ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – гр. Варна

Факултет ФИТА

Катедра „Компютърни науки и технологии“

**ДИПЛОМНА РАБОТА**

**Тема:**

*Проектиране и разработване на система за управление на отпуски*

Дипломант: Делян Емилов Милчев

Специалност: КСТ Ръководител: ………...

Факултетен номер: 18621504 /ас. Мирослав Марков/

**Съдържание**

Contents

[1. Въведение 4](#_Toc112919713)

[1.1. Актуалност на проблема и мотивация 4](#_Toc112919714)

[1.2. Обзор на съществуващи решения 5](#_Toc112919715)

[2. Обзор на използваните програмни средства и технологии 6](#_Toc112919716)

[2.1. Java 6](#_Toc112919717)

[2.2. Spring Boot 7](#_Toc112919718)

[2.3. React 13](#_Toc112919719)

[2.4. JWT 13](#_Toc112919720)

[2.5. MySQL 15](#_Toc112919721)

[3. Проектиране на системата 17](#_Toc112919722)

[3.1. Функционалности на системата 17](#_Toc112919723)

[3.2. Проектиране на база с данни 18](#_Toc112919724)

1. Въведение

1.1. Актуалност на проблема и мотивация

1.2.Обзор на съществуващи решения

2. Обзор на използваните програмни средства и технологии

2.1. Java

2.2. Spring Boot

2.3.

2.3. React

2.4. JWT

2.5. MySQL

3. Проектиране на системата

3.1. Функционалности на системата

3.2. Проектиране на база с данни

3.3. Проектиране на бизнес логиката

3.4. Проектиране на потребителския интерфейс

4. Реализация и валидация

4.1. Разработка

4.2. Тестване

5. Заключения и изводи

6. Използвани литературни източници

7. Приложения

# Въведение

# Актуалност на проблема и мотивация

Дигитализацията е неизменна част от нашето ежедневие, като това определя как общуваме с околните - работим, пътуваме, употребяваме услуги, учим, вършим задължения. Поради пандемията причинена от COVID-19, дигитализацията стана още по-осезаема и настъпи с по-бързи темпове. Тя вече не е нещо препоръчително, а стана неизменна единица от ученици до компании от всички сектори и браншове. Всичко това ускори интегрирането на технологиите в работните процеси и комуникацията между хората и направи извършването на онлайн дейности нещо нормално за потребителите.

Точно в тези времена на дигитализация все повече бизнеси използват технологиите, за да улеснят и автоматизират работата си. Компаниите се възползват точно от тази технологична вълна за да надградят вече съществуващи решения както и да имплементират нови такива, с които биха били по-добри от своята конкуренция. В същото време, благодарение на дигитализацията, ще се подобри прозрачността на комуникацията, както и ще има по-добри условия за създаване на иновации.

Настоящото решение е именно една такава иновация, която ще подобри бързината на процеса, както и ще намали разходите на фирмите, които биха имали, ако наемат служители да вършат работата. Системата за отпуски, ще улесни както служителите, така и техните мениджъри, ще спести пари и ресурси на компаниите, които го използват.

Мотивацията за решението на този проблем, е, че той би помогнал както на бизнесите, така и на работниците, но не и на последно място на околната среда. Благодарение на системата за отпуски ще бъде спестена много хартия . При намаляването на производството на физически файлове и мастило ще се намалят вредните емисии.

# 1.2. Обзор на съществуващи решения

На пазара има няколко съществуващи решения, като сами по себе си са доказателство, че бизнесите имат нужда от подобни услуги.

Сравнение между две системи за отпуска налични на пазара

Vacation Tracker absence.io

Цена 25$ на месец 2$ на месец

Потребители 11-200 1-1000

Характеристики на системите:

Процес за одобряване да да

Календар да да

График на служителите да да

История на отпуските да да

Портал за самообслужване не да

Семейни и медицински отпуски да не

Рейтинг 4.7 4.7

Някои от съществуващите решения предлагат плащане на месец, а други таксуват спрямо броя на потребителите. Част от решения предлагат пакети, които включват различни екстри спрямо цената на пакета. Например, екстри в по-скъпия пакет са: неограничен брой потребители използващи системата, интеграция с различни календари (Google Calendar, Microsoft Outlook), импорт на потребители и отпуски, телефонна поддръжка, интеграция с други системи.

# 2. Обзор на използваните програмни средства и технологии

# 2.1. Java

Java е обектно ориентиран език за програмиране, разработен от Sun Microsystems и пуснат в употреба през 1995 година, като част от Java платформата. В последствие се появяват множество други реализации включително от GNU, Microsoft, IBM, Oracle и други технологични доставчици.

Сред главните предимства са:

• лесната преносимост между различните платформи (софтуерни или хардуерни) – веднъж написана и компилирана, една Java-програма може да бъде стартирана на различни компютри, независимо от архитектурата или от операционната им система. За целта е необходимо само да се инсталира виртуална машина за съответната платформа

• допълнителните действия, извършвани от виртуалната машина, като освобождаване на паметта от обекти (класове), които не се използват), проверка за размерността на масивите

• възможността за контрол на правата на потребителя на ниво виртуална машина

• първоначално заделяне на динамична памет, част от паметта резервирана за java

• висока степен на сигурност поради факта, че програмистите не работят директно с паметта

Главен недостатък е необходимостта от допълнителни ресурси (под формата на процесорно време и памет) за работата на самата виртуална машина.

Съществуват и компилатори, които превеждат байт кода до машинен код. Недостатък при използването им е, че тогава програмата не може да се стартира на различни платформи.

Ранните версии на виртуалните машини страдат от слаба производителност поради необходимостта от компилиране на байт кода до машинен код. Този проблем е решен в значителна степен след въвеждането на JIT компилатора във виртуалната машина.

Java e език за програмиране от високо ниво с общо предназначение. Синтаксисът му е подобен на C и C++, но не поддържа много от неговите възможности с цел опростяване на езика, улесняване на програмирането и повишаване на сигурността. Програмите, написани на Java, представляват един или няколко файла с разширение .java.

## 2.2. Spring Boot

Spring фреймуъркът е технологична рамка с отворен код за Java платформата. Spring предоставя много функции, които улесняват разработването на Java-базирани корпоративни приложения.

Spring включва различни модули, които предоставят широк набор от функционалности:

• Контейнер на зависимостите - Основното ядро в Spring фреймуъркът е неговият контейнер за обръщане на контрола, който осигурява логически средства за конфигуриране и управление на Java обекти с помощта на отражение. Контейнерът е отговорен за управлението на жизнения цикъл на специфични обекти: създаването на тези обекти, извикването на инициализиращите им методи и конфигурирането им свързвайки ги заедно. Обектите, създадени от контейнера се наричат още управлявани обекти или бийнове .

Обектите могат да бъдат получени чрез търсене на зависимост или инжектиране на зависимост. Инжектирането на зависимост е софтуерен шаблон за дизайн, който опростява обектно - ориентираното програмиране (ООП) чрез намаляване на зависимостите между отделните класове. Така при създаването на един проект, използвайки някой обектно - ориентиран език се избягва зависимостта от конкретни имплементации, и се набляга повече на абстракции. Това се нарича още обръщане на контрола – когато модулите от високо ниво не зависят от модули от по-ниско ниво.

При инжектирането на зависимост обектите се проектират по начин, при който те получават инстанции на обекти от различен външен модул код, вместо да са конструирани вътре в него. Инжектирането на зависимост включва най-общо три елемента:

o зависим елемент;

o декларация на зависимостите на елемента, дефинирани като интерфейси, абстрактни класове;

o контейнер на зависимостите (инжектор), който създава инстанции на класове, които реализират дадена интерфейсна зависимост при заявка.

Зависимият елемент определя на какъв софтуерен компонент зависи за да си свърши работата, а контейнерът решава какви конкретни класове отговарят на изискванията на зависимия обект, и ги предоставя като зависимости. По този начин контейнерът по свой избор, може да променя конкретните имплементации, които се използват по време на изпълнение на програмата, а не по време на компилация . По време на изпълнение могат да бъдат създадени множество различни инстанции на даден софтуерен компонент, без значение, каква конкретна е била инжектирана.

Това позволява кодът, който пишем да е много по модуларен и разкачен, което го прави в същото време и много по-тестваем.

• Аспектно-ориентираното програмиране (АОП) е относително нова парадигма в програмирането която цели да се увеличи модулността на софтуера и включва методи и инструменти които помагат за това.

Почти всички програмни парадигми поддържат някакво ниво на групиране и капсулиране на проблеми в отделни, независими части чрез представянето им чрез абстракции (например, функции, процедури, модули, класове, методи), които могат да бъдат използвани за изпълнение, абстрахиране и композиране на отношения. Някои отношения „пресичат“ множество абстракции в програмния код, и се наричат междусекторни отношения.

Аспект наричаме единица, която капсулира дадено отношение, което пресича системните елементи в множество точки. Техниката на програмиране, която се занимава с тези дизайн проблеми се нарича Аспектно-ориентирано програмиране (АОП). Аспектно-ориентираното разработване на софтуер (АОРС) е нова технология, която се опитва да наложи АОП в масовото производство на софтуер с цел по-добро разграничение и моделиране на отношенията. Аспектите се ползват във всяка една стъпка от цикъла на производствения процес. Типични примери за аспекти са: сигурност, качество на услугата , поддръжка на трансакции, синхронизация и трасиране.

• Достъп на данни

Така наречените Spring фреймуърци за достъп на данни се отнасят за често срещани проблеми при работа с база данни, които разработчиците срещат. Както следват всички най-известни фреймуърци за достъп на данни в Java се поддържат: JDBC, iBatis/MyBatis, Hibernate, JDO, JPA, Oracle TopLink, Apache OJB, и Apache Cayenne.

За всички изброени дотук фреймуърци, Spring поддържа следните характеристики:

o Автоматично поискване и получаване на ресурси от базата данни

o Превежда уникалните данните за достъп в Spring йерархия

o Прозрачно участие в настоящи трансакции

o Абстракция за BLOB и CLOB манипулации

Всички тези възможности са достъпни когато се използват шаблонни класове предоставени за всеки фреймуърк който Spring поддържа.

Заедно с управлението на Spring трансакциите, този фреймуърк предлага гъвкава абстракция за работа с фреймуърци за достъп на данни. Spring не предлага стандартен API достъп на данни; вместо това цялата мощ на поддържащото API е оставена непокътната. Spring фреймуъркът е единствения фреймуърк достъпен за Java, който предлага управление на достъпа до данните извън сървъра или контейнера на приложението.

• Модел-Изглед-Контролер

Модел-Изглед-Контролер (MVC) е архитектурен модел за изграждане на потребителски интерфейси. Той разделя дадено софтуерно приложение на три взаимносвързани части, така че да се отделят вътрешни представяния на информация от начините, по които информацията се представя на потребителя/или от страна на потребителя: контролери, изгледи, модели. Както и при други софтуерни модели, MVC изразява ядрото на решението на проблем, докато позволява да бъде адаптиран за всяка система. Специфични MVC архитектури могат да се различават значително.

Елементи: Централният елемент на MVC – моделът, улавя поведението на приложението по отношение на своя домейн, независимо от потребителския интерфейс. Моделът директно управлява данните, логиката и правилата на приложението.

Изгледът може да бъде всеки изход за представяне на информация, например чрез графика или диаграма; множество изгледи към една и съща информация са възможни, като стълбчета за управление и табличен изглед за счетоводители.

Третата част контролера, приема входа от потребителя и го конвертира в команди за модела или изгледа.

Взаимодействия: В допълнение за разделяне на приложението на трите вида елементи, дизайн на MVC определя взаимодействието между тях. Контролерът може да изпраща команди към модела за актуализиране състоянието на модела. Той също може да изпраща команди към свързаната с изгледа част, за да се промени представянето на модела.

Моделът уведомява свързаните части с изгледа и контролера, когато е налице промяна в състоянието му. Това уведомяване позволява на изгледа да изведе на изхода актуалната информация и контролера да променя разположението на наборите от команди.

Изгледа изисква информация от модела, който използва, за да се генерира изходната информация за представяне на потребителя.

Използване в уеб приложения: Въпреки че първоначално е разработен за настолните компютри MVC е широко приет като архитектура за World Wide Web приложения в голяма част от езици за програмиране. Няколко търговски и нетърговски уеб приложения са създадени, които прилагат тази схемата. Тези приложения се различават по своите интерпретации, основно в начина, по който отговорностите им MVC са разделени между клиента и сървъра.

По рано уеб MVC рамките поведоха с подход, който оставя почти цялата логика модел, изглед и контролер на сървъра. При този подход, клиентът изпраща или хипервръзки искания или форма вход към контролера и след това получава пълна и актуализирана уеб страница (или друг документ) от гледката– модела съществува изцяло на сървъра.

• Интеграция

Интеграционния Spring фреймуърк е фреймуърк с отворен код за интеграция на корпоративни приложения. Той предоставя възможност за разработването на интеграционни решения, типични за архитектурите свързани със събития и съобщения.

o рутери;

o трансформатори;

o адаптери за интегриране с други технологии и системи (HTTP, AMQP, JMS, XMPP, SMTP, IMAP, FTP (както и FTPS/SFTP), файлови системи)

# 2.3. React

React е JavaScript библиотека за изграждане на потребителски интерфейс, с която може да се програмират уебсайтове и мобилни софтуерни приложения, която е с отворен код . Поддържа се от Facebook и общност от разработчици и компании.

React може да се използва като основа при разработването на уеб страници или мобилни приложения. React обаче се занимава само с предоставяне на данни в DOM и затова създаването на React приложения обикновено изисква използването на допълнителни библиотеки за управление на състоянието на графичните елементи, както и за маршрутизиране. Redux и React Router са съответните примери за такива библиотеки.

# 2.4. JWT

JSON Web Token (JWT) е JSON-базиран отворен стандарт (RFC 7519) за създаване на „жетони“, които съдържат определен брой твърдения. Например, един сървър може да генерира жетон, който съдържа твърдението „влязъл като администратор“ и достави тази информация до клиент. Клиентът тогава може да използва това означение да докаже, че е влязъл като администратор. Жетоните са подписани от ключа на сървъра, така че сървърът да е в състояние да удостовери, че подписът е законен. Жетоните са проектирани да бъдат компактни, URL-безопасни и използваеми най-вече в уеб браузър единично влизане (SSO) контекст. Твърденията JWT обикновено се използват, за да се предаде самоличността на регистрирани потребители между доставчик на идентичност и доставчик на услуга, или всякакъв друг вид твърдения, както се изисква от бизнес процесите. Жетоните могат да бъдат потвърдени и криптирани.

JWT обикновено има три части: заглавие, полезен товар и подпис. Заглавието идентифицира кой алгоритъм се използва за генериране на подписа, и изглежда по следния начин:

header = '{"alg":"HS256","typ":"JWT"}'

HS256 показва, че този жетон е подписан с помощта HMAC-SHA256.

Полезният товар съдържа твърденията, които искаме да добавим:

payload = '{"loggedInAs":"admin","iat":1422779638}'

Както се предлага в спец JWT, включваме дата и час, наречени IAT, съкратено от „издаден в“.

Подписът се изчислява чрез base64url кодиращи заглавието и полезния товар и ги слепва с точка за разделител:

key = 'secretkey'

unsignedToken = encodeBase64(header) + '.' + encodeBase64(payload)

signature = HMAC-SHA256(key, unsignedToken)

Вземайки всичко заедно, ние кодираме подписа с base64url, и присъединяваме трите части, използвайки точки:

token = encodeBase64(header) + '.' + encodeBase64(payload) + '.' + encodeBase64(signature) # token is now:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFzIjoiYWRtaW4iLCJpYXQiOjE0MjI3Nzk2Mzh9.gzSraSYS8EXBxLN\_oWnFSRgCzcmJmMjLiuyu5CspyHI

Изходът е три Base64 низа, разделени с точки, които могат лесно да бъдат прехвърлени в HTML и HTTP среди, като същевременно бъдейки по-компактен в сравнение с XML-базирани стандарти като SAML. Типични криптографски алгоритми, които са използвани, са HMAC с SHA-256 (HS256) и RSA подпис с SHA-256 (RS256).

В удостоверяване, когато потребителят успешно се регистрира, използвайки своите данни, JSON Web Token ще бъдат върнат и трябва да бъде запазен локално (обикновено в локално пространство, но също така могат да се използват и бисквитки), вместо традиционния подход за създаване на сесия в сървър и връщане на бисквитка.

Всеки път, когато потребителят поиска да получи достъп до защитен път или ресурс, потребителската програма трябва да изпрати JWT, обикновено в авторизационната заглавна част, използвайки схемата носител. Съдържанието на заглавието трябва да изглежда по следния начин: Authorization: Bearer <token>

Това е нестатичен механизъм за удостоверяване като състоянието на потребителя никога не е записано в паметта на сървъра. Защитените пътища на сървъра ще проверяват за валиден JWT в заглавната част на Авторизиращата заглавна част, и ако той е налице, на потребителят ще бъде позволено да получи достъп до защитените ресурси. Както JWT са самостоятелни, цялата необходима информация е налице, намалявайки необходимостта да запита базата от данни на няколко пъти.

# 2.5. MySQL

MySQL е релационна база данни, която използва SQL (Structured Query Language) – най-популярният език за добавяне, достъпване и обработване на информация в база от данни. Тя се разпространява и поддържа от Oracle Corporation.

MySQL базата данни е с отворен код, може да бъде свалена от уеб и използвана от всеки, според неговите нужди, без нужда от лицензиране.

Предимството на MySQL е, че специфичната информация може да бъде съхранявана в отделни, много на брой малки таблици с установени между тях се връзки (релации), вместо в една единствена огромна таблица. С това много се улеснява и ускорява обработката на информацията.

Релационната база данни е тип база данни, която съхранява множество данни във вид на релации, съставени от записи и атрибути (полета) и възприемани от потребителите като таблици. Релационните бази данни понастоящем преобладават при избора на модел за съхранение на финансови, производствени, лични и други видове данни.

# 3. Проектиране на системата

# 3.1. Функционалности на системата

Приложението ще включва следните функционалности:

• Страница за вписване на потребител

• Създаване, редактиране, изтриване на потребители

• Създаване, редактиране, изтриване на работни отдели

• Създаване, редактиране, изтриване на отпуски

• Възможност за добавяне/премахване на потребител към работен отдел

• Възможност за добавяне/премахване на мениджър за потребител

• Три вида отпуска - платен, неплатен и болничен

• При искане на платен или неплатен отпуск, потребителя трябва да въведе следните данни - от коя дата до коя дата ще е отпуска, причина за иска на отпуска, типа на отпуската - платен или неплатен отпуск.

• При искане на болничен потребителя трябва да въведе следните данни - от коя до коя дата ще е болничния му,

причина за иска на отпуска, избиране на тип отпуска - болничен, прикачване на болничен лист

• Отпуската му за болничен ще бъде автоматично одобрена

• Визуална репрезентация на календар, който показва за кои дни потребителя си е пуснал отпуска, като ги оцветява в различни цветове спрямо типа на отпуската

• От страни на календара има легенда - кой цвят, какъв вид отпуска е

• Информация за официалните почивни дни, като те няма да могат да се включват към отпуските

• Информация кой е мениджъра, който трябва да одобри отпуската

• Информация за всички отпуски, които потребителя си е пуснал(история на отпуските)

# 3.2. Проектиране на база с данни

Базата с данни се състои от четири таблици. Таблица „Отдели“ съхранява информация за всички налични отдели във фирмата – id и име на отдела. Таблица „Потребители“ съдържа информация за потребителя – id, емайл, парола, име, фамилия, налични дни за платен отпуск, дали потребителя е админ и id на отдела. Таблица „Взети дни“ съхранява информация за всички поискани от потребителя отпуски. Таблица „Заявка“ съхранява информация за текущите заявки за отпуска потребителя.