



ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Уеб базирана 2D игра с управление на материали и автоматизиране
на производство

Дипломант:

/Йордан Мирославов Златанов/

Научен ръководител:

/инж. Марио Бенов/

СОФИЯ

2022



ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Дата на заданието: 14.12.2021 г.
Дата на предаване: 14.03.2022 г.

Утвърждавам:.....
/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 Йордан Мирославов Златанов от 12 "В" клас

1. Тема: Уеб базирана 2D игра с управление на материали и автоматизиране на производство
2. Изисквания:
 1. Клиентско приложение, базирано на JavaScript и Phaser Game Engine
 2. Java сървър за играта, управляващ потребителя и игровите сесии
 3. Реално времева комуникация, основана върху web сокети използвайки SocketIO
 4. Система за създаване и поставяне на обекти от играча, без да се пресичат с други
 5. Система за консумиране и производство на материали от взаимосвързани елементи
 6. Структура за транспортиране и съхранение на предмети и течности (напр. конвейери, тръби, контейнери, резервоари и т.н.)
 7. Създаване и конфигуриране на автономни устройства (напр. дронове, роботи и т.н.)
3. Съдържание
 - 3.1 Теоретична част
 - 3.2 Практическа част
 - 3.3 Приложение

Дипломант:.....
/Йордан Мирославов Златанов /
Ръководител:.....
/инж. Марио Бенов/
Директор:.....
/доц. д-р инж. Ст. Стефанова/



СТАНОВИЩЕ

на дипломния ръководител

инж. Марио Бенов

за дипломната работа на Йордан Златанов на тема

“Уеб базирана 2D игра с управление на материали и автоматизиране на производство”

1. Становище за работата на дипломанта по дипломния проект
Дипломантът е избрал технологии и езици, които са му добре познати. Използва успешно Projects и Issues функционалностите на GitHub за да организира задачите към проекта. Не е имал нужда от помощ извън периодични проверки на прогреса. В заданието са изброени много и различни елементи в играта, но времето на дипломанта не стигна за да бъдат имплементирани всичките и да се подsigури правилната им работа.
2. Степен на завършеност
Клиент-сървърната част на играта и комуникацията чрез уебсокет съобщения е завършена. Основната логика и структура на играта е завършена, както и достатъчно голямо разнообразие от изброените игрови елементи за да могат да бъдат демонстрирани.
3. Готовност за допускане до защита
Дипломантът е готов да бъде допуснат до защита.
4. Предложение за рецензент
За рецензент на дипломната работа предлагам Деница Балабанова, тел. 0885645623, email denitsa.balabanova@haemimont.com от Хемимонд АД.

29.03.2022

инж. Марио Бенов

УВОД

В днешните времена виждаме растеж и наличност на всякакъв вид видео игри и тяхната консумация. Историята на видеоигрите започва през 50-те и 60-те години на миналия век, когато компютърните учени започват да проектират прости игри и симулации на големи компютри, със „Spacewar“ на MIT. през 1962 г. като една от първите подобни игри, които се играят са с видео дисплей. В началото на 70-те години на миналия век се появи първият хардуер за видеоигри: първата конзола за домашна видеоигри, „Magnavox Odyssey“, и първите аркадни видеоигри, „Computer Space“ и „Pong“, последният, по-късно беше превърнат във версия за домашна конзола. Много компании тогава се появиха, за да уловят успеха на Pong както в аркадните игри, така и в дома, като създават клонинги на играта. След индустриалната революция на транзисторите и понижаването на размера им, хората видяха че могат да поберат много повече калкулативна мощност в много по – малко пространство, създавайки основите върху които модерното общество е стъпило по отношение на технологията. Това какво означава за видео игрите обаче? Да, те вече са много повече и могат да бъдат консумирани от много видове устройства, но

растежа на мощност на технологиите винаги върви ръка за ръка с сложността на игрите. Това означава, че дори да имаш последна генерация компоненти на компютъра/телефона, е сигуно че няма да бъде достигната максимална производителност. Няколко млади платформи се опитват да решат този проблем със “Поточна” услуга като “Google Stadia” или „Geforce NOW“, но те изискват месечен абонамент и сравнително по – добра и стабилна кабелна интернет връзка от стандартната. Това е заради високата величина на данни нужни да бъдат пренесени през нея включвайки не само графична информация но и метадата за играта. Много от тези игри също използват не достатъчно ефективни алгоритми за четене/променяне на данни за сметка на потребителя, за да се пести компютационно време от страна на издателите на услугата. Освен тези проблеми, се забелязва тенденция на пазара една игра да има първо добро качество графики, преди дори всичко останало, но цената за навлизането в тези игри е висока поради нарастващата инфлация на видео карти на пазара. Настоящата дипломна работа цели да покаже хибридно решение на този проблем.

ПЪРВА ГЛАВА

Проучване на технологии, среди за разработка и подобни игри

1.1. Основни жанрове на игри в днешния свят

1.1.1 Научно-фантастичните компютърни игри

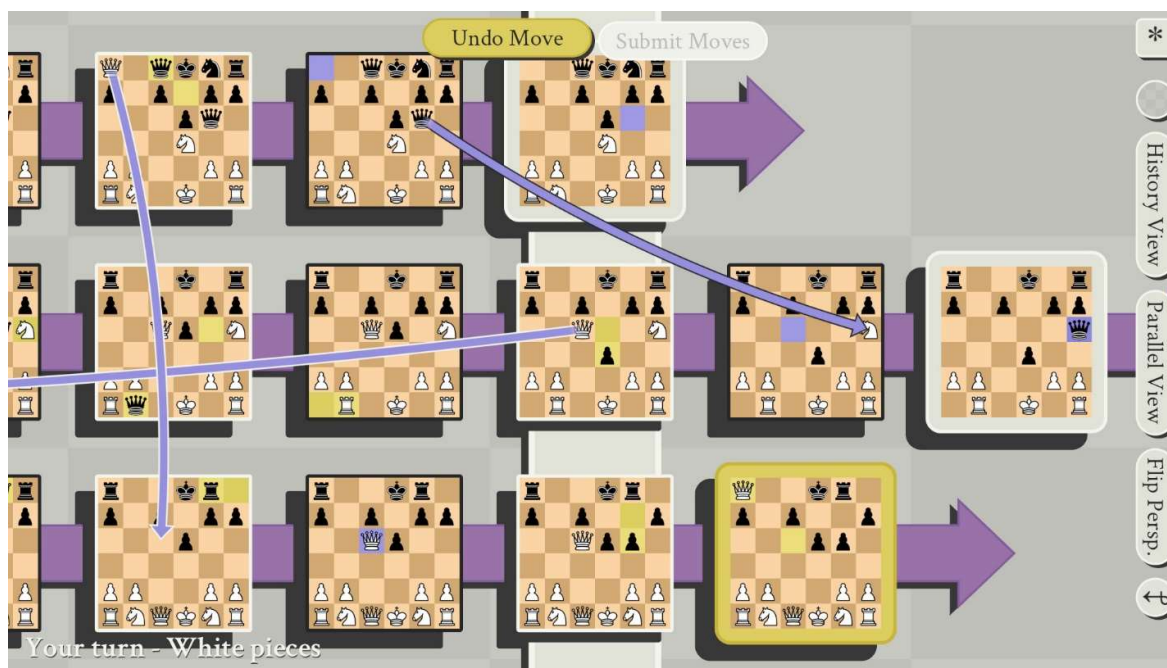


Фигура 1.1. No Man's Sky

Един от любимите ни жанрове в сферата на видеоигрите е научната фантастика или т.нар sci-fi игри, които отвеждат играчите в близко или далечно бъдеще, където те създават въображаеми технологии и

инженерни изобретения и развиват своите философски идеи. Точно както във филмите и телевизията, жанрът научна фантастика в игрите играе специална роля в поп културата. Проучване между американските тинейджъри показва, че 99% от момчетата и 94% от момичетата играят видеоигри. Прихода от продажби на видеоигри в САЩ за 2018г надхвърля 2 милиарда долара (Granic et al, 2018).

1.1.2 Игри с логическо мислене

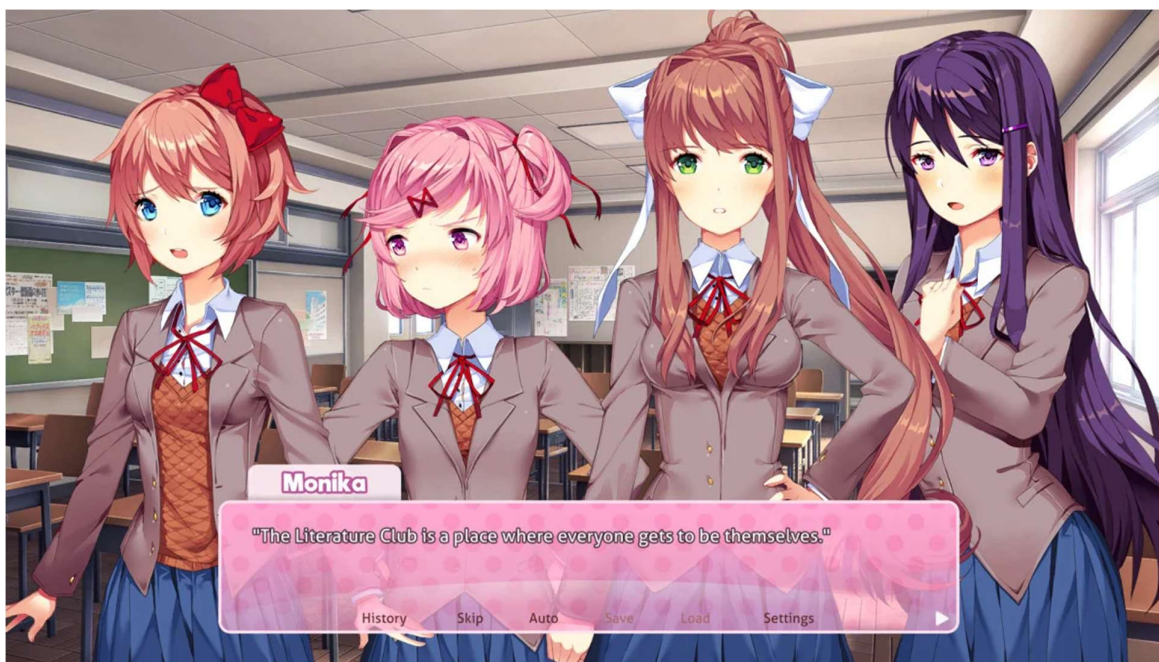


Фигура 1.2. 5D Chess With Multiverse Time Travel

Такива игри осигуряват забавна и социална форма на забавление, като едновременно с това стимулират ума и тялото. В едно проучване, проведено от Института за човешко развитие Макс Планк и

Университетския медицински институт в Берлин, изследователите твърдят, че ангажирането на ума със логически свързана последователност на действия в sci-fi игрите увеличава сивото вещество и помага за усъвършенстване на заучените и закрепени умения. Играта на видеоигри пряко засяга и въздейства върху областите на мозъка, отговорни за паметта, пространствената ориентация, организацията на информацията и фините двигателни умения.

1.1.3 Игри с фокусиран социален аспект



Фигура 1.3. Doki Doki Literature Club!

Друго проучване от 2012 г., публикувано в American Journal of Preventive Medicine, дори посочва, че видеоигрите могат да бъдат добри за

облекчаване на болката, като също така намаляват тревожността и хроничната болка от медицински процедури. Докато друго проучване сочи, че 3D видео игрите увеличават капацитета на паметта и осигуряват на мозъка значима стимулация. Такива игри могат да бъдат не само логически, но и социални, симулиращи ситуации и действия в истинския живот с цел развлечение или себеопознаване на потребителя.

1.2. Основни технологии за реализиране на уеб игри

1.2.1 Phaser Game Engine



Фигура 1.4. Phaser

Phaser е софтуерна рамка за 2D игри, използвана за създаване на HTML5 игри за настолни и мобилни устройства. Това е безплатен софтуер, разработен от Photon Storm. Phaser използва рендера на Canvas и WebGL и може автоматично да превключва между тях въз основа на поддръжката на браузъра. Специално е създадена да се възползва от

възможностите на модерните браузъри - и за компютри, и за телефони. Използва библиотеката Pixi.js за изобразяване as. Phaser може да работи във всеки уеб браузър, който поддържа елемента canvas. Игрите, направени с phaser, се разработват или на JavaScript, или на TypeScript. Необходим е уеб сървър за зареждане на ресурси като изображения, аудио и други файлове с игри, тъй като браузърите изискват уеб страниците да имат достъп до файлове само от същия произход.

Phaser може да бъде рендиран или в WebGL, или в Canvas елемента, с опция за използване на WebGL, ако браузърът го поддържа, или ако устройство не го поддържа, ще се върне обратно към Canvas.

Phaser се доставя с 3 физически системи: Arcade Physics, Ninja Physics и P2.JS.

Анимацията може да се направи във Phaser чрез зареждане на spritesheet, texture atlas и създаване на анимационна последователност.

Уеб аудио и HTML5 аудио могат да се използват за възпроизвеждане на звук във Phaser.

1.2.2 HTML - HyperText Markup Language



Фигура 1.5. HTML

HTML (HyperText Markup Language) е най-основният градивен елемент на мрежата. Той определя значението и структурата на уеб съдържанието. Други технологии освен HTML обикновено се използват за описване на външния вид/презентацията (CSS) или функционалността/поведението на уеб страницата (JavaScript).

„Хипертекст“ се отнася до връзки, които свързват уеб страници една с друга, или в рамките на един уебсайт, или между уебсайтове. Връзките са основен аспект на мрежата. Като качвате съдържание в Интернет и го свързвате със страници, създадени от други хора, вие ставате активен участник в World Wide Web.

HTML елемент се отделя от друг текст в документ чрез "тагове", които се състоят от името на елемента, заобиколено от "<" и ">". Името на елемент вътре в маркер не е чувствително към малки и големи букви. Тоест може да се пише с главни, малки букви или смесено. Например, етикетът <title> може да бъде написан като <Title>, <TITLE> или по друг начин.

1.2.3 CSS - Cascading Style Sheets



Фигура 1.6. CSS

Каскадните стилови таблици (CSS) е език за стилове, използван за описване на представянето на документ, написан на HTML или XML (включително XML диалекти като SVG, MathML или XHTML). CSS описва как елементите трябва да бъдат изобразени на екран, на хартия, в реч или на друга медия.

CSS е сред основните езици на отворената мрежа и е стандартизиран в уеб браузърите според спецификациите на W3C. Преди това разработването на различни части от спецификацията на CSS се извършваше синхронно, което позволяваше версия на най-новите препоръки. Може би сте чували за CSS1, CSS2.1, CSS3. Въпреки това, CSS4 никога не е станал официална версия.

1.2.4 Базы данни – SQL и MySQL



Фигура 1.7. MySQL

Базите данни играят жизненоважна роля в дизайна и разработването на игри. Те съхраняват данни за играчите, състоянието на играта, информация за производителността и поддържат среди, в които

работят екипите на разработчиците. Без добра база данни игрите не могат да функционират правилно.

SQL е специфичен за домейн език, което означава, че може да се използва само за игра с релационни бази данни. SQL се използва при обработка на структурирани данни – такива със специфична структура / формат. Може да се съхранява в таблици като формат или плоски файлове като CSV и TSV. Това е език, използван за търсене на таблични данни. Това е ANSI стандартен език, използван за манипулиране, съхраняване и достъп до данни в база данни.

Това е стандартизиран език за заявки за обработка на данни, съхранявани в RDBMS (система за управление на релационни бази данни).

Разработена от Oracle, MySQL е най-популярната в света база данни с отворен код. Работи със следните сървърни операционни системи: FreeBSD, Linux, OS X, Solaris и Windows. Това е система за управление на релационна база данни (RDBMS), базирана на специфичния за домейн език за програмиране Structured Query Language (SQL). За разлика от Firebase, MySQL е база данни с отворен код. Потребителите могат да разгръщат MySQL в облака или на място.

Като RDMS MySQL класифицира данните в различни таблици въз основа на свързани типове данни. Програмистите използват SQL за трансформиране и извличане на данните от RDMS. Когато е внедрен в операционна система, MySQL управлява потребителите, достъпа до мрежата и други компоненти на компютърна система за съхранение.

1.2.5 Базы данни – NoSQL и MongoDB



Фигура 1.8. MongoDB

Базата данни NoSQL (първоначално се отнася до "не-SQL" или "нерелационни") база данни осигурява механизъм за съхранение и извличане на данни, който е моделиран със средства, различни от табличните релации, използвани в релационните бази данни. Такива бази данни съществуват от края на 60-те години на миналия век, но името "NoSQL" е измислено едва в началото на 21-ви век, задействано от нуждите на компаниите Web 2.0. NoSQL бази данни се използват все по-често в големи данни и уеб приложения в реално време. NoSQL системите

също понякога се наричат Не само SQL, за да се подчертае, че те могат да поддържат подобни на SQL езици за заявки или да седят редом с SQL бази данни в полиглот-устойчиви архитектури.

С този нов модел, можем да създаваме много по – универсални хранилища на данни, в повечето случай като JSON обекти, позволявайки лесен достъп и промяна от множество различни Driver адаптери (както и Driver-и за MySQL), който позволяват директна местна употреба в програмните езици.

1.2.6 Javascript



Фигура 1.9. Javascript

JavaScript (JS) е лек, интерпретиран или компилиран език за програмиране с първокласни функции. Въпреки, че е най-известен като скриптов език за уеб страници, много среди, които не са браузъри, също го използват, като Node.js, Apache CouchDB и Adobe Acrobat. JavaScript е базиран на прототип, многопарадигмен, еднонишков, динамичен език, поддържащ обектно-ориентирани, императивни и декларативни (например функционално програмиране) стилове.

JavaScript работи от страна на клиента в мрежата, което може да се използва за проектиране/програмиране как се държат уеб страниците при възникване на събитие. JavaScript е лесен за научаване, а също и мощен скриптов език, широко използван за контролиране на поведението на уеб страниците.

JavaScript е динамичен скриптов език, поддържащ изграждане на обект, базиран на прототип. Основният синтаксис е умишлено подобен на Java и C++, за да се намали броят на новите концепции, необходими за изучаване на езика. Езиковите конструкции, като „if“ оператори, цикли „for“ и „while“, както и блоковете „catch“ функционират по същия начин, както в тези езици. JavaScript може да функционира както като процедурен, така и като обектно-ориентиран език. Обектите се създават програмно в JavaScript, чрез прикачване на методи и свойства към иначе празни обекти по време на изпълнение, за разлика от дефинициите на синтактичния клас, често срещани в компилираните езици като C++ и Java. След като обектът е конструиран, той може да се използва като чертеж (или прототип) за създаване на подобни обекти.

Динамичните възможности на JavaScript включват изграждане на обекти по време на изпълнение, списъци с променливи параметри, функционални променливи, динамично създаване на скрипт (чрез eval), интроспекция на обект (чрез за ... in) и възстановяване на изходния код (програмите на JavaScript могат да декомпилират функционални тела обратно в техния изходен текст).

1.2.7 Socket.IO



Фигура 1.10. Socket.IO

Socket.IO е базирана на събития JavaScript библиотека за уеб приложения в реално време. Той позволява двупосочна комуникация в реално време между уеб клиенти и сървъри. Той има две части: библиотека от страна на клиента, която работи в браузъра, и библиотека от страна на сървъра за Node.js. И двата компонента имат почти идентичен API.

Socket.IO използва предимно протокола WebSocket с допитване като резервна опция, като същевременно предоставя същия интерфейс. Въпреки че може да се използва просто като обвивка за WebSockets, той предоставя много повече функции, включително излъчване към множество гнезда, съхраняване на данни, свързани с всеки клиент, и

асинхронен I/O. Може да се инсталира, пакетира и разгърва с Node Package Manager (NPM).

1.2.8 Spring Framework



Фигура 1.11. Spring

Spring Framework предоставя изчерпателен модел за програмиране и конфигуриране за съвременни Java-базирани приложения - на всякакъв вид платформа за внедряване.

Ключов елемент на Spring е инфраструктурната поддръжка на ниво приложение: Spring се фокусира върху „плавността“ на приложения, така че екипите да могат да се съсредоточат върху логиката на ниво приложение, без ненужни връзки със специфични среди за внедряване.

Spring е най-популярната рамка за разработка на приложения за Java. Милиони разработчици по целия свят използват Spring Framework, за да

създават високопроизводителен, лесен за тестване и повторно използваем код.

Spring framework е Java платформа с отворен код. Първоначално е написан от Род Джонсън и за първи път е пуснат под лиценза Apache 2.0 през юни 2003 г.

Основните характеристики на Spring Framework могат да се използват при разработването на всяко Java приложение, но има разширения за изграждане на уеб приложения върху платформата Java EE. Spring Framework има за цел да направи разработката на J2EE по-лесна за използване и насърчава добрите практики за програмиране чрез активиране на базиран на POJO модел на програмиране.

POJO е съкращение от „Plain Old Java Object“, което означава че този модел представлява най – елементарния и ефективен метод от страна на памет и производителност за използване на обектите в Java. POJO моделът също е адаптиран в много архитектури с цел лесна интеграция и заимстване през изградените системи.

Spring е организирана по модулен начин. Въпреки че броят на пакетите и класовете е значителен, трябва да се тревожите само за тези, от които се нуждаете, и може да игнорирате останалите.

Тестването на приложение, написано със Spring, е лесно, защото зависим от средата код е преместен в тази рамка. Освен това, чрез използване на JavaBeanstyle POJO, става по-лесно да се използва инжектиране на зависимост за инжектиране на тестови данни.

Spring разработчиците са изградили автоматично конфигуриращи се приспособления за тестване на приложението както на Frontend така и на Backend. Най – големите библиотеки синхронизирани със Spring са Selenium и JUnit, като за първата е нужен Gecko Driver или подобен за изпълнението на тестовете.

Уеб рамката на Spring е добре проектираната уеб MVC рамка, която предоставя чудесна алтернатива на уеб рамки като Struts или други прекалено проектирани или по-малко популярни уеб рамки.

MVC представлява Model-View-Controller структура която е широко приета като стандарт в света на уеб framework-овете. Тя свързва нужните характеристики зададени от сървъра, за да може да променя динамично съдържанието на страницата. Всеки „view“ е HTML страница, “model” е изградената структура която се подава през “request-response” моделът, а “controller” е сървърната част, която манипулира информацията за модела.

Spring предоставя удобен API за превеждане на специфични за технологията изключения (изхвърлени от JDBC, Hibernate или JDO, например) в последователни, непроверени изключения.

Един от ключовите компоненти на Spring е рамката за обектно ориентирано програмиране (AOP). Функциите, които обхващат множество точки на приложение, се наричат междусекторни проблеми и тези междусекторни проблеми са концептуално отделени от бизнес логиката на приложението. Има различни общи добри примери за аспекти, включително регистриране, декларативни транзакции, сигурност, кеширане и др.

Какво е Micro Service?

Micro Service е архитектура, която позволява на разработчиците да разработват и внедряват услуги независимо. Всяка изпълнявана услуга има свой собствен процес и това постига олекотения модел за поддръжка на бизнес приложения.

Предимства

Микро услугите предлагат следните предимства на своите разработчици –

- Лесно разгръщане
- Лесна мащабируемост
- Съвместим с контейнери

- Минимална конфигурация
- По-малко време за производство

1.2.9 Spring Boot



Фигура 1.12. Spring Boot

Spring Boot предоставя добра платформа за Java разработчиците за разработване на самостоятелно и производствено Spring приложение,

което можете просто да бъде стартирано. Можете да се започне с минимални конфигурации, без да е необходима цялостна конфигурация на Spring.

Spring Boot предлага следните предимства на своите разработчици:

- Лесни за разбиране и разработване на приложения
- Увеличава производителността
- Намалява времето за разработка
- Избягване на сложна XML конфигурация през Spring
- Разработване на готови за производство Spring приложения по по-лесен начин
- Намалява времето за разработка и да стартирате приложението самостоятелно

1.3. Подобни съществуващи разработки на игри

1.3.1 Factorio

Factorio е една от най-популярните sandbox sci-fi игри, в която играчът изгражда и поддържа фабрики, копае ресурси, проучва технологии, изгражда инфраструктура, автоматизира производството и се бори с врагове. Играта предоставя възможности, за проектиране на фабрики, чрез комбиниране на прости елементи в гениални структури, а

играчът развива управленски умения, за да я поддържа да работи, и я предпазва от същества, които искат да навредят.

На най-ниските нива играчът цепи дървета, копасте руди и да изработва механични оръжия и транспортни ремъци на ръка, но за кратко време може да се превърне в индустриална електроцентрала с огромни слънчеви полета, рафиниране и крекинг на нефт, производство и внедряване на строителни и логистични работи, всичко произведено със собствени ресурси. Тази тежка експлоатация на ресурсите на планетата обаче не се харесва на местните, и ако играча нарушава замърсява околната среда със своите дейности, ще трябва да е готов да защитава себе си и своята машинна империя.

Играчът има възможност да обединява усилията си с други играчи в кооперативния мултиплейър, създавайки огромни фабрики, да работи съвместно и да делегира задачи със другите играчи.

Играта е с отворен код: добавянето на модификации - от малки настройки и помощни модификации до завършени промени в играта, основната поддръжка на Factorio Modding, позволява на създателите на съдържание от цял свят да проектират интересни и иновативни функции. Докато основният геймплей е под формата на сценария за безплатна игра, има

редица интересни предизвикателства под формата на пакета сценарии, достъпен като безплатно DLC.

Алфа версията на играта излиза през 2016г., а версия 1.0 официално е пусната през август 2020г. “Dwarf Fortress”, “Transport Tycoon”, “The Civilization series”, “SpaceChem” са само част от игрите, вдъхновили създателите на Factorio. Но нейният създател, Майкъл Коварик, казва че идеата за Factorio идва от играта “Minecraft”, и по-специално от нейния „Buildcraft“ мод.



Фигура 1.13. Factorio в действие

1.3.2 Minecraft

Minecraft е 3D игра с отворен код, първоначално създадена от Маркус "Notch" Персон. Поддържа се от Mojang Studios, част от Xbox Game Studios, която от своя страна е част от Microsoft.

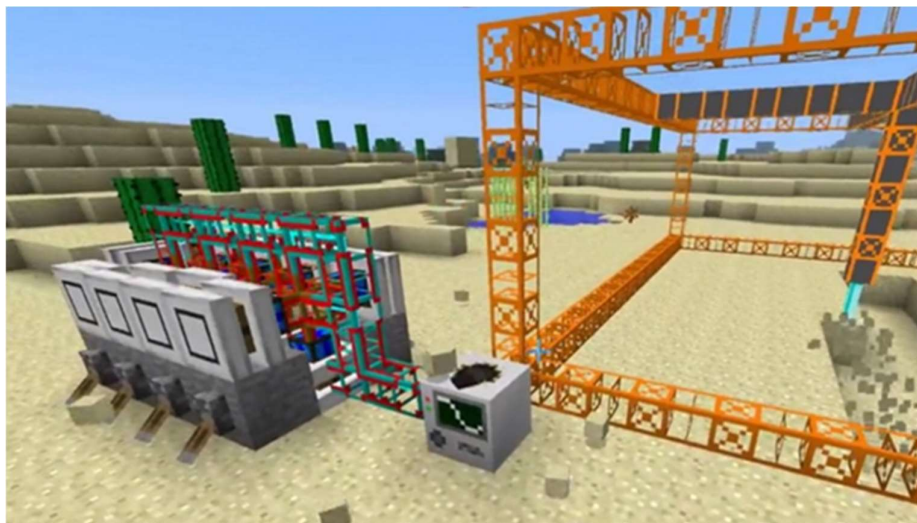
От създаването си Minecraft е разработен почти изключително от Notch, докато Йенс „Джеб“ Бергенстен започва да работи с него и оттогава става ръководител на неговото развитие. Той включва музика от Даниел "C418" Розенфелд, Куми Тамиока и Лена Рейн и графики на Кристофър Зетерстранд. Първоначалната версия на играта сега е известно като Minecraft Classic и излиза на 17 май 2009 г., а пълната версия на играта е пусната на 18 ноември 2011 г. След излизането си, Minecraft се разшири до мобилни устройства и конзоли. На 6 ноември 2014 г. Minecraft и всички активи на Mojang Studios бяха придобити от Microsoft за 2,5 милиарда долара.

Minecraft се фокусира върху това да позволи на играча да изследва, да взаимодейства и да променя динамично генерирана карта, съставена от блокове с размер един кубичен метър. В допълнение към блокове, околната среда включва растения, мобове и предмети. Някои дейности в играта включват изграждане, копаене на руда, борба с враждебни тълпи и изработване на нови блокове и инструменти чрез събиране на различни

ресурси, открити в играта. Отвореният модел на играта позволява на играчите да създават структури, творения и произведения на изкуството на различни конкурентни или съвместни мултиплейър сървъри или техните карти за един играч. Други характеристики включват схеми на Redstone за логически изчисления и дистанционни действия, колички и писти, както и мистериозен подземен свят, наречен Nether. Определена, но напълно незадължителна цел на играта е да пътуването до измерение, наречено Краят, и да победа над дракона на Ender.

BuildCraft е модификация на Minecraft, който използва машини за копаене на ресурси, изработване на предмети и сортиране на ресурси. Той също така разполага с машини, които могат автоматично да изграждат структури въз основа на чертежи. Модът също така включва тръби за транспортиране на предмети, течности и мощност.

BuildCraft беше оригиналният мод за представяне на Minecraft Joules (MJ). Машини като Quarry, Pump и Builder могат да се захранват с помощта на един от трите двигателя, които BuildCraft добавя. BuildCraft също така въвежда две нови течности, масло и гориво, за да осигури повишена мощност на енергия. BuildCraft Wrench е нов инструмент за преориентиране на машини и тръбопроводни системи.



Фигура 1.14. Quarry копае използвайки енергия (MJ)

1.3.3 Satisfactory

Satisfactory е доста по - модерна игра от Coffee Stain Studios. Тя е игра на управление на фабрики и експлоатация на планетата. Като пионер, работещ за FICSIT Inc., вие сте хвърлен на извънземна планета с шепа инструменти и трябва да съберете природните ресурси на планетата, за да построите все по-сложни фабрики за автоматизиране на всичките ви нужди от ресурси. След като сте настроени, ще дойде време да изградите космическия асансьор и да започнете сглобяването на проекта, доставяйки FICSIT с все по-многобройни и сложни компоненти за техните неизвестни цели.

Основният фокус на играта е в изграждането на сгради и свързването им заедно с конвейерни ленти, така че вашата фабрика да може да се справи с целия процес на изграждане на артикули вместо вас, включително добив, транспорт, доставка и генериране на енергия, за да поддържа работата си. По-напредналите пионери ще регулират точно тактовата честота на своите сгради, за да увеличат максимално ефективността.



Фигура 1.15. Връзки от конвейери



Фигура 1.16. Тръби пренасящи вода

ВТОРА ГЛАВА

ПРОЕКТИРАНЕ НА СТРУКТУРАТА НА УЕБ 2D ИГРА

2.1. Функционални изисквания към реализирането

- Да има сървър (Backend)
- Да има бази данни от двата типа SQL и NoSQL, в този случай MySQL и MongoDB
- Backend да поддържа логин, регистрация и изход
- Backend да поддържа грешки при невалидни операции като логин с грешно име и парола или регистрация с вече съществуващ мейл
- Да се създава сървър на Socket.IO при пускане на Backend
- Socket.IO сървърът да свързва сесията на потребителя с Backend-а и да се поддържа двупосочна комуникация
- Да има Frontend с логин, регистрационна и целева страница която пренасочва и зарежда потребителя към играта му

- Да има система за поставяне на обекти
- Да има система за пренасяне на предмети и течности
- Да има автономни роботи

2.2. Избор на езици

2.2.1 Backend

Що се отнася до бекенд технологиите, въпросът какво да бъде използвано е малко по-труден заради по-големия брой варианти като NodeJS - Express, RubyOnRails, Laravel, Flask и много други. Spring Boot е избрано за този проект по следните причини:

- ✓ Разработван от много време – Spring Boot е базиран на Spring Framework, който сам по себе си е разработван над 15 години като алтернатива на JEE стакът.
- ✓ Стабилен – Основните модули на Spring екосистемите са стабилни за дълго време и повечето промени лесно могат да бъдат отменени
- ✓ JVM - Spring Boot е базиран на Виртуалната машина на Java, което осигурява по-голяма защита и независимост от използваната платформа.

- ✓ Приятен за разработка – Spring Boot предоставя доста удобства за да улесни разработката, с което привлича доста софтуеристи.

За логиката и на играта е избран Java, който единствен позволява сравнение с другите обектно-ориентирани езици, лесен начин за развиване на скаларни абстракции, синхронизиране и манипулиране на нишки. Това води до промяна на фокуса на разработчика от спецификите на основната технология която използва до само и единствено на максималното и плодовитото развитие на идеите му.

2.2.2 Frontend

HTML, CSS и Javascript. Първите два, защото са в основата на всяко уеб приложение и в днешно време е невъзможно да се създаде структурирана система от взаимно изпълняващи се събития без тях. CSS е заменен с framework библиотеки от трета страна като Bootstrap, Tailwind CSS и т.н., но това ненужно разширява апликацията, заради добавянето на огромни пакети от функции. В повечето малки приложения, не отделящи фокус

на дизайн на сайта, като текущата дипломна работа, може да се използва чист CSS. Javascript е избран като скриптов език и като език с който се създава и конфигурира играта и нейния Phaser игров двигател. Гъвкавостта на JavaScript е най-подходяща за средно напреднали разработчици. Езикът просто помага да се свършат нещата, като позволява на разработчика да се съсредоточи върху решаването на проблема.

Разработчиците могат да използват комбинация от плъгини и свои собствени кодови фрагменти, за да накарат приложението да работи. JavaScript има мениджъри на пакети, които са лесни за използване и интуитивни. Това е от решаващо значение, тъй като мениджърите на пакети позволяват на разработчиците да споделят своя код между екипи и спестяват много време. Най – използваният NPM е мениджърът на пакети по подразбиране за средата на изпълнение на JavaScript Node.js. Състои се от клиент на командния ред, наричан още npm, и онлайн база данни с публични и платени частни пакети, наречена регистър на npm. Достъпът до регистъра се осъществява чрез клиента, а наличните пакети могат да бъдат преглеждани и търсени чрез уебсайта на npm. Мениджърът на

пакети и регистърът се управляват от npm, Inc. Javascript е също доста конкурентно способен спрямо производителността си, която в някои случаи дори надминава тази на C++.

2.3. Избор за програмни средства

2.3.1 Разработване

Средата за разработване е IntelliJ IDEA. IntelliJ IDEA е интегрирана среда за разработка (IDE), написана на Java за разработване на компютърен софтуер. Той е разработен от JetBrains (известен преди като IntelliJ) и е достъпен като лицензирано издание на общността на Apache 2 и в собствено търговско издание. И двете могат да се използват за търговско развитие.

Това е една от най – разпространените среди за програмиране на Java и включва всички нужни инструменти за бързо и комфортно развитие на всякакъв вид приложения. За разлика обаче от Eclipse, който е open-source, IntelliJ IDEA не е, което означава че има по – малко персонализирани плъгини създадени от личните потребители, за да променят неща които преди това не са могли само със средата. Освен това то има

вграден пакетен мениджър Maven, без нужда да бъде теглен и инсталиран отделно, който позволява за експресното пакетиране и изпълняване на целия Java проект само чрез няколко команди.

2.4. Графични ресурси

2.4.1 Ресурси за играта

Всички ресурси за играта са взети от играта “Factorio”. Те предлагат 2 различни версии в формата на spritesheet-ове[3]. Едната е нормално, другата високо качество. Всеки спрайт е с размер около 120x120, но е преработен чрез Photoshop за да постигне 30x30 или колкото са му нужни размери за да бъде ползваем в играта.

2.5. Структура на базата данни

2.5.1 MySQL

MySQL базата от данни се състои от 2 таблици: „users“ и „email_tokens“. Users таблицата пази данните за потребителя като неговото уникално id, потребителско име, парола, имейл и състояние на валидация. Ако то не е логическо да, достъп към сървърът е отказан. Потребител може да си активира акаунта,

като след регистрацията отиде на имейл адреса си и натисне активационна линк в рамките на 24 часа. В таблицата за токени се запазват токените за активация на потребителите, чрез които се изпраща GET заявка при натискане на активационния линк. В таблицата е запазена връзка с id-то на потребителя чиито е токена, активирайки го чрез нея.

2.5.2 MongoDB

MongoDB базата данни е използвана като съхранение на данните на потребителя. На всяка заявка за промяна/четене от нея, се трансформира от Driver-а към MongoDB заявка и се изпълнява зададената операция върху колекция. Колекцията запазва всеки един обект който е нужен за играта, като документите запазени вътре могат да имат различни характеристики, без да афектират функциите на базата от данни.

2.6. Описание на алгоритъма на действие

2.6.1. Backend

Backend цели да зареди всички нужни класове в паметта на Spring което да „хлабав куплунг“ връзка между всеки обект. Първоначално се зарежда контролер който излага валидните

връзки към сървъра чрез HTTP заявки. При получаване на заявка, сървърът зарежда „model“ обектът във “view” HTML темплейт страницата и я връща обратно на потребителя. Тя динамично зарежда всички характеристики от модел чрез Thymeleaf. Сървърът също зарежда и конфигурира Socket.IO сървър който също излага двупосочни събития между себе си и свързващите се клиенти, винаги една когато е осъществена успешна връзка и една когато връзката се прекъсне, заедно с останалите зададени от разработчика. Освен това Backend има за задача да зарежда игровата сесия в паметта заедно със всичките обекти който са динамични (променящи се с времето). Това става чрез множество заявки към MongoDB базата от данни. Потребителят може да се логне, при опит се изпращат заявки към MySQL базата от данни за проверка на съществуващ потребител или изтекъл срока за потвърждаване на имейла.

2.6.2 Frontend

Frontend се зарежда с текущия му зададен модел от Thymeleaf, ако има отразявайки грешки при жизнения цикъл на сесията на потребителя, без да бъде натрапчиво от дизайнова

перспектива. Елементите са текстови полета, който са пряко свързани с обекта от модела на който отговарят. Те са във форма, която завършва с бутони пращащи POST заявки на сървърва, който съобразимо с техния тип, препращат потребителя към друга страница или изпълняват основната си функция, при положение без наложени проблеми отразяващи се като грешки в страницата, позволявайки на потребителя да ги види и промени операциите си базирани върху тях. При зареждането на играта се осъществява пряка връзка между сървърва и клиента чрез Socket.IO, позволявайки за реално времеви действия в играта и отразяването им първоначално в сървърва, и посредством връзката чрез сокет, получаване на информация за промяната на обектите така че да се отразят и при клиентската страна на приложението. При всяко влизане се зареждат обектите в играта отново по пътя на сокета след потвърждени за успешна връзка със сървърва.

ТРЕТА ГЛАВА

Програмна реализация на уеб 2D игра

3.1. Сървър

3.1.1 Стартиране и конфигуриране на сървър

```
package org.elsys.ufg;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class UfgApplication {
    public static void main(String[] args) {
        SpringApplication.run(UfgApplication.class, args);
    }
}
```

Фигура 3.1. UfgApplication.java

Стартирането на сървъра става с извикването на основната за Spring Boot `run()` функция, това стартира централният `Http` в който се запазват инстанциите на „Bean“[4] – ове и се зареждат конфигурации, назовани с `@Configuration` анотацията[5].


```
package org.elsys.ufg;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler(...pathPatterns: "/assets/**").addResourceLocations("/WEB-INF/assets/");
    }
}
```

Фигура 3.2. WebConfig.java

Тази конфигурация създава пътищата, по които могат да се правят GET заявки за достъп на графичните ресурси на играта.

```
@Controller
public class MainController{
    @Autowired
    private UserRepository userRepository;

    @Autowired
    private EmailTokenRepository emailTokenRepository;

    @Autowired
    private JavaMailSender emailSender;

    @Autowired
    private RedirectHandler redirectHandler;

    @Autowired
    private CookieService cookieService;
```

Фигура 3.3. MainController.java

В този контролер дефинираме всички останали заявки, към които сървърът ще отговаря. `@Autowired`[6] анотацията ни позволява да вземем нужния генериран клас от `Нар-а`, елиминирайки нуждата да го създаваме наново при всяка дефиниция.

```
@RequestMapping(value = "/register", method = RequestMethod.GET)
public String getRegister(@ModelAttribute("user") User user, HttpServletRequest request, HttpServletResponse response){
    return "register";
}
```

Фигура 3.4. Влизане в страницата за регистрация

The image shows a registration form with a dark background. At the top, the word "Register" is written in a large, bold, white font. Below it, there are four input fields, each with a label in a light gray font: "USERNAME", "EMAIL", "PASSWORD", and "REPEAT PASSWORD". Each label is positioned to the left of its corresponding input field. At the bottom of the form, there is a green button with the word "REGISTER" in white capital letters.

Фигура 3.5. Форма за регистрация

Показаното на 3.5. представя връщането на темплеит “register.html” в който е вкаран User обект.

```
@RequestMapping(value = "/register", method = RequestMethod.POST)
public String postRegister(@ModelAttribute("user") User user, @RequestParam Map<String, String> params)
    String redirect = redirectHandler.redirection(params);

    if(redirect != null){
        return redirect;
    }

    user.validateRegister(userRepository);

    userRepository.save(user);

    EmailToken emailToken = new EmailToken(emailTokenRepository, user.getId());

    emailTokenRepository.save(emailToken);

    MimeMessage mailMessage = emailSender.createMimeMessage();
    MimeMessageHelper messageHelper = new MimeMessageHelper(mailMessage, multipart: true, encoding: "UTF-8")
    messageHelper.setFrom("officialufgggame@gmail.com");
    messageHelper.setTo(user.getEmail());
    messageHelper.setText( text: "Press Link to Confirm Email: <a href=\"http://localhost:8080/confirmmai
    emailSender.send(mailMessage);

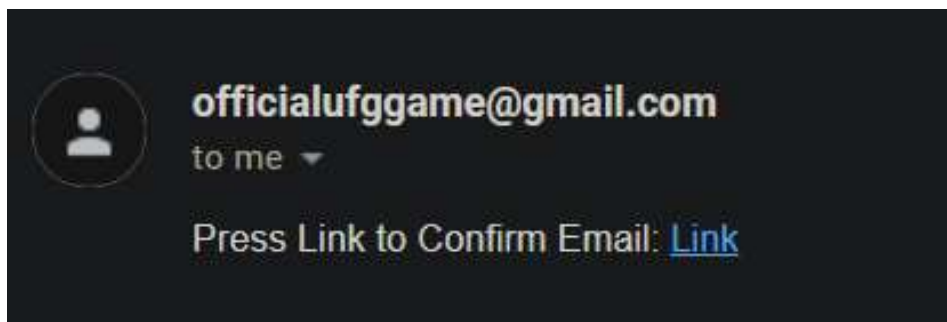
    return "sent_confirmation_mail";
}
```

Фигура 3.5. Регистриране на потребител

Регистрирането на потребител се осъществява чрез използването на предишно вкарания User обект който съдържа полетата: username, password, passwordRepeat, email и activated. След това използваме валидиращи методи за да проверим дали вкараните данни са правилни и може да се създаде нов акаунт. При успех, потребителя се препраща към страница индикираща му да си провери мейла където може да си потвърди акаунта.

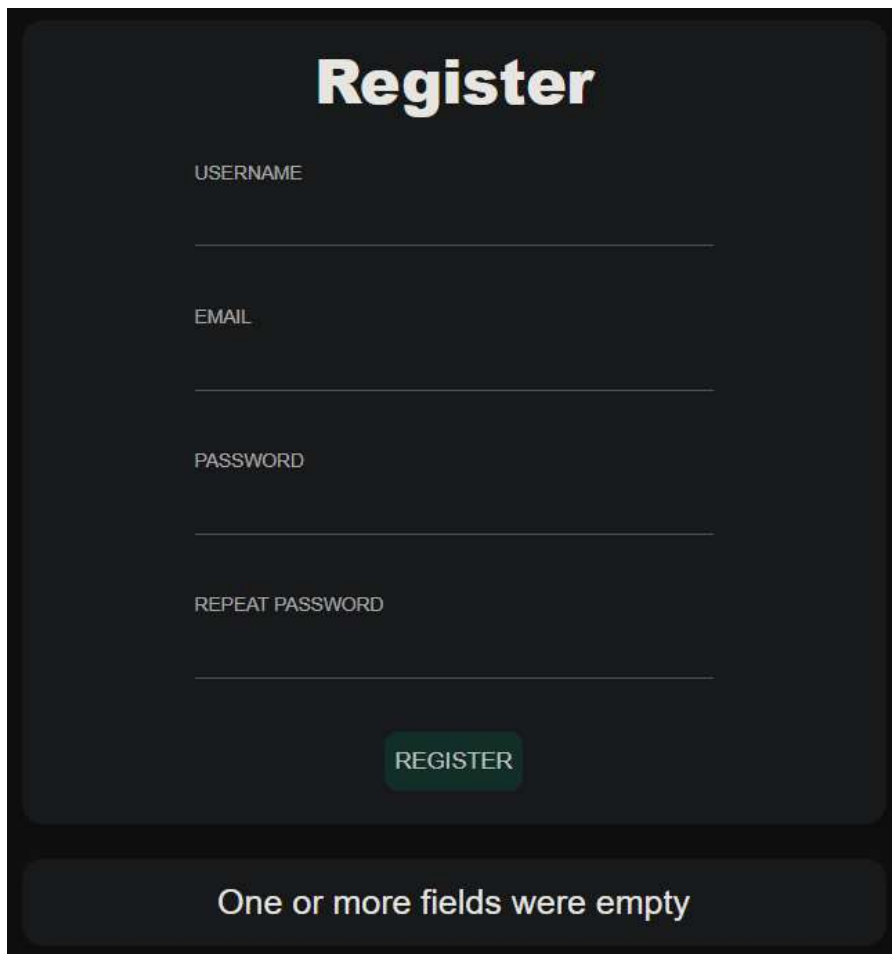


Фигура 3.6. Индикираща страница за потвърждаване на мейл



Фигура 3.7. Получен мейл от сървъра

В фигура 3.5. сървърът изпраща мейл чрез MimeMessage класът, подпомогнат чрез създаването на мейла от emailSender който инициализира имейла. Нужните данни са му зададени и бива изпратен на имейла на потребителя.



Register

USERNAME

EMAIL

PASSWORD

REPEAT PASSWORD

REGISTER

One or more fields were empty

Фигура 3.8. Регистрация с грешка (Едно или повече полета са били оставени празни при регистриране)

```
@RequestMapping(value = "/login", method = RequestMethod.GET)
public String getLogin(@ModelAttribute("user") User user) {
    return "login";
}

@RequestMapping(value = "/login", method = RequestMethod.POST)
public String postLogin(@ModelAttribute("user") User user, @RequestParam Map<String, String> params
    String redirect = redirectHandler.redirection(params);

    if(redirect != null){
        return redirect;
    }

    user.validateLogin(userRepository);

    cookieService.login(user, response);
    return "redirect:/home";
}
```

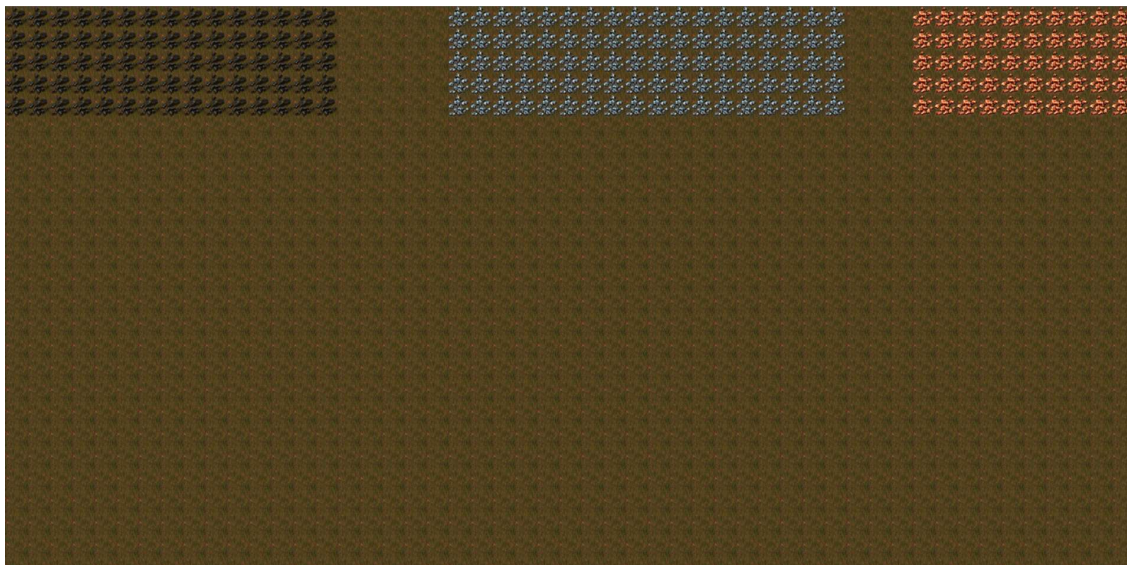
Фигура 3.9. Логин връзки

При GET заявка на логин получаваме темплейта за логин, а при POST както при регистрацията проверяваме за валидност на данните. На GET заявка използваме cookieService сервизът който използва @Service анотацията. Този сервиз отговаря за контролирането на бисквитките на потребителя.

```
public void login(User user, HttpServletResponse response){
    create( name: "username", user.getUsername()).setMaxAge(Integer.MAX_VALUE).setPath("/").build(response);
    create( name: "password", user.getPassword()).setMaxAge(Integer.MAX_VALUE).setPath("/").build(response);
}
```

Фигура 3.10. Запазването на бисквитка

Когато потребител влезе за първи път, ако браузърът му позволява, се запазват локално бисквитки и при следващи влизания в сайта директно се препраща към играта.



Фигура 3.11. Играта

Начален екран към който потребителя е препратен при успешно валидиране на бисквитката. Бисквитката бива създадена чрез „Builder“ шаблонът. В края на всяка задаваща („set“) функция е върната референция към текущия обект, в този случай бисквитката, което позволява отново викане на други задаващи функции, и накрая се вика `build()` метода който добавя бисквитките към браузъра на потребителя чрез отговора на първоначалната заявка (`response`).


```
@Service
public class RedirectHandler {
    private List<String> redirectLinks;

    public RedirectHandler(){
        redirectLinks = List.of("login", "register");
    }

    public String redirection(Map<String, String> params){
        for (String link : redirectLinks) {
            if(params.get(link) != null){
                return "redirect:/" + link;
            }
        }

        return null;
    }
}
```

Фигура 3.12. RedirectHandler.java

Този клас е използван за препращане при натискане на различен бутон от основния във формите на страниците. За пример, в метода за логин страницата във фигура 3.9., когато влизаме проверяваме дали моделът съдържа зададено препращане като викаме “redirection” метода с параметрите.

```
@Repository
public interface UserRepository extends CrudRepository<User, Integer> {

    @Query("SELECT 1 FROM users WHERE username = :username AND password = :password LIMIT 1")
    Boolean existsByUsernameAndPassword(@Param("username") String username, @Param("password") String password);

    @Query("SELECT 1 FROM users WHERE username = :username LIMIT 1")
    Boolean existsByUsername(@Param("username") String username);

    @Query("SELECT 1 FROM users WHERE email = :email LIMIT 1")
    Boolean existsByEmail(@Param("email") String email);

    @Modifying
    @Query("UPDATE users SET activated = 1 WHERE id = :id LIMIT 1")
    void activateById(@Param("id") Integer id);

    @Modifying
    @Query("DELETE FROM users WHERE id = :id LIMIT 1")
    void deleteById(@Param("id") Integer id);

    @Query("SELECT activated FROM users WHERE username = :username LIMIT 1")
    Boolean getActivatedStateByUsername(@Param("username") String username);
}
```

Фигура 3.13. UserRepository.java

Използваме този интерфейс, за да пропагираме команди към MySQL базата от данни. Това се получава чрез интерпретаця на метода тук от името му или конфигурираната заявка от `@Query` анотацията. Тя се извършва от JDBC[7] драйвера.

```
@Repository
public interface EmailTokenRepository extends CrudRepository<EmailToken, Integer> {
    @Query("SELECT 1 FROM email_tokens WHERE token = :token LIMIT 1")
    Boolean existsByToken(@Param("token") String token);

    @Query("SELECT timestamp FROM email_tokens WHERE token = :token")
    String findTimestampByToken(@Param("token") String token);

    @Modifying
    @Query("DELETE FROM email_tokens WHERE token= :token")
    void deleteByToken(@Param("token") String token);

    @Query("SELECT user_id FROM email_tokens WHERE token = :token")
    Integer findUserIdByToken(@Param("token") String token);
}
```

Фигура 3.14. EmailTokenRepository.java

По – същия начин е създаден и функциониращ този клас, позволявайки динамична връзка със таблицата email_tokens.

```
@Controller
public class DefaultErrorController implements ErrorController {
    @RequestMapping("/error")
    public String handleError(HttpServletRequest request){
        int statusCode = (int) request.getAttribute(RequestDispatcher.ERROR_STATUS_CODE);

        if(statusCode == HttpStatus.NOT_FOUND.value()) {
            return "redirect:/home";
        }

        return "redirect:/home";
    }
}
```

Фигура 3.15. Общи грешки

При общи грешки на сървърната страна, се влиза в този контролер, който има за цел чрез статус кода на заявката, да

разбере как да реагира. Има проверка в случай че грешката е несъществуващ ресурс, която препраща към началната страница, чрез „redirect:/**“ синтаксиса, който използва директно контекста на приложението (в нашият случай localhost:8080/) и зададеният линк след това.

```
@Controller
@ControllerAdvice
public class ErrorController {

    @ExceptionHandler(PasswordRepeatPasswordMismatchException.class)
    public String passwordRepeatPasswordMismatchExceptionHandler(Model model, PasswordRepeatPasswordMismatchException exception){
        model.addAttribute( attributeName: "errorMessage", exception.getErrorMessage());
        model.addAttribute( attributeName: "user", new User());
        return exception.getPage();
    }

    @ExceptionHandler(EmptyInputException.class)
    public String emptyInputExceptionHandler(Model model, EmptyInputException exception){
        model.addAttribute( attributeName: "errorMessage", exception.getErrorMessage());
        model.addAttribute( attributeName: "user", new User());
        return exception.getPage();
    }

    @ExceptionHandler(UserDoesNotExistException.class)
    public String userDoesNotExistExceptionHandler(Model model, UserDoesNotExistException exception){
        model.addAttribute( attributeName: "errorMessage", exception.getErrorMessage());
        model.addAttribute( attributeName: "user", new User());
        return exception.getPage();
    }

    @ExceptionHandler(UsernameIsTakenException.class)
    public String usernameIsTakenExceptionHandler(Model model, UsernameIsTakenException exception){
        model.addAttribute( attributeName: "errorMessage", exception.getErrorMessage());
        model.addAttribute( attributeName: "user", new User());
        return exception.getPage();
    }

    @ExceptionHandler(EmailIsTakenException.class)
    public String emailIsTakenExceptionHandler(Model model, EmailIsTakenException exception){
        model.addAttribute( attributeName: "errorMessage", exception.getErrorMessage());
        model.addAttribute( attributeName: "user", new User());
        return exception.getPage();
    }
}
```

Фигура 3.16. Конфигурирани грешки

Както можем да конфигурираме хващане на exception с try – catch блок, можем да създадем контролер който хваща всеки throw на exception класовете, независимо къде са викнати в

програмата. Това позволява за лесно конфигуриране на различно поведение при различните exception-и.

```
@RequestMapping(value = "/confirmemail/{token}", method = RequestMethod.GET)
public String getConfirmEmail(@PathVariable("token") String token){
    if(emailTokenRepository.existsByToken(token) == null){
        return "email_confirmation_failure";
    }

    Instant start = Instant.parse(emailTokenRepository.findTimestampByToken(token));
    Instant end = Instant.now();

    Integer user_id = emailTokenRepository.findUserIdByToken(token);
    emailTokenRepository.deleteByToken(token);

    if(Duration.between(start, end).toHours() >= 24){
        userRepository.deleteById(user_id);
        return "email_confirmation_failure";
    }

    userRepository.activateById(user_id);

    return "email_confirmation_success";
}
```

Фигура 3.17. Потвърждаване на мейл

При проверката на мейла, се проверява дали не са изминали повече от 24 часа от изпращането на токена на мейла на потребителя. В такъв случай акаунта не е активиран и бива изтрит. Сравнението става с проверката на две точки от времето, под формата на “timestamp”, една сега и една взета от EmailTokenRepository от момента на създаване на токена. Ако времето е по – малко от 24 часа, акаунта бива активиран.

3.2 Сокет сървър

3.2.1 Стартиране на сокет сървъра

```
@Configuration
public class SocketIOServerConfiguration {
    private String host;
    private Integer port;

    @Bean
    public SocketIOServer socketIOServer() {
        host = "localhost";
        port = 3000;

        com.corundumstudio.socketio.Configuration configuration = new com.corundumstudio.socketio.Configuration();
        configuration.setHostname(host);
        configuration.setPort(port);
        return new SocketIOServer(configuration);
    }
}
```

Фигура 3.20. Конфигурация на сокет сървър

Този клас се използва за задаването на параметри на сокет сървърът. То включва и много други като време през което да пробва да се свърже отново ако се е прекъснала връзката. Всички се подават на вграденият интерфейс от библиотеката Netty-Socket.IO, която е имплементация от трета страна на официалната Socket.IO библиотека, преработена за Java, като тя първоначално е направена за Javascript.

```
socketIOServer.addConnectListener((client) -> {  
    clients.put(client.getSessionId().toString(), client);  
});  
  
socketIOServer.addDisconnectListener((client) -> {  
    String userId = client.getSessionId().toString();  
    gameService.stopGame(clientUsernames.get(userId));  
  
    clients.remove(userId);  
    clientUsernames.remove(userId);  
    client.disconnect();  
});  
  
socketIOServer.addEventListener( eventName: "username", String.class, (client, username, ackRequest) -> {  
    String userId = client.getSessionId().toString();  
    List<GameObject> gameObjects = gameStorageRepository.findGame(username);  
    gameService.addGame(username, gameStorageRepository);  
    gameService.loadGame(gameObjects, username);  
    client.sendEvent( s: "game", gameObjects);  
    clientUsernames.put(userId, username);  
});
```

Фигура 3.21. Слушатели

При създаването на сървъра се добавят слушатели, които се изпълняват, при извикването на тяхното събитие. На този принцип работи „username“ събитието, което получава името на даден потребител, потвърждавайки че той е готов да получи информация за играта, която е изпратена със `sendEvent()` метода. Всички игрови обекти се сериализират в JSON и биват изпратени през сокета към Frontend частта.


```

        socketIOServer.start();
    }

    @PreDestroy
    public void stop(){
        if(socketIOServer != null){
            socketIOServer.stop();
            socketIOServer = null;
        }
    }
}

```

Фигура 3.22. Жизнен цикъл на сокет сървър

Жизнения цикъл се изразява в стартирането му, подобно на нишките в Java, чрез `start()` вграден метод и спирането и освобождаването на памет, постигнато през `@PreDestroy` анотацията, която има за цел да бъде извикана при спирането на главният сървър.

3.3 Frontend

3.3.1 Влизане в играта

```

socket = io('http://localhost:3000')

socket.on('connect', () => {
    socket.emit('username', username)
})

socket.on('game', (newGameObjects) => {
    initialGameObjects = newGameObjects
    game = new Phaser.Game(config)
})

```

Фигура 3.23. Пускането на сокет клиент

За връзка се използва оригиналната Socket.IO 2.4 версия вкарана преди основният game.js файл от който е пусната играта. След изпращането на „username“ събитието, клиента изчаква за получаване на игровите обекти и инициализира нова Phaser игра.

```
let config = {  
  type: Phaser.AUTO,  
  width: window.innerWidth,  
  height: window.innerHeight,  
  backgroundColor: '#000000',  
  scene: [GameScene]  
}
```

Фигура 3.24. Конфигурация на Phaser игра

От този обект config задаваме желаните характеристики на играта. Много другия могат да бъдат зададени като начина на физика в играта, сблъскването на обекти и т.н. като всички могат да бъдат проверени в Phaser документацията. Най – важните за текущата игра са window.innerWidth и window.innerHeight, които задават измерения на максималния размер на прозореца в браузъра автоматично.

3.3.2 Селекция и създаване на игров обект

```
setStartX(){
    let boxX = this.x
    let closestX = Math.round(boxX / 30) * 30
    let floor = Math.floor(boxX / 30) * 30
    let farthestX = ((0.5 - (((closestX - floor) % 4) / 4)) * 60) + floor

    let widthCoef = (this.objectWidth % 4) / 4
    let coef = (closestX - farthestX) / 30
    let centerX = closestX - (coef * (30 * widthCoef))
    let projectX = centerX - (this.objectWidth / 2)

    this.startX = projectX
    return this
}

setStartY(){
    let boxY = this.y
    let closestY = Math.round(boxY / 30) * 30
    let floor = Math.floor(boxY / 30) * 30
    let farthestY = ((0.5 - (((closestY - floor) % 4) / 4)) * 60) + floor

    let heightCoef = (this.objectHeight % 4) / 4
    let coef = (closestY - farthestY) / 30
    let centerY = closestY - (coef * (30 * heightCoef))
    let projectY = centerY - (this.objectHeight / 2)

    this.startY = projectY
    return this
}
```

Фигура 3.25. Намирането на координати на обект

```
update(x, y) {
    this.setX(x).setY(y)

    if(this.getBuildSpriteActive()) {
        this.setStartX().setStartY().setBuildSpriteX().setBuildSpriteY()
    }
}
```

Фигура 3.26. Опресняването на селектирания обект

На фигура 3.25. вземаме координатите на горния десен ъгъл на избрания игров обект базиран на последните получени координати на мишката, като я взима за център на текстурата изобразена на екрана. Ако не сме селектирали обект, калкулациите не се изпълняват и селектирания обект не се показва.



Фигура 3.27. Поставянето на обект

Както центриране, използваните формули и като цяло използваната математика, цели да закръгли генерираните координати на 30x30, които са всъщност размерите на основните tile-ове в grid системата.



Фигура 3.28. Поставен обект

Това е успешно поставен обект върху 4 площи от материал.

```
this.input.keyboard.on('keyup', (key) => {
    action.clear()

    switch (key.keyCode) {
        case Phaser.Input.Keyboard.KeyCodes.ONE:
            action.setObject(0)
            break

        case Phaser.Input.Keyboard.KeyCodes.TWO:
            action.setObject(1)
            break

        case Phaser.Input.Keyboard.KeyCodes.D:
            action.setType('destroy')
            socket.emit('clickD', action)
            break
    }
})
```

Фигура 3.29. Създаването на обект

За да бъде създаден един обект, е нужно да се знаят характеристиките му. Те се задават през `setObject(Number)`, като „Number“ отговаря за индекса на масива с данните за всички обекти от който се взима информацията, като текстура, размери и тип на обекта. При натиск и отпускането на бутоните от 1-9 се избира обект, който първо се визуализира, след това ако е натиснат и отпуснат бутона на мишката, се изпраща събитие за действие на сървъра, за построяване на обект, със типа на обекта и координатите му.

```
// Game Objects Metadata
let gom = [{type: 'burnerDrill', texture: 'burner-drill', width: 60, height: 60},
           {type: 'pipe', texture: 'pipe-cross', width: 30, height: 30}]
```

Фигура 3.30. Данните за обектите

```
socket.on('build', (gameObject) => {
  gameObjects.push({sprite:
this.add.sprite(gameObject.startX, gameObject.startY,
gameObject.texture).setOrigin(0), uuid:
gameObject.uuid})
})

socket.on('destroy', (gameObject) => {
  let index = gameObjects.findIndex((element) =>
(element.uuid === gameObject.uuid))

  gameObjects[index].sprite.destroy()
  gameObjects.splice(index, 1)
})
```

Фигура 3.31. Събития за създаване и унищожаване на обект

След получаване на отговор от сървъра, отново във формата на събития от страна на сокета, ги добавяме в масива на обекти заедно с техните уникални id-та (UUID). При изтриване на обект, действието трябва да се извика чрез натискане на D бутона, при което отново първо се изпраща заявка на сървъра която е за изтриване включвайки координатите на които е мишката. Обектът при създаване както и изтриване, състоянието му е първо актуализирано в Backend-а и след това отразено на Frontend-а. На фигура 3.31. забелязваме, че за да изтрием обект

не е само достатъчно да освободим променливата която го държи, но и да извикаме `destroy()` метода, за да го изкара от буферите за рисуване на Phaser. След което като имаме индекса му, взет по сравняване на `id`-тата итеративно, го махаме чрез вградения за масив обекти в Javascript метод `splice()`.

```
socketIOServer.addListener( eventName: "clickLeft", Action.class, (client, action, ackRequest) -> {
    String username = clientUsernames.get(client.getSessionId().toString());

    GameObject gameObject = gameStorageRepository.buildObject(action, username, client, gameService);

    if(gameObject == null){
        return;
    }

    client.sendEvent( s: "build", gameObject);
    gameService.addGameObject(gameObject, username);
});

socketIOServer.addListener( eventName: "clickD", Action.class, (client, action, ackRequest) -> {
    String username = clientUsernames.get(client.getSessionId().toString());

    GameObject gameObject = gameStorageRepository.findGameObject(action, username);

    if(gameObject == null) {
        return;
    }

    gameService.deleteGameObject(gameObject, username);
    gameStorageRepository.deleteGameObject(gameObject, username, client);
    client.sendEvent( s: "destroy", gameObject);
});
```

Фигура 3.31. Добавянето и изтриването на обект

В събитието „clickLeft“ се опитваме да създадем обект и ако метода е успешен, го добавяме към MongoDB базата, пращаме го сериализиран на клиента и го добавяме към логиката на неговата игра. При унищожаване

на обекта се изпълняват същите операции в обратен ред. Първо трябва да се махне от логиката на играта на клиента, след това от базата данни, и накрая се изпраща заявка събитие на клиента за махането му от Frontend-a.

ЧЕТВЪРТА ГЛАВА

Ръководство на потребителя

4.1 Инсталация

4.1.1 Spring Boot

За да се пусне сървърът, трябва потребителят да има инсталирана Java 17+, и да не са му заети портовете 8080 и 3000. Алтернативно може да ги промени в кода на други.

4.1.2 MySQL и MongoDB

Хостът е също нужно да притежава пуснати сървъри на MySQL и MongoDB, локални или алтернативно да промени в кода данните за вход към сървърите в които да се пази информацията за потребителите и съдържанието на игровите сесии.

4.1.3 Създаване на акаунт

След успешно изпълнение на сървърът, потребителят може да се регистрира на <http://localhost:8080/register>, след регистрация си потвърди мейла, влезе в акаунта си от <http://localhost:8080/login>.

От този момент, може да влиза в играта само през началната страница <http://localhost:8080>.

ЗАКЛЮЧЕНИЕ

В началото на осъществяването на заданието имаше доста непредвидими проблеми, поради големият набор от нужните за създаване връзки и някои несъвместимости между големият набор от технологии. До този момент са развити основните фондации и системи, които ще служат за допълнителното пълноценно развитие на практическата част. Повечето функционални изисквания са покрити, без да е компрометирано нивото на производителност на приложението. Бъдещото развитие на играта е изразено в довършване на логистическа система за пренасянето на предметите и добавянето на работи, които автоматично да поддържат съответните обекти в играта.

ИЗПОЛЗВАНА ЛИТЕРАТУРА

1. Isabela Granic, Adam Lobel, and Rutger C. M. E. Engels, 2018, The Benefits of Playing Video Games
2. <https://ably.com/topic/socketio>
3. <https://www.javascript.com/>
4. <https://www.tutorialspoint.com/sql/index.htm>
5. <https://www.factorio.com/>
6. <https://www.minecraft.net/en-us>
7. <https://spring.io/projects/spring-boot>

ТЕРМИНОЛОГИЯ

1. WebGL - Графично изобразяващ двигател за Javascript, пренесен от 2. 2. OpenGL
2. XML - Разширен език за маркиране
3. Spritesheet - Изображение съдържащо всички варианти на дадена текстура
4. Bean - Инстанция на обект, заредена в контекста на Spring приложението
5. Анотация в Spring - Анотацията е маркировка, определяща функцията на даден клас
6. @Autowired анотация - Анотация, която позволява на обекта да си вземе инстанция от контекста на Spring приложението
7. JDBC - Java свързаност към базата данни, интерфейс който интерпретира заявки към бази от данни

СЪДЪРЖАНИЕ

УВОД	4
ПЪРВА ГЛАВА	6
1.1. Основни жанрове на игри в днешния свят	6
1.1.1 Научно-фантастичните компютърни игри	6
1.1.2 Игри с логическо мислене	7
1.1.3 Игри с фокусиран социален аспект	8
1.2. Основни технологии за реализиране на уеб игри	10
1.2.1 Phaser Game Engine	10
1.2.2 HTML - HyperText Markup Language	12
1.2.3 CSS - Cascading Style Sheets	14
1.2.4 Базы данни – SQL и MySQL	15
1.2.5 Базы данни – NoSQL и MongoDB	17
1.2.6 Javascript	19
1.2.7 Socket.IO	21
1.2.8 Spring Framework	22
1.2.9 Spring Boot	26
1.3. Подобни съществуващи разработки на игри	27
1.3.1 Factorio	27
Фигура 1.13. Factorio в действие	29
1.3.2 Minecraft	30
1.3.3 Satisfactory	32
ВТОРА ГЛАВА	35
2.1. Функционални изисквания към реализирането	35
• Да има сървър (Backend)	35
• Да има бази данни от двата типа SQL и NoSQL, в този случай MySQL и MongoDB	35
• Backend да поддържа логин, регистрация и изход	35
• Да се създава сървър на Socket.IO при пускане на Backend	35
• Socket.IO сървърът да свързва сесията на потребителя с Backend-a и да се поддържа двупосочна комуникация	35
• Да има Frontend с логин, регистрационна и целева страница която пренасочва и зарежда потребителя към играта му	35

• Да има система за поставяне на обекти	36
• Да има система за пренасяне на предмети и течности	36
• Да има автономни роботи	36
2.2. Избор на езици	36
2.2.1 Backend	36
2.2.2 Frontend	37
2.3. Избор за програмни средства	39
2.3.1 Разработване	39
2.4. Графични ресурси	40
2.4.1 Ресурси за играта	40
2.5. Структура на базата данни	40
2.5.1 MySQL	40
2.5.2 MongoDB	41
2.6. Описание на алгоритъма на действие	41
2.6.1. Backend	41
2.6.2 Frontend	42
ТРЕТА ГЛАВА	44
3.1. Сървър	44
3.1.1 Стартиране и конфигуриране на сървър	44
3.2 Сокет сървър	58
3.2.1 Стартиране на сокет сървъра	58
3.3 Frontend	60
3.3.1 Влизане в играта	60
3.3.2 Селекция и създаване на игров обект	62
4.1 Инсталация	69
4.1.1 Spring Boot	69
4.1.2 MySQL и MongoDB	69
4.1.3 Създаване на акаунт	69
ЗАКЛЮЧЕНИЕ	70
ИЗПОЛЗВАНА ЛИТЕРАТУРА	71
ТЕРМИНОЛОГИЯ	72
1. WebGL - Графично изобразяващ двигател за Javascript, пренесен от 2. 2. OpenGL	72
2. XML - Разширен език за маркиране	72
3. Spritesheet - Изображение съдържащо всички варианти на дадена текстура	72



4. Bean - Инстанция на обект, заредена в контекста на Spring приложението	72
5. Анотация в Spring - Анотацията е маркировка, определяща функцията на даден клас	72
6. @Autowired анотация - Анотация, която позволява на обекта да си вземе инстанция от контекста на Spring приложението	72
7. JDBC - Java свързаност към базата данни, интерфейс който интерпретира заявки към бази от данни	72
СЪДЪРЖАНИЕ	73