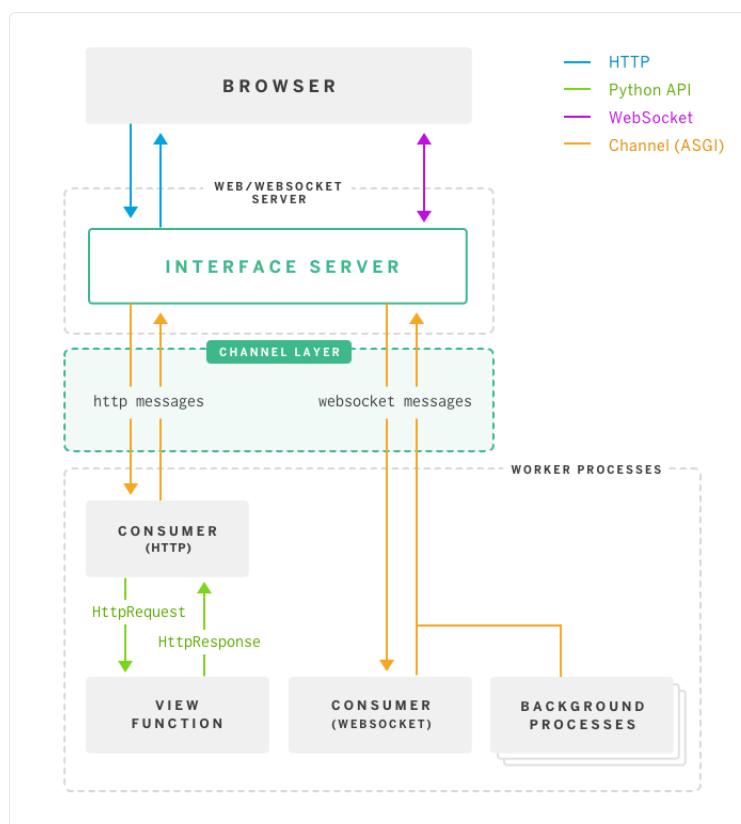
Информация от PGN

ВТОРА ГЛАВА

ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА WEB БАЗИРАНО ПРИЛОЖЕНИЕ ЗА ИГРА НА ШАХ

2.1 Функционални изисквания на проекта

Проектът ще се състои от три основни компонента WEB контролер, API и ГПИ. Приложението трябва да представлява приятна и удобна за използване система, която позволява на потребителите си да играят шах. Всеки играч ще има две възможности: да си създаде собствена игра или да влезе в чужда. Когато се създаде нова игра тя ще се запазва в базата данни, заедно с идентификатор за създателя ѝ. При влизане в игра ще се изисква код, чрез който да се намери исканата да бъде достъпена от потребителя игра. Играчите ще могат да местят фигурите си, които ще се обработват от логиката на приложението и се запазват в базата данни.



Фиг. 2.1

Работа на приложението

2.1.1 WEB контролер

WEB контролерът ще зарежда на определени адреси ГПИ и API, които ще могат да се достъпват и манипулират. Трябва да може да взима въведената или поискана информация и с нея да управлява базата данни. Информацията ще съдържа наредбата на шахматна дъска и реда на играчите, както и код за идентификация. Ще бъде възможно свързването на друг потребител в дадена игра, за да се съобщава приключването на ход от страна на единия играч, това ще става благодарение на индивидуалния код за достъп.

2.1.2 Графичен потребителски интерфейс

Лесен и приятен за използване ГПИ, който ще представя шахматната дъска и нейните фигури на потребителите си. Трябва да позволява на играчите да местят фигурите и ще проверява валидността на дадения ход, благодарение на вградената му логика, която ще се направи. Трябва да може да прави заявки, чрез които да обновява състоянието на дъската или да взима запазеното.

2.2 Съображения за избор на програмни средства и развойната среда

2.2.1 WEB контролер

Езикът избран за реализацията на WEB контролера е Python, а по-точно framework Django.

- **Python**

Python е програмен език от високо ниво, той няма конкретна употреба, използва за всичко. Това е един от най-използваните езици и то поради големия набор от библиотеки, лесната му четимост и мащабното му приложение.

- **Django**

Django е WEB framework за Python, който позволява бързо и лесно създаване на нов проект. Използва улеснен начин за манипулиране на база данни (“Model”), система за обработка на HTTP заявки (“View”), поддръжка на огромен набор от библиотеки, чрез които приложението разширява своята функционалност . Това е един от по-използваните методи за създаване на WEB приложение на Python, поради което има много научни материали в Интернет.

- **Pipenv**

При разработката на това приложение е необходимо инсталирането на различни библиотеки за Python. Pipenv е съвкупност от мениджър на библиотеки и виртуална среда. Виртуалната среда е необходима за едно WEB приложение, за да може инсталацията на допълнителни библиотеки да бъде само в този проект и да не се смесва с други приложения.

- **SQLite3**

Почти всички WEB приложения използват база данни. Това е място за съхранение на данни свързани с апликацията. За реализацията на този проект е необходима таква, за запазването на информация свързана с шахматната дъска и нейните потребители. При създаването на Django проект, по подразбиране, се използва SQL езикът SQLite3.

- **Django REST framework**

Това е библиотека за създаване на WEB API. Тя е много добра интегрирана с Django. Поддържа “Serialization”, което представлява превръщането на Django “Model” в Python тип данни, които могат да бъдат използвани, като JSON, XML.

2.2.2 Графичен потребителски интерфейс

За направата на графичния потребителски интерфейс е избран програмния език JavaScript, а по-конкретно библиотеката React.

- **JavaScript**

JavaScript е програмен език, който е обектно ориентиран и поддържа функционален стил на програмиране. Това е един от най-популярните и използвани езици, което предоставя широк набор от материали за изучаване и библиотеки, предоставящи допълнителни услуги. Най-често езикът се използва в WEB приложенията, за добавяне на допълнителна функционалност.

- **React**

Това е библиотека на JavaScript за създаване на ГПИ. Той поддържа компоненти, които могат да менажират своето състояние и да се презареждат автоматично. Това е една от най-разпространените библиотеки на JavaScript за създаването на ГПИ и има широк набор от библиотеки, направени специфично за съвместна употреба с нея, разширявайки основните ѝ функционалности.

- **Babel**

Babel е компилатор за JavaScript, който превръща “ECMAScript 2015+” код в използваем за стари версии на браузърите.

- **React-Bootstrap**

Това е най-използваният framework за React. В него се съдържат вече готови за използване елементи, които могат да се приложат в WEB приложението. Той поддържа промяна на стила, размера и други функционалности на елементите, така че приложението да бъде красivo и персонализирано.

- **React-Icons**

React-Icons е библиотеки за икони в React. Тя съдържа голям набор от разновидности, които може да използваме.

- **React Router DOM**

Тази библиотека предоставя възможността за създаване на маршрутизатор на страници.

- **Webpack**

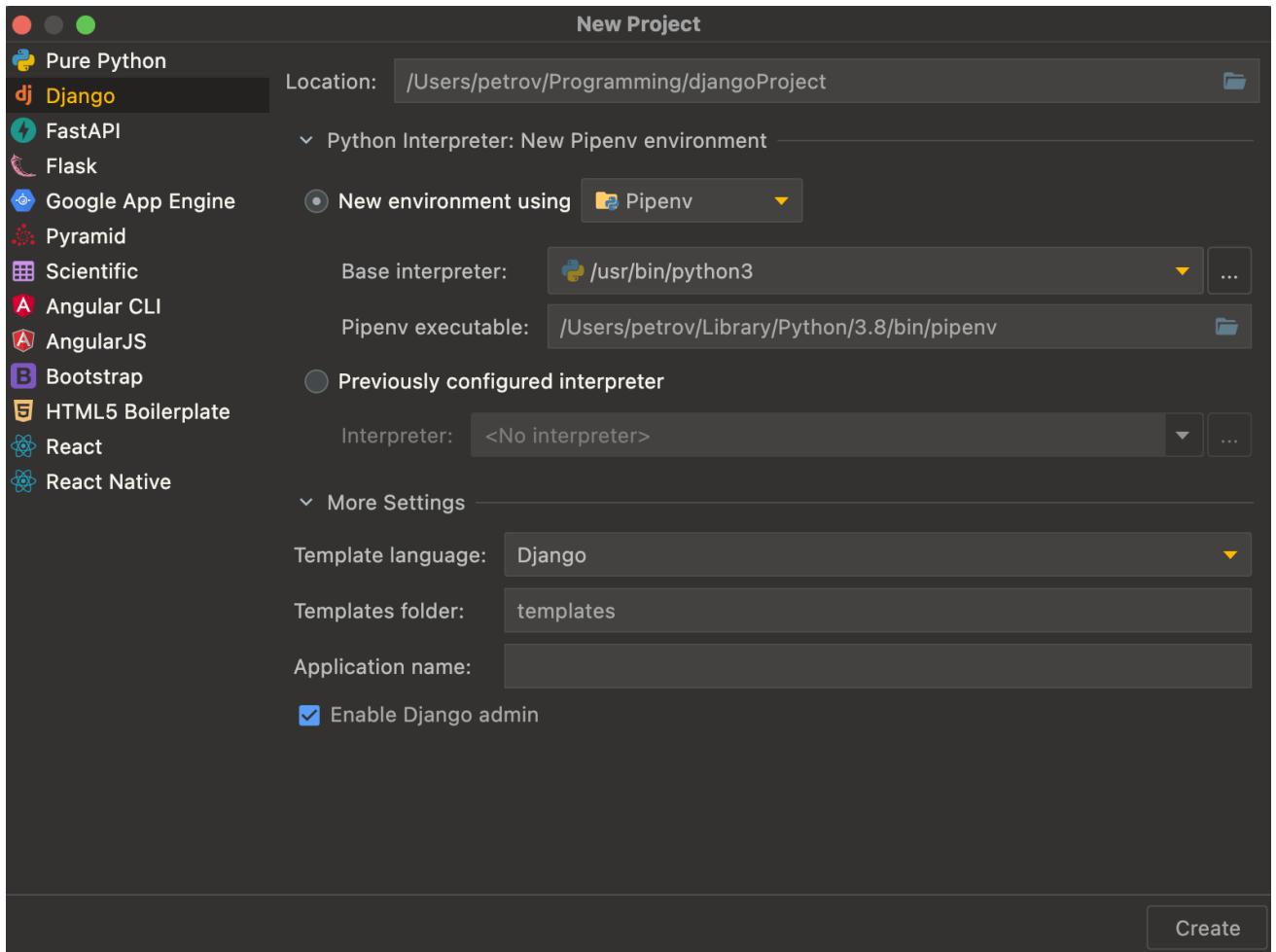
Webpack представлява библиотека, която пакетира файлове. Главното и предназначение е за групиране на JavaScript, но поддържа и други разширения.

- **React-copy-to-clipboard**

React-copy-to-clipboard е библиотека, която ни дава възможността за създаване на React компонент, който копира зададената му стойност и я поставя в клипборда на потребителя.

2.2.3 Среда за разработка

Проектът ще бъде написан с помощта на PyCharm. Това е IDE използвано за компютърно програмиране, направено от Чешката компания JetBrains. То е предназначено за езика Python, но поддържа и други. Има приятен и красив интерфейс, който може да бъде променян спрямо нуждите на потребителя. Основното му различие с други такива апликации е поддръжката на много приставки, които улесняват и забързват процеса на писане на код. Създаването на Django сървър и конфигурирането му е улеснено, което го прави удобен за изготвянето на този проект.



Фиг. 2.2 Създаване на Django проект

в PyCharm

2.3 Структура на базата данни

В базата данни ще се пази информация за всяка създадена шахматна дъска. Таблицата ще има идентификационен номер, генериран на случаен принцип код за достъп на играта, който трябва да е достатъчно сложен, за да се избегне случай на повтаряне. Трябва да има поле за записване на създателя на играта, с което да се разграничава от обикновения потребител. Подредба на фигурите, за да бъде използвана при зареждане на дъската. Ще има и поле за цвета на фигурите, които са били преместени последно.

Ще бъде създавана и пазена информация за текущата сесия на клиента. Информацията, която е необходима е персоналния ключ, който не се променя

при повторно отваряне на приложението. Този ключ ще бъде използван за създаване на връзка между шахматната дъска и нейния собственик.

api_chessboard	
id	integer
code	varchar(36)
host	varchar(64)
board	varchar(128)
personToMove	varchar(8)

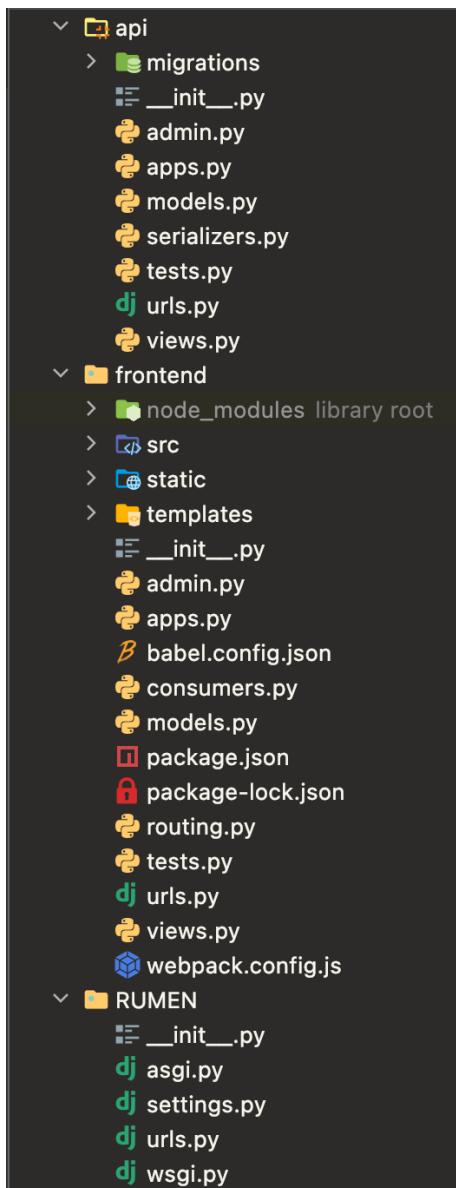
Фиг. 2.3 Таблица с данни за шахматната дъска

ТРЕТА ГЛАВА

ПРОГРАМНА РЕАЛИЗАЦИЯ НА WEB БАЗИРАНО ПРИЛОЖЕНИЕ ЗА ИГРА НА ШАХ

3.1 Създаване на основа за реализиране на проекта

Система е изграден от две основни части - контролер и ГПИ. За първата част се използва Django [10], създаден през PyCharm, използвайки Pipenv за управление на виртуалната среда и допълнителните библиотеки.



Фиг. 3.1 Структура на проекта

В проекта са създадени две отделни апликации - “api” и “frontend”, показано във **Фиг. 3.1**.

Api апликацията управлява базата данни, чрез създаването на адреси, към които се изпращат заявки. Използват се Django модели за създаването на таблици, съдържащи информация за шахматната дъска. С помощ на Django “Serializers” се обработват данните от заявките, за да бъдат лесно четими и използваеми.

Frontend е апликацията, която отговаря за ГПИ. В нея се зареждат страниците създадени с React. Те управляват визуалните елементи, както и допълните функционалности, като алгоритми за проверка, изпращане и приемане на данни и създаване на връзка между потребителите.

Проектът съдържа две апликации, за да могат да се достъпват адресите използвани във всяка една от тях се използва главен файл, който ги управлява.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),
    path('', include('frontend.urls'))
]
```

Фиг. 3.2 Адреси на апликациите

Базата данни, която се използва е SQLite3, създава се по подразбиране от Django.

За да се осъществи комуникация между двамата играчи се използва WebSocket комуникация и Django “Channels” [11], през която се изпраща съобщение за приключчен ход. (Обяснено в 3.3.4). Конфигурирането на връзката е направено в настройките на Django.

```
ASGI_APPLICATION = 'RUMEN.asgi.application'

CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels.layers.InMemoryChannelLayer'
    }
}

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'RUMEN.settings')

application = ProtocolTypeRouter({
    'http': get_asgi_application(),
    'websocket': AuthMiddlewareStack(
        URLRouter(
            frontend.routing.websocket_urlpatterns
        )
    )
})
```

Фиг. 3.3 Конфигуриране на Django Channels и WebSocket канал

3.2 Управление на базата данни

3.2.1 Django Models

Моделът, който записва данните в базата данни е “ChessBoard”, показан във **Фиг. 3.4**. Той съдържа няколко полета:

- **Code** - Генериран при създаване на обект, дълъг и сложен код, използвайки “uuid”. С него потребителите на приложението могат да се присъединяват към чужди шахматни дъски.
- **Host** - Това е ключ, генериран от сесията на потребителя, който се използва за създаване на връзка между дадена игра и нейния собственик.
- **Board** - Съдържа подредбата на фигурите, използвайки играден формат, от създадения алгоритъм за разчитане на шахматната дъска в ГПИ.
- **PersonToMove** - Представлява поле, което съдържа информация за цвета на фигурите, които са направили последния ход.

```
from django.db import models
import uuid

# Create your models here.

def codeGenerator():
    code = uuid.uuid4()
    if ChessBoard.objects.filter(code=code).exists():
        codeGenerator()

    return code


class ChessBoard(models.Model):
    code = models.CharField(max_length=36, default=codeGenerator, unique=True)
    host = models.CharField(max_length=64, unique=True, default='')
    board = models.CharField(max_length=128, default='bRbNbBbQbKbBbNbR/bPbPbPbPbPbPbP/8/8/8/wPwPwPwPwPwPwPwP'
                           '/wRwNwBwQwKwBwNwR')
    personToMove = models.CharField(max_length=8, default='black')
```

Фиг. 3.4 Моделът на шахматната дъска

3.2.2 Django Serializers

Използват се за превръщането на Django модели в Python достъпни обекти. Mogат да бъдат заредени, като JSON, XML и други. В проекта се използват за представяне на “ChessBoard” модела, като се използват само необходимите за функцията полета.

```
from rest_framework import serializers
from .models import ChessBoard

class ChessBoardSerializer(serializers.ModelSerializer):
    class Meta:
        model = ChessBoard
        fields = ('id', 'code', 'host', 'board', 'personToMove')

class CreateBoardSerializer(serializers.ModelSerializer):
    code = serializers.CharField(validators=[])

    class Meta:
        model = ChessBoard
        fields = ('code', 'board', 'personToMove')
```

Фиг. 3.5 Създадени Django Serializers

“ChessBoardSerializer” и “CreateBoardSerializer” са създадените Django “Serializers”. Разликата между тях са полетата “id” - автоматично генериран идентификационен номер за всяка таблица и “host”.

3.2.3 Django Views

Това е мястото, където са реализирани функциите за управление на заявки и базата данни. Използвайки библиотеката “rest_framework” е изграден API на приложението, с методи за създаване на дъска, вземане на информация и обновяване на данните ѝ.

Във **Фиг. 3.6** е реализирано създаването на шахматна дъска. Функцията приема “POST” заявки - обикновено се използват при направата на нови записи в базата данни.

```
class CreateBoardView(APIView):

    def post(self, request):
        if not self.request.session.exists(self.request.session.session_key):
            self.request.session.create()

        host = self.request.session.session_key
        board = 'bRbNbBbQbKbBbNbR/bPbPbPbPbPbPbP/8/8/8/8/wPwPwPwPwPwPwP/wRwNwBwQwKwBwNwR'
        personToMove = 'black'
        queryset = ChessBoard.objects.filter(host=host)
        if queryset.exists():
            chessBoard = queryset[0]
            chessBoard.board = board
            chessBoard.personToMove = personToMove
            chessBoard.save(update_fields=['board', 'personToMove'])
            return Response(ChessBoardSerializer(chessBoard).data, status=status.HTTP_200_OK)
        else:
            chessBoard = ChessBoard(host=host, board=board, personToMove=personToMove)
            chessBoard.save()
            return Response(ChessBoardSerializer(chessBoard).data, status=status.HTTP_201_CREATED)
```

Фиг. 3.6 Метод за създаване на шахматна дъска

Проверя се за налична сесия на извикалия функцията клиент, ако не съществува се създава. Променливите “host”, “board” и “personToMove” пазят стойности за инициализиране на една игра. След това се проверя за направена вече игра от дадения потребител, ако има такава тя се обновява, със стартова наредба на фигурите. Ако потребителя няма създадена игра, се генерира нова и се запазва. Връщат се данните на променената шахматна дъска и нейния статус.

Функцията за обновяване на дъска е показан във **Фиг. 3.7**, която приема “PUT” заявка - използва се за обновяване на данни.

```

class UpdateBoardView(APIView):
    serializer_class = CreateBoardSerializer

    def put(self, request):
        if not self.request.session.exists(self.request.session.session_key):
            self.request.session.create()
        serializer = self.serializer_class(data=request.data)

        if serializer.is_valid():
            personToMove = serializer.data.get('personToMove')
            board = serializer.data.get('board')
            code = serializer.data.get('code')
            queryset = ChessBoard.objects.filter(code=code)
            if queryset.exists():
                chessBoard = queryset[0]
                chessBoard.board = board
                chessBoard.personToMove = personToMove
                chessBoard.save(update_fields=['board', 'personToMove'])
                return Response(ChessBoardSerializer(chessBoard).data, status=status.HTTP_200_OK)

        return Response({'Bad Request': 'Invalid data...'}, status=status.HTTP_400_BAD_REQUEST)

```

Фиг. 3.7 Метод за обновяване на шахматна дъска

Проверява се наличността на сесия. Тук се използва “CreateBoardSerializer”, който проверява валидността на данните. Когато информацията е валидна, се търси съществуваща дъска с изпратения код в базата данни и ако се намери се обновява с въведените стойности. Ако всичко е преминало успешно се връща променената дъска и нейния статус, ако ли не се връща грешка.

Последният метод реализиран във **Фиг. 3.8** е за извлечане на информация на дадена дъска. Функцията приема “GET” заявки, които се използват при вземане на данни. В адреса за достъп до метода се въвежда кода на дъската, като след това се търси тази дъска в базата данни и се извлича нейната информация. Добавено е поле, което показва дали клиентът извикал заявката е създателя на играта. Ако не съществува дъска с въведения код се връща грешка.

```

class GetBoard(APIView):
    serializer_class = ChessBoardSerializer
    lookup_url_kwarg = 'code'

    def get(self, request):
        code = request.GET.get(self.lookup_url_kwarg)
        if ChessBoard.objects.filter(code=code).exists():
            board = ChessBoardSerializer(ChessBoard.objects.filter(code=code)[0]).data
            board['secondPlayer'] = self.request.session.session_key != ChessBoard.objects.filter(code=code)[0].host
            return Response(board,
                            status=status.HTTP_200_OK)
        return Response({'Bad Request': 'Invalid data...'}, status=status.HTTP_400_BAD_REQUEST)

```

Фиг. 3.8 Метод за извлечане на информация

3.2.4 DjangoUrls

В този файл са зададени адресите, от които могат се достъпват създадените методи.

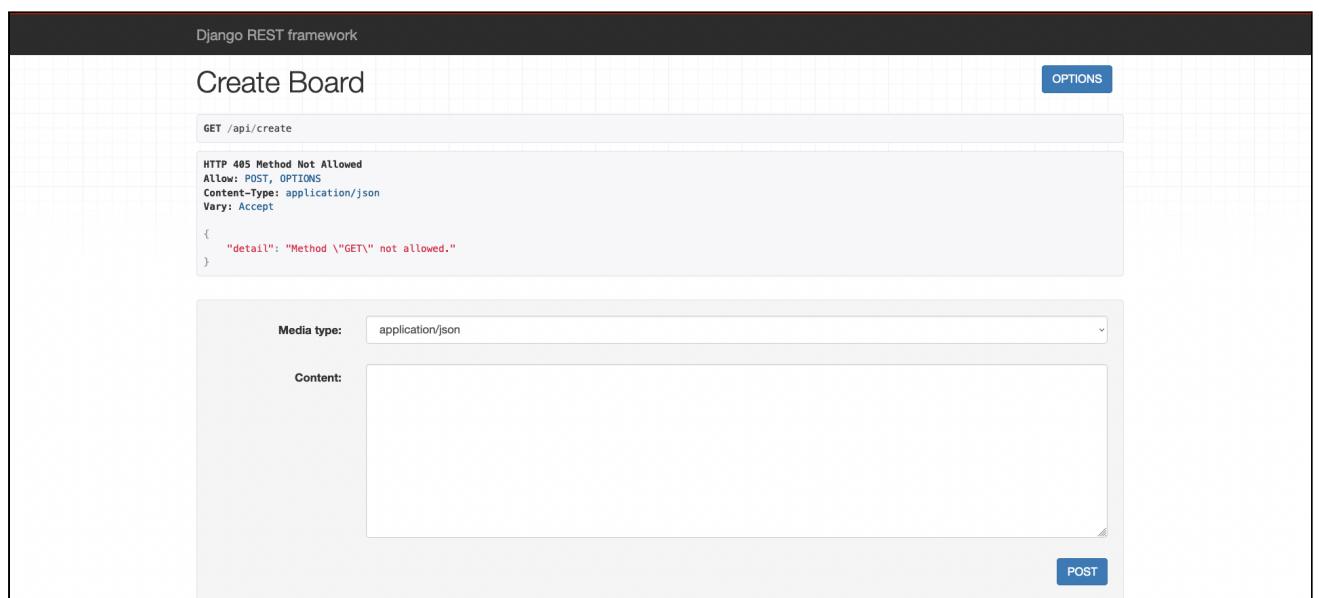
```

from django.urls import path
from .views import GetBoard, CreateBoardView, UpdateBoardView, BoardView

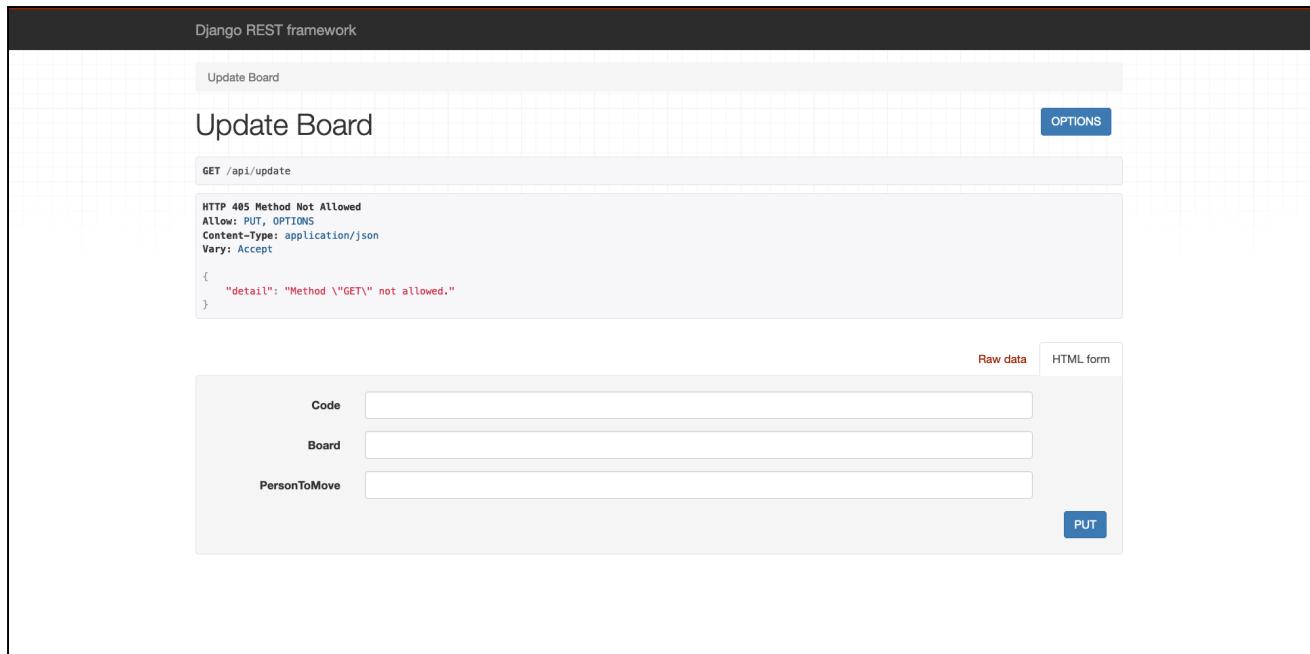
urlpatterns = [
    path('chessboard', GetBoard.as_view()),
    path('create', CreateBoardView.as_view()),
    path('update', UpdateBoardView.as_view()),
]

```

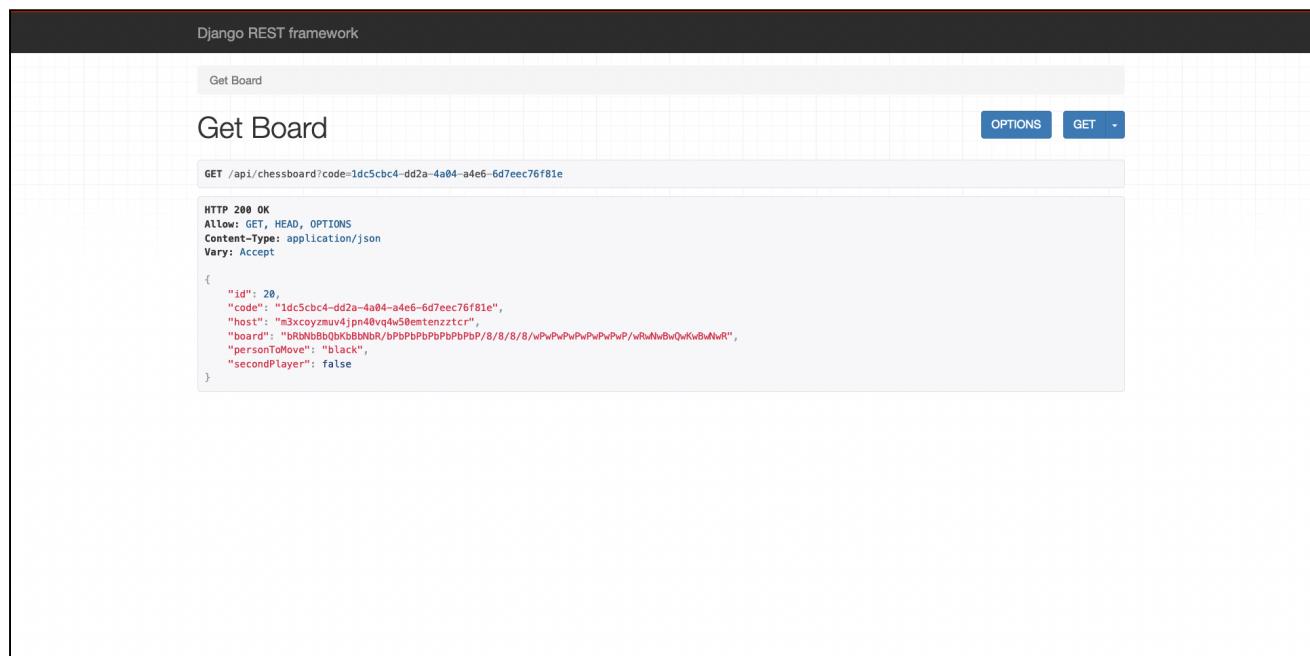
Фиг. 3.9 Адреси на функциите



Фиг. 3.10 Страница за създаване на шахматна дъска



Фиг. 3.11 Страница за обновяване на шахматна дъска



Фиг. 3.12 Страница за извличане на информация

3.3 Разработка на ГПИ

Графичният потребителски интерфейс представлява апликация в Django проекта. Използван е React за разработването му, като той се зарежда в HTML файл. В методите (“Views”) има само една функция, която извиква този файл.

```
def index(request, *args, **kwargs):
    return render(request, 'frontend/index.html')
```

Фиг. 3.13 Функция в frontend, която зарежда HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>RUMEN - Chess game</title>
    {% load static %}
    <link rel="icon" href="{% static "images/icon.png" %}" type="image/x-icon"/>
        <link
            rel="stylesheet"
            href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
            integrity="sha384-1BmE4kWBq78iYhFlvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
            crossorigin="anonymous"
        />
        <link rel="stylesheet" type="text/css" href="{% static "css/index.css" %}">
        <link rel="stylesheet" type="text/css" href="{% static 'css/ReactToastify.css' %}">
</head>
<body>
<div id="app">
</div>

<script src="{% static "frontend/main.js" %}"></script>
</body>
</html>
```

Фиг. 3.14 HTML файл, която зарежда скриптове

В HTML се вкарват файловете за украсяване (stylesheet), както и се зарежда събирателния на всички JavaScript файлове “main.js”.

За да се пакетират всички React компоненти на един място се използва Webpack, който е конфигуриран да събира всички JavaScript файлове в един (“main.js”). Използва се Babel за компилиране на код.

```

const path = require("path");
const webpack = require("webpack");

module.exports = {
  entry: "./src/index.js",
  output: {
    path: path.resolve(__dirname, "./static/frontend"),
    filename: "main.js",
  },
  module: {
    rules: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        use: {
          loader: "babel-loader",
        },
      },
    ],
  },
};

```

Фиг. 3.15 Конфигурация на Webpack

3.3.1 Маршрутизатор на страници

В “index.js” е създаден маршрутизатор на страници, който зарежда React компоненти на зададените им адреси.

```

function App() {

  return (
    <BrowserRouter>
      <Routes>
        <Route path=''' element={<Home/>} />
        <Route path='join' element={<JoinGame/>} />
        <Route path='join/:gameCode' element={<Chessboard/>} />
      </Routes>
    </BrowserRouter>
  );
}

render(<App/>, document.getElementById( elementId: 'app' ));|

```

Фиг. 3.16 Маршрутизатор на страници

Когато пътят е празен се зарежда “Home” компонентът. При път “/join” се зарежда “JoinGame”, а при въвеждане на “/join/:gameCode”, се зарежда

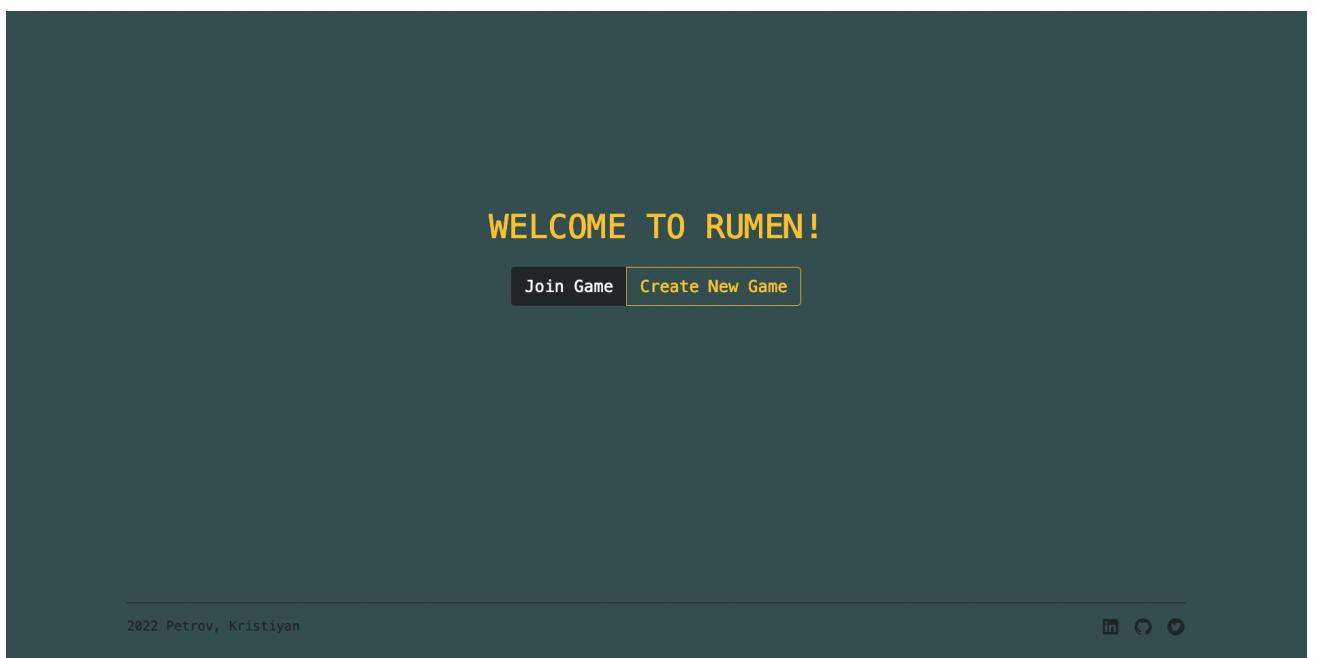
“Chesboard”, “:gameCode” представлява символен низ, който е написан след наклонената черта. Тези адреси са заредени и в Django “Urls”, показано във **Фиг. 3.17.**



Фиг. 3.17 Зареждане на адреси в Django

3.3.2 React Home компонент

Това е заглавната страница на приложението, която предоставя два бутона - за влизане в чужда игра или създаване на своя.



Фиг. 3.18 Заглавна страница на приложението

При създаване на собствена игра се използва функция показана във **Фиг. 3.19**, която изпраща заявка към API за направата на шахматна дъска и прехвърля потребителя към страницата за шах.

```

const createGame = () => {

    const sendPOST = {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify( value: {}),
    };
    fetch( input: '/api/create', sendPOST) Promise<Response>
        .then((response : Response ) => response.json()) Promise<any>
        .then((data) => window.location.href = '/join/' + data['code']);

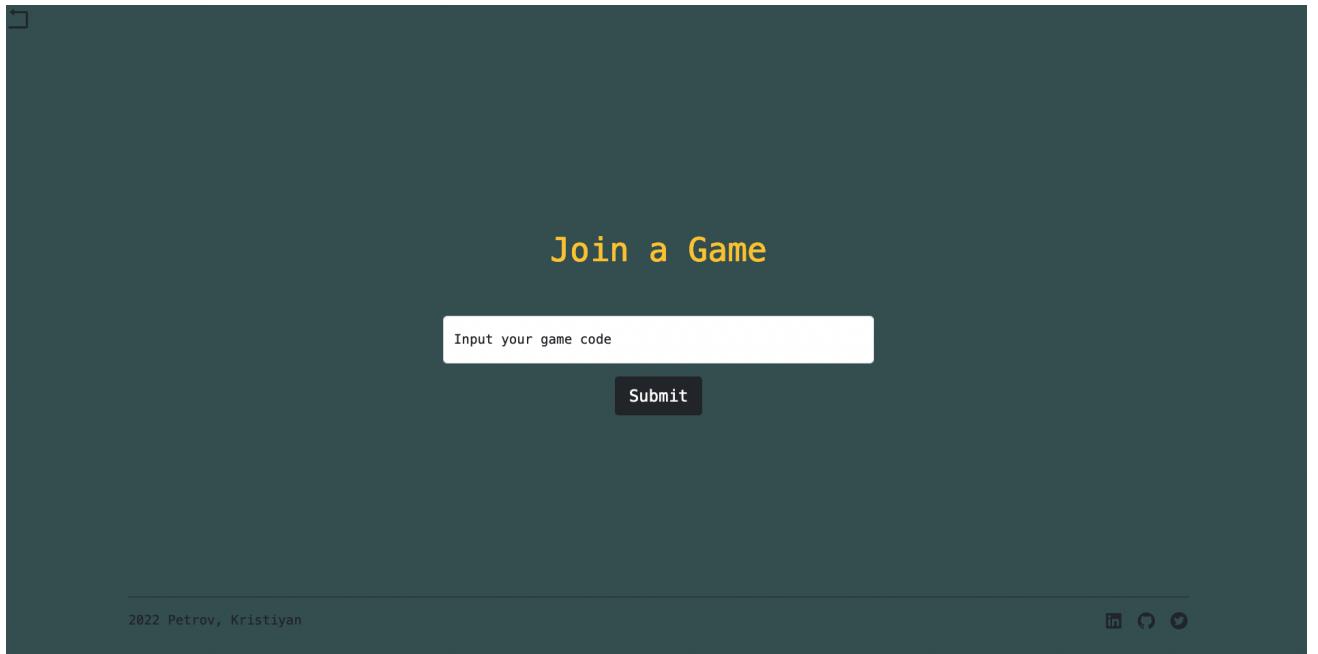
}

```

Фиг. 3.19 Функция за създаване на игра

3.3.3 React JoinGame компонент

При влизане в игра потребителят се прехвърля към компонента “JoinGame”.

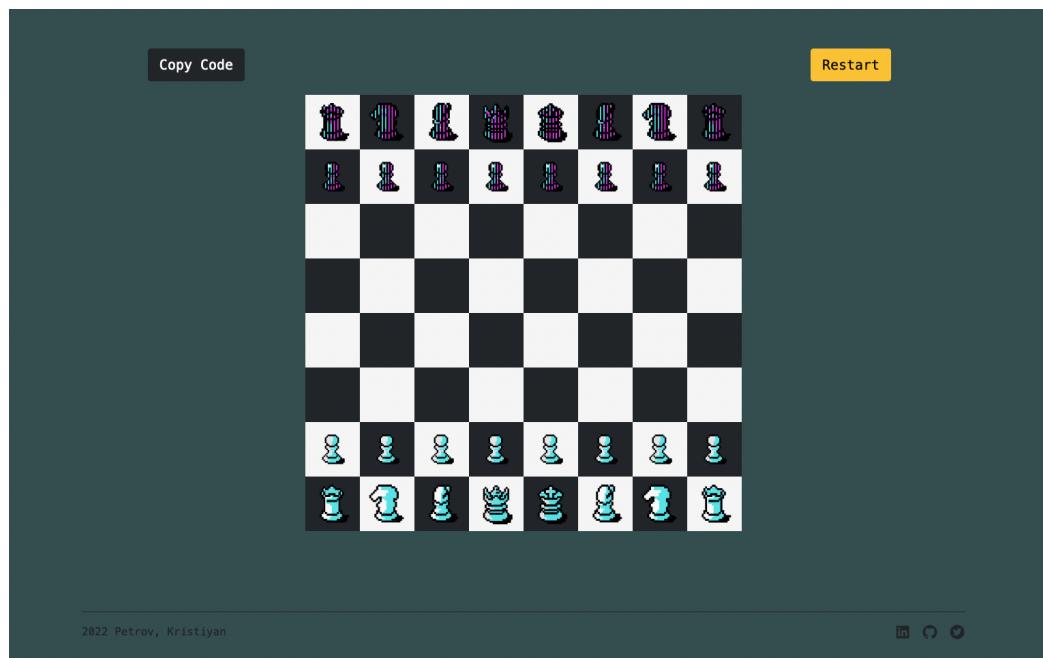


Фиг. 3.20 Страница за влизане в игра

Основната функционалност на тази страница е формата, в която се въвежда код за идентифициране на игра, след което прехвърля потребителя към компонента “Chessboard”.

3.3.4 React Chessboard компонент

Това е страницата, в която потребителите могат да играят шах помежду си, тя взаимодейства с API, алгоритми за проверка и WebSocket.



Фиг. 3.21 Страница за провеждане на игра на шах

Създаден е двуизмерен масив с размери 8x8, той се пълни с имена на фигури, като взима информацията за шахматната дъска от API.

```
useEffect( effect: () => {
  if (!didLoad) {
    code = window.location.pathname.split( separator: '/') [2];
    fetch( input: '/api/chessboard?code=' + code).then((response : Response ) => response.json()).then(data => {
      FENtoBoard(data['board']);
      flagForMove = data['personToMove'] === 'white';
      setDidLoad( value: true);
      secondPlayer = data['secondPlayer'];
    });
  }
}, deps: [didLoad]);
```

Фиг. 3.22 Извличане на информация и зареждане на данни

При първоначалното зареждане на страницата се взима информация за дъската, като се използва кодът от адреса. Символния низ за подредба на фигурите се обработва от функцията “FENtoBoard”, алгоритъм реализиран по подобие на

FEN, но направен според нуждите на проекта, след което се запазват данните за цвета на пионките, които са били последно преместени и това дали потребителя е създател на играта.

```
const FENtoBoard = (fen) => {

    let chessBoard = Array.from(Array( arrayLength: 8), mapfn: () => new Array( arrayLength: 8));
    let boardRows = fen.split('/');

    for (let y = 0; y < 8; y++) {
        let rowIndex = 0;
        for (let x = 0; x < 8; x++) {
            if (/^\d/.test(boardRows[y].charAt(rowIndex))) {
                let nullAmount = x + parseInt(boardRows[y].charAt(rowIndex))
                for (; x < nullAmount; x++) {
                    chessBoard[y][x] = null;
                }
                rowIndex++;
            }
            if (/^([a-zA-Z]).test(boardRows[y].charAt(rowIndex))) {
                chessBoard[y][x] = boardRows[y].charAt(rowIndex) + boardRows[y].charAt(rowIndex + 1);
                rowIndex += 2;
            }
        }
    }
    setBoard(chessBoard);
}
```

Фиг. 3.23 Алгоритъм за обработване на подредбата на фигурите

След извличане на символния низ, той бива прехвърлен към функцията показана във **Фиг. 3.23**. Тя го разделя в отделни елементи на масив, като символът “/” представлява означение за нов ред на дъската. Когато има число, то показва бройката на елементи, които трябва да са празни на дадения ред. Когато е буква, това означава, че е фигура. Фигурите са два символа, първата буква е “w” или “b”, представляща цвета на пионката. Втората буква дефинира типа:

- **P** - пешка
- **R** - топ
- **N** - кон
- **K** - цар
- **B** - офицер
- **Q** - кралица

Когато функцията завърши своето изпълнение запазва новата шахматна дъска.

При съхраняване на новата информация за играта се зареждат фигураните, чрез създаване на “div” класове в HTML, които зареждат на мястото на пиона, нейната снимка.

```
<div id='chessboard'>

{[...Array( arrayLength: 8).keys()].map((y : number ) => [...Array( arrayLength: 8).keys()].map((x : number ) => {
    let color = 'black';
    if ((x + y) % 2 === 0) {
        color = 'white'
    }
    if (board[y][x] != null) {
        return (
            <div onClick={() => redirect(y, x)} id={y + '' + x} className={color + ' tile'}>
                <img src='../static/images/chessPieces/' + board[y][x] + '.svg' alt='chess piece' />
            </div>
        );
    } else {
        return (
            <div onClick={() => redirect(y, x)} id={y + '' + x} className={color + ' tile'} />
        );
    }
})}

</div>
```

Фиг. 3.24 Зареждане на фигураните в страницата

Всяко едно поле поддържа функция за натискане, която премахва фигура и след това с ново натискане на друго място я поставя. Когато ходът е завършен, тоест е сложена махната пиона се изпраща заявка за обновяване. Тази заявка отправя данни за новата дъска, цвета на фигураните, които са били преместени и кода на играта. Дъската бива превърната във формата на символения низ, който е бил при получаване на данните чрез функцията “boardToFen”. След като бъде изпратена информацията, се съобщава по WebSocket канала, че е бил извършен ход от опонента, показано във **Фиг. 3.25**.

```

if (flag) {

    //removing the piece and saving its coordinates

    if (chessBoard[y][x] != null) {
        piece = chessBoard[y][x];
        chessBoard[y][x] = null;
        oldX = x;
        oldY = y;
        flag = false;
    }
} else {

    //checking if the move is legal, if not returning it to its original place

    if (rulesOfChess(y, x, chessBoard)) {
        chessBoard[y][x] = piece;
        const sendPOST = {
            method: 'PUT',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify( value: {
                personToMove: blackOrWhite,
                board: boardToFEN(chessBoard),
                code: code,
            }),
        };
        fetch( input: '/api/update', sendPOST).then(() => {
            requestSocket.send(JSON.stringify( value: {
                'message': 'update',
                'code': code,
            }));
        });
    }
}

```

Фиг. 3.25 Функция за преместване на фигури

За да бъде осъществена WebSocket връзка при зареждане на страницата се отваря канал към даден адрес, който подслушва за съобщения от тип обновяване на дъската. При свързването се използва Django “Websocket Consumer”. Той поддържа три вида функции - за свързване, получаване на данни и изпращане.

```
websocket_urlpatterns = [
    re_path(r'ws/socket-server/', consumers.RequestConsumer.as_asgi())
]
```

Фиг. 3.26 Канал за осъществяване на WebSocket вързка

```
class RequestConsumer(WebSocketConsumer):
    def connect(self):
        self.room_group_name = 'test'

        async_to_sync(self.channel_layer.group_add)(
            self.room_group_name,
            self.channel_name
        )
        self.accept()

    def receive(self, text_data=None, bytes_data=None):
        message = json.loads(text_data)['message']
        code = json.loads(text_data)['code']
        async_to_sync(self.channel_layer.group_send)(
            self.room_group_name,
            {
                'type': 'chat_message',
                'message': message,
                'code': code,
            }
        )

    def chat_message(self, event):
        message = event['message']
        code = event['code']

        self.send(text_data=json.dumps({
            'type': 'chat',
            'message': message,
            'code': code,
        }))
```

Фиг. 3.27 WebSocketConsumer

При свързване към канала, потребителят става “Consumer”. Той има няколко метода:

- **Свързване** - Когато потребител се опита да осъществи връзка се добавя към група и канал, след което се приема неговата заявка за присъединяване.
- **Примене на съобщение** - Когато каналът получи съобщение, той го обработва, като запазва данните и ги препраща нужната информация към групата.
- **Изпращане на съобщение** - Изпраща се информация съдържаща код и съобщение от тип “chat”.

При получаване на съобщение от канала в ГПИ, се проверя дали се отнася за дадената игра на потребителя. Сверява се кодът на играта, дали съответства с текущия, дали типа е “chat” и съобщението е за обновяване, показано във **Фиг. 3.28.**

```
const requestSocket = new WebSocket(`ws://${window.location.host}/ws/socket-server/`)
requestSocket.onmessage = (e : MessageEvent) => {
  let data = JSON.parse(e.data);
  if (data.message === 'update' && data.type === 'chat' && data.code === code) {
    setDidLoad({ value: false });
  }
}
```

Фиг. 3.28 Осъществяване на връзка и обработване на информация в ГПИ от WebSocket

Преди да бъде завършен ходът на играча, той трябва да се одобри от алгоритмите за проверка на вярността, които са направени спрямо правилата на играта шах. Всяка една фигура се мести по различен начин, затова има проверки за всички.

3.3.5 Алгоритъм за проверка на изигран ход

Алгоритъма проверява всички възможни ходове за преместената фигура, спрямо правилата на играта шах [12]. Всяка пionка се мести по различен начин. Единствено коня може да прескача други фигури и не може да се взема цар. Когато играчът е под шах, трябва да се прекрати, чрез прикриване на линията на атака, вземане на застрашаваща пionка или преместване на царя.

Всеки играч има равен брой фигури, които се различават по цветове:

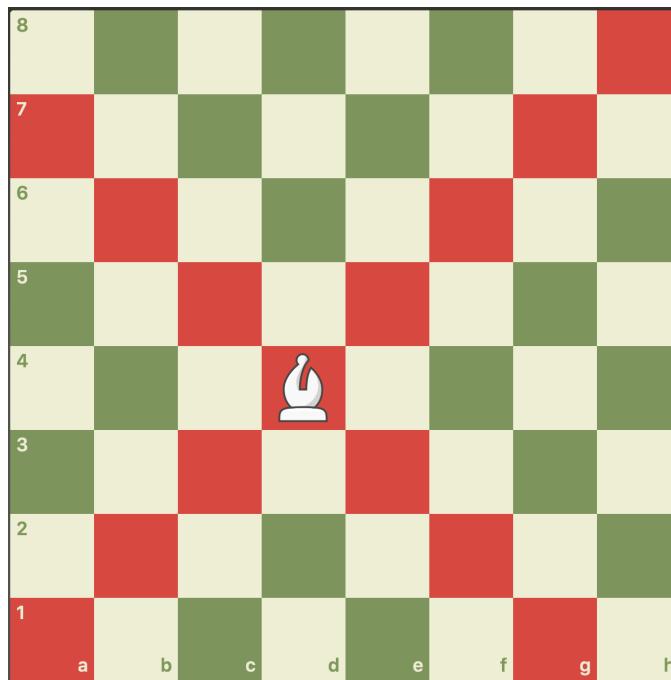
- 8 пешки
- 2 офицера
- 2 коня
- 2 топа
- 1 царица
- 1 цар

Една пешка може да се мести с две полета напред, когато началната ѝ позиция е стартовата и в играта. Пионката се мести напред или по диагонал, когато има фигура, която може да вземе. Когато пешката стигне до края на дъската се превръща в царица.



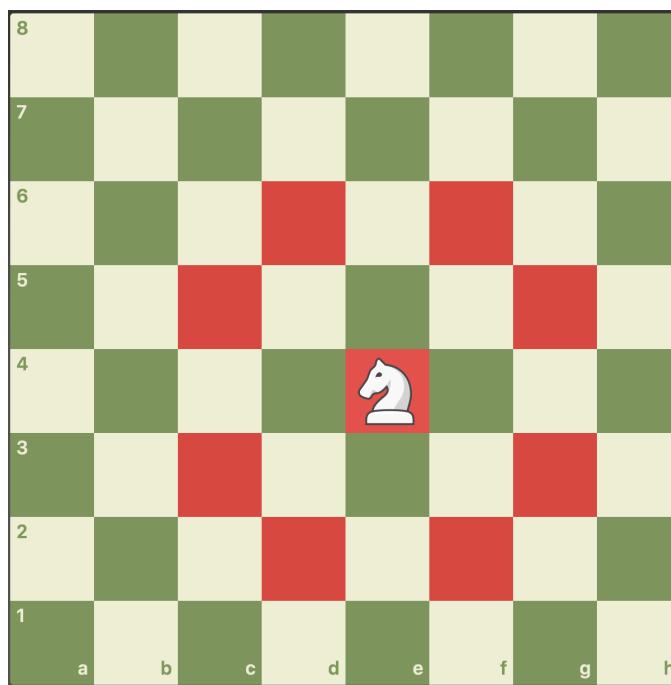
Фиг. 3.29 Местене на пешка

Офицерите се движат по диагонал, като единия започва на бяло поле, а другия на черно. Те не могат да сменят цвета диагонала, който ползват. Не им е позволено да прескачат фигури, но могат да се спрат на първата, като я вземат.



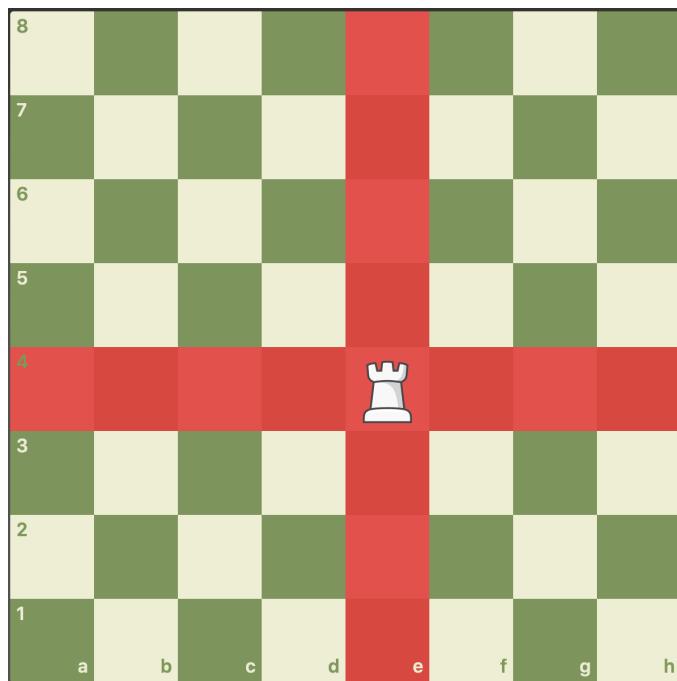
Фиг. 3.30 Местене на офицер

Конете са единствените фигури, които могат да прескачат други. Движат се под формата на буквата “L”. Взима пионките върху които падне, а не тези, които прескочи.



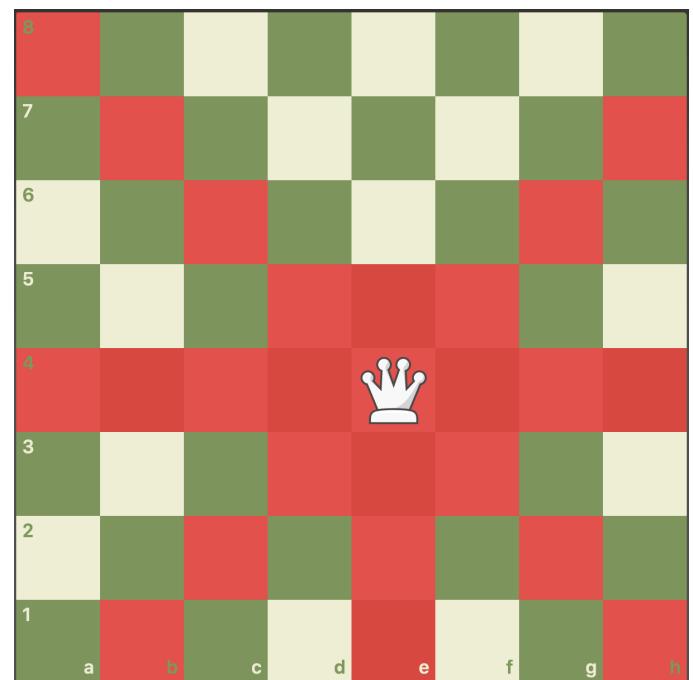
Фиг. 3.31 Месетене на кон

Топът се мести в права линия по хоризонтала или вертикалa. Няма определен лимит за преместени квадратчета, но не може да минава през фигури, както офицера взима първата срещната.



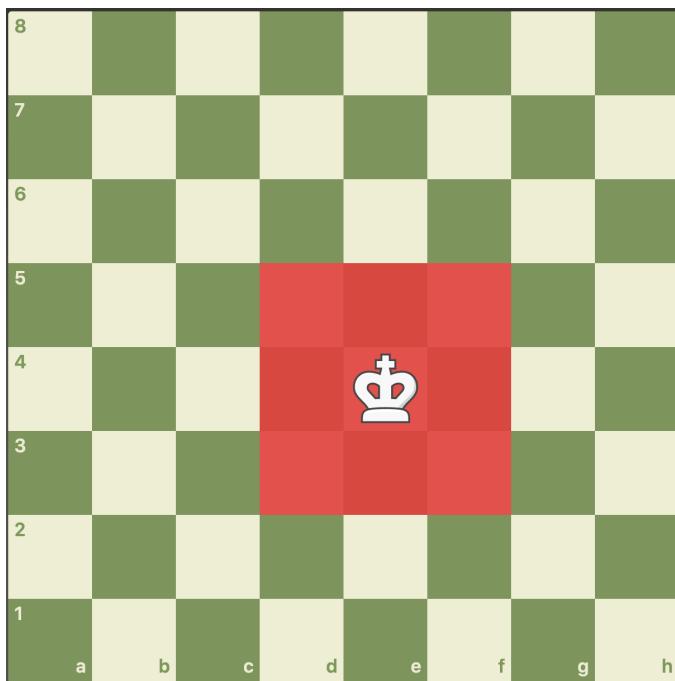
Фиг. 3.32 Местене на топ

Царицата е най-подвижната фигура, която включва начина на местена на топа и офицера. Не е ограничена в броя квадратчета, като пешката, но не може да прескача пционки, както коня. Взима първата срещната фигура.



Фиг. 3.33 Местене на царица

Последната и най-важна фигура е царят. Той може да бъде местен само с едно поле във всяка посока, като полето не трябва да бъде атакувано от противникова пionка. Когато царят няма позия в която да отиде и е под шах играта приключва, а когато не е под шах, но трябва да бъде преместен и няма къде, играта завършва равна. Царят може да прави рокада с топа, ако двете фигури не са премествани, когато и двете пionки и полетата между тях не са атакувани. Царят прескача едно поле на ляво или на дясно, а топът застава в прескоченето квадратче.



Фиг. 3.34 Местене на цар



Фиг. 3.35 Рокада

След успешно преминаване на функцията за проверка, преместването се изпълнява, ходът бива завършен и шахматната дъска се обновява. След което е ред на другия играч, чието изиграване минава през същите стъпки.

ЧЕТВЪРТА ГЛАВА

РЪКОВОДСТВО ЗА ПОТРЕБИТЕЛЯ

4.1 Конфигурационни изисквания

За използването на системата са необходимите следните изисквания:

- Python - v3.8.9 [13]
- Node.js - v17.4.0 with npm [14]
- Git - Latest Version [15]

4.2 Инсталационни инструкции

При инсталация на системата, конфигурационните изисквания трябва да са налице. За да бъде успешно стартирано приложението, трябва да се изпълнят следните стъпки:

1. Изтегляне на проекта от GitHub:

```
git clone https://github.com/krispetrov/rumen.git
```

2. Влизане в папката на приложението:

```
cd rumen/RUMEN/
```

3. Инсталиране на pipenv:

```
pip3 install pipenv
```

4. Създаване на виртуална среда:

```
pipenv shell
```

5. Сваляне на необходимите библиотеки за контролера:

```
pipenv sync
```

6. Влизане в директорията на ГПИ:

```
cd frontend
```

7. Инсталиране на пакетите за ГПИ:

```
npm i
```

8. Компилиране на ГПИ:

```
npm run build
```

9. Конфигуриране на базата данни:

```
cd ..
```

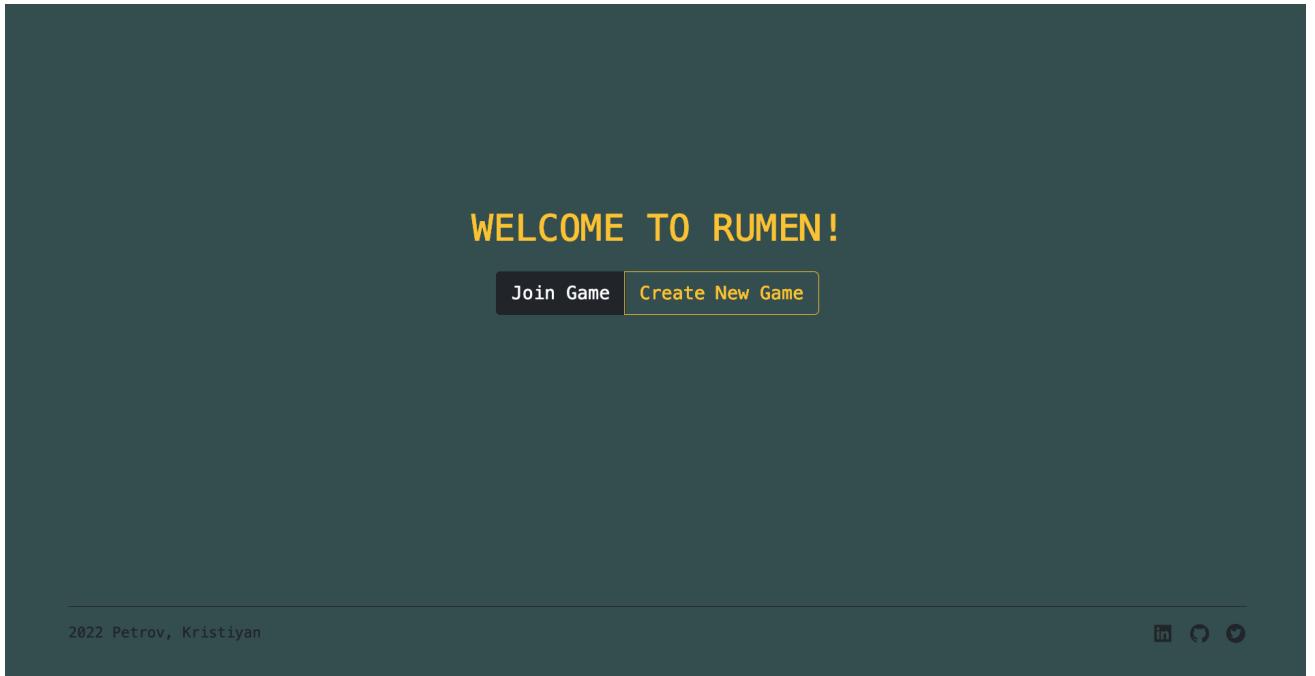
```
python3 manage.py migrate
```

10.Стартиране на Django приложението:

```
python3 manage.py runserver
```

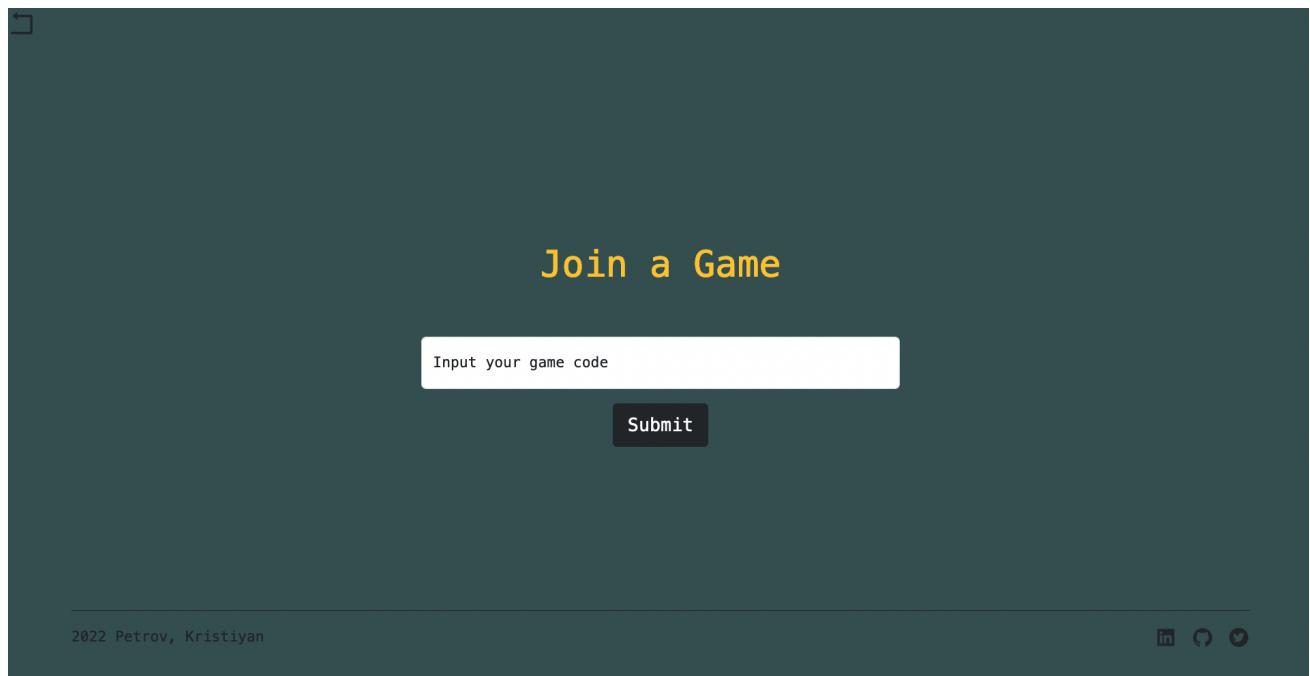
4.3 Инструкции за употреба

Структурата на приложението е опростена и лесна за употреба. За отваряне на заглавната страница, трябва да се използва адреса “127.0.0.1:8000”. Началната страница е показана във **Фиг. 4.1**. В нея има два бутона - за влизане в игра или създаване на своя. За създаване на собствена игра на шах, в която да влезе друг играч се използва бутона “Create New Game”. Ако потребителят има своя игра, със същия бутон се влиза във вече създадената стая, но играта се рестартира. При желание за влизане в чужда игра се натиска бутона “Join Game”.

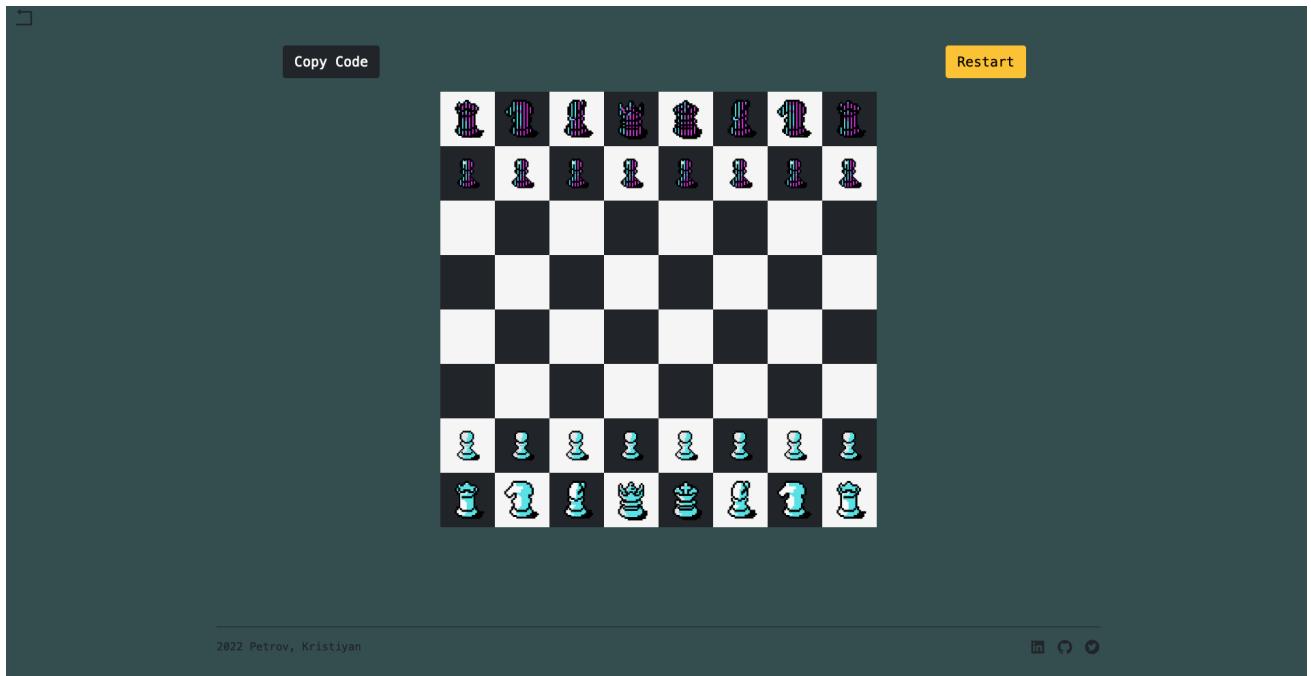


Фиг. 4.1 Начална страница

Страницата за влизане в игра е показана на **Фиг. 4.2**. В нея се въвежда код на съществуваща игра, който препраща потребителя към нея. В горният ляв ъгъл има бутон за връщане към началната страница. При въвеждане на кода се използва бутона "Submit" или натискане на "Enter" на клавиатурата.



Фиг. 4.2 Страница за влизане в игра



Фиг. 4.3 Игра на шах

В тази страница се реализира играта на шах. В нея има 3 бутона - в горният ляв ъгъл е бутона за връщане към начална страница, “Copy Code” поставя кода на играта в клипборда на потребителя и “Restart”, който рестартира играта, но само собственикът може да го използва. За местенето на фигури се натиска ляв бутон на мишката върху избрана от играча пionка, след което тя изчезва от полето и с повторно натискане на бутона върху друго поле, тя се поставя, но само ако ходът е валиден. Собственикът на играта може да премества само белите фигури, след изиграване на хода си, той чака друг играч да се присъедини и да премести черните фигури.

Заключение

Разработената дипломна работа реализира онлайн система, за провеждане на игра на шах. В хода на разработката едно от изискванията е заменено, защото е сметнато, че системата ще бъде по-скалируема. Премахнато е изискването, за създаване на най-оптимален ход, като на негово място са добавени функционалности - API на приложенето, който да управлява базата данни, поддържане на няколко стаи за игри, като различни клиенти могат да създават свои или да влизат в чужди. Направена е WebSocket комуникация, която да обновява данните в конкретната шахматна игра, при завършен ход на даден играч.

Приложението има много място за развитие, като добавяне на турнири по шах, уроци, които позволяват на потребителите да учат различни атаки или защити в играта на шах. Създаване на табло за точкуване, на база играчите с най-много победи и т.н.

Използвана литература

1. Number of internet users worldwide from 2005 to 2021

<https://www.statista.com/statistics/617136/digital-population-worldwide/>

2. Website vs Web App: What's the Difference?

<https://medium.com/@essentialdesign/website-vs-web-app-whats-the-difference-e499b18b60b4/>

3. Number of video gamers worldwide in 2021

<https://www.statista.com/statistics/293304/number-video-gamers/>

4. A 15-Step Guide on How to Build a Web App

<https://www.netsolutions.com/insights/how-to-build-a-web-app/#1-source-an-idea/>

5. How Many Chess Players Are There In The World?

<https://www.chess.com/article/view/how-many-chess-players-are-there-in-the-world/>

6. Lichess

<https://lichess.org/>

7. Chess.com

<https://www.chess.com/>

8. Forsyth-Edwards Notation (FEN)

<https://www.chess.com/terms/fen-chess/>

9. Portable Game Notation (PGN)

<https://www.chess.com/terms/chess-pgn/>

10. Django Documentation

<https://docs.djangoproject.com/en/4.0/>

11. Django Channels - Documentation

<https://channels.readthedocs.io/en/latest/>

12. Chess Terms - Chess Pieces

<https://www.chess.com/terms/chess-pieces/>

13. Python v3.8.9

<https://www.python.org/downloads/release/python-389/>

14. Node.js v17.4.0 with npm

<https://nodejs.org/download/release/v17.4.0/>

15. Git Downloads

<https://git-scm.com/downloads/>

Съдържание

Увод	3
Използвани термини	5
ПЪРВА ГЛАВА	6
1.1 Основни принципи, технологии и развойни среди за разработване на WEB приложение	6
1.1.1 Обзор на WEB приложение	6
1.1.2 Принципи за разработване на WEB приложение	7
1.1.3 Технологии за разработване на WEB приложение	8
1.1.4 Развойни среди за разработване на WEB приложение	9
1.2 Съществуващи решения и реализации	9
1.2.1 Алгоритъм за описание на достигната позиция позиция повреме на игра на шах (FEN)	12
1.2.2 Алгоритъм за записване на всички изиграни ходове в една шахматна игра (PGN)	14
ВТОРА ГЛАВА	15
2.1 Функционални изисквания на проекта	15
2.1.1 WEB контролер	16
2.1.2 Графичен потребителски интерфейс	16
2.2 Съображения за избор на програмни средства и развойната среда	16
2.2.1 WEB контролер	16
2.2.2 Графичен потребителски интерфейс	18
2.2.3 Среда за разработка	19
2.3 Структура на базата данни	20
ТРЕТА ГЛАВА	22
3.1 Създаване на основа за реализиране на проекта	22
3.2 Управление на базата данни	24
3.2.1 Django Models	24
3.2.2 Django Serializers	25
3.2.3 Django Views	25
3.2.4 DjangoUrls	28
3.3 Разработка на ГПИ	30
3.3.1 Маршрутизатор на страници	31
3.3.2 React Home компонент	32
	52

3.3.3 React JoinGame компонент	33
3.3.4 React Chessboard компонент	34
3.3.5 Алгоритъм за проверка на изигран ход	40
ЧЕТВЪРТА ГЛАВА	45
4.1 Конфигурационни изисквания	45
4.2 Инсталационни инструкции	45
4.3 Инструкции за употреба	46
Заключение	49
Използвана литература	50