Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка інформаційної системи адміністрування робочого часу співробітників підприємства»

Студента 2м курсу, 6 групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Шабельника Іллі Ярославовича

Науковий керівник кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис керівника — Рзаєва Світлана Леонідівна

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Криворучко Олена Володимирівна

КИЇВ - 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студента

Шабельника Іллі Ярославовича
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Розробка інформаційної системи адміністрування робочого часу співробітників підприємства»
Затверджена наказом ректора від "13" грудня 2019 р. № 4304

2. Строк здачі студентом закінченої проекту 01 грудня 2020 р.

3. Цільова установка та вихідні дані до проекту
Мета проекту дослідити існуючі інформаційні системи та розробити програмне забезпечення адміністрування робочого часу на підприємстві.
Об'єкт дослідження організація робочого процесу працівників.
Предмет дослідження розробити програмне забезпечення адміністрування робочого часу на підприємстві.

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО АДМІНІСТРУВАННЯ

1.1. Аналіз методів управління

1.2. Робочий час та способи контролю

1.3. Висновок до розділу 1

РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ ТА СЕРВЕРУ

2.1. Вибір бази даних та її проектування

2.2. Опис архітектури додатку

2.3. Вибір інструментарію для розробки

2.4. Висновок до розділу 2

РОЗДІЛ 3. РОЗРОБКА СЕРВЕРУ. РЕАЛІЗАЦІЯ КОМПОНЕНТІВ КЛІЄНТСЬКОЇ ЧАСТИНИ

3.1. Створення структури серверної частини

3.2. Поетапне створення реєстрації на сервері

3.3. Розробка клієнтської частини

3.4. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ТЕХНІЧНЕ ЗАВДАННЯ

ТЕСТУВАННЯ ВЕБ-САЙТУ WORK TIME MANAGER

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проекту*	*20.09.2019*	*20.09.2019*
2.	*Розробка та затвердження завдання на проект магістра*	*08.11.2019*	*08.11.2019*
3.	*Вступ та перелік літературних джерел*	*24.01.2020*	*24.01.2020*
4.	*Наукова стаття*	*01.09.2020*	*01.09.2020*
5.	*Технічне завдання*	*28.02.2020*	*28.02.2020*
6.	*Розділ 1. Загальні відомості про адміністрування*	*25.06.2020*	*25.06.2020*
7.	*Розділ 2. Проектування бази даних та серверу*	*07.09.2020*	*07.09.2020*
8.	*Розділ 3. Розробка серверу. Реалізація компонентів клієнтської частини*	*19.10.2020*	*19.10.2020*
9.	*Програма та методика тестування*	*21.10.2020*	*21.10.2020*
10.	*Керівництво користувача*	*23.10.2020*	*23.10.2020*
11.	*Висновки та пропозиції*	*30.10.2020*	*30.10.2020*
12.	*Здача випускного кваліфікаційного проекту на кафедру (перша перевірка)*	*05.11.2020*	*05.11.2020*
13.	*Підготовка автореферату та презентації доповіді*	*05.11.2020*	*05.11.2020*
14.	*Попередній захист випускного кваліфікаційного проекту*	*25.11.2020-27.11.2020*	*25.11.2020-27.11.2020*
15.	*Зовнішнє рецензування випускного кваліфікаційного проекту*	*01.12.2020*	*01.12.2020*
16.	*Підготовка до публічного захисту випускного кваліфікаційного проекту*	*08.12.2020-09.12.2020*	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту Рзаєва С. Л.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О. В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Шабельник І. Я.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

Науковий керівник випускного кваліфікаційного проекту

(підпис, дата)

Відмітка про попередній захист Рзаєва С. Л.

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента Шабельник І. Я.

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми Криворучко О. В.

(прізвище, ініціали, підпис)

Завідувач кафедри Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

У відповідності до мети дослідження робота присвячена розробці веб-сайту, який дозволить створювати організації, реєструвати в ній працівників з різними ролями (правами доступу), управляти їх даними, створювати для них завдання з різними пріорітетами, продивлятися та перевіряти ці завдання та створювати звіти.

Відповідно до вимог було створене технічне завдання, яке дало більш ширше поняття, які технології бажано використовувати при розробці. В результаті, розробка веб-сайту була поділена на дві частини: клієнтську та серверну.

В клієнтській частині використовувалась мова програмування JavaScript з React JS та Redux JS фреймворками, які дозволили створити структурований веб-сайт з технологією SPA(single page application).

Серверна частина була розроблена з використанням мови програмування JavaScript (Node JS) та фреймворком Express JS. В якості сховища де зберігатимуться була вибрана нереляційна база даних MongoDB.

Весь проект розроблявся в середовищі WebStorm. Для взаємоді цих частин, використовувались додаткові інструменти та сервіси.

По закінченю розробки були проведені тести, які показали успішні результати.

Ключові слова: SPA, веб-сайт, сервер, інформаційна система, React JS, Redux JS, MongoDB, Express JS.

ABSTRACT

In accordance with the research goal, the work is dedicated to developing a website that will allow creating organizations, registering employees with different

roles (access rights), managing their data, creating tasks for them with different priorities, viewing and checking these tasks and creating reports.

In accordance with the requirements, the terms of reference were created, which gave a broader concept of what technologies are better to use in development. As a result, site development was divided into two parts: client and server.

The client part used JavaScript programming language with React JS and Redux JS frameworks, which allowed to create a structured website with SPA (single page application) technology.

The whole project was developed in WebStorm environment. For interactions of these parts, additional tools and services were used.

At the end of development, tests were carried out, which showed successful results.

Keywords: SPA, website, server, information system, React JS, Redux JS, MongoDB, Express JS.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

ПК – персональний комп’ютер

ОС – операційна система

JSON (англ. JavaScript Object Notation) – це текстовий формат обміну даними між комп'ютерами.

MVC (англ. Model–View–Controller) – це шаблон дизайну програмного забезпечення.

ТЗ – технічне завдання

					КНТЕУ 121 06-23.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*19.10.20*	Розробка інформаційної системи адміністрування робочого часу співробітників підприємства	*Стадія*	*Арку*	*Аркушів*
Керівник		Рзаєва С.Л.		*19.10.20*		*ПС*	*2*	*52*
Гарант		Криворучко О.В.		*19.10.20*		Факультет інформаційних технологій 2м курс, 6 група		
Розробив		Шабельник І.Я.		*19.10.20*	*Перелік умовних скорочень*			

ЗМІСТ

					КНТЕУ 121 06-23.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		*19.10.20*	Розробка інформаційної системи адміністрування робочого часу співробітників підприємства	*Стадія*	*Арку*	*Аркушів*
Керівник		Рзаєва С.Л.		*19.10.20*		*Зміст*	*2*	*52*
Гарант		Криворучко О.В.		*19.10.20*		Факультет інформаційних технологій 2м курс, 6 група		
Розробив		Шабельник І.Я.		*19.10.20*	*Зміст*			

ВСТУП

Актуальність. Розвиток інформаційно-комунікаційних технологій (ІКТ) у другій половині 20-го століття, що включає в себе створення та масштабне впровадження інформаційних систем, з упевненістю можна назвати інформаційною революцією, адже за своєю насиченістю, темпу, глобальності історія людства не має подібних аналогів. ІКТ займають важливе місце у всіх сферах людського життя, а також це не оминуло і діяльність підприємства.

Навіщо потрібен облік робочого часу? Таким питанням може задаватися тільки той співробітник, який не хоче, щоб його регулярні запізнення і відлучки в особистих справах, інакше кажучи «порушення трудової дисципліни», були зафіксовані. А коли число співробітників перевищує десятки, а то і сотні людей, простежити за робочим часом кожного стає нездійсненним завданням, що вимагає певних підходів до вирішення. Будь-який грамотний керівник знає, що цілком віддаватися роботі протягом усього робочого часу не може жодна жива людина, проте не можна допускати випадків зловживання запізненнями або тривалими перервами. Для керівника кожен співробітник - це найбільш цінний інтелектуальний ресурс в процесі виробництва товарів або послуг, що виконує свої цілі і завдання. І якщо цей ресурс стає систематично недоступний або знаходиться в неналежному стані, це негайно позначається не тільки на виконанні індивідуального плану співробітника, але і на роботу відділу, організації, а то і на економічних показниках діяльності компанії в цілому.

Згідно з даними різних досліджень, від 30 до 50% робочого часу витрачається не на робочий процес. Майже половина тимчасових втрат відбувається через неграмотного планування, а ще приблизно третина - від слабкого нагляду за співробітниками.

Мета дослідження: дослідити існуючі інформаційні системи та розробити програмне забезпечення адміністрування робочого часу на підприємстві.

Об'єкт дослідження: організація робочого процесу працівників.

Предмет дослідження: розробка програмного забезпечення адміністрування робочого часу співробітників

У відповідності з метою дослідження поставлені наступні *завдання*:

- Проведення аналізу організації робочого часу працівників
- Визначити процес, який потрібно впровадити у проект для підвищення ефективності роботи підприємства
- Розробити веб-сайт відповно вимогам, які вказані у технічному завдані.
- Провести тестування наявного функціонала за допомогою мануального тестування.

Методи дослідження: процес взаємодії працівників з керівництвом на підприємстві.

Наукова новизна дослідження полягає в удосконаленні організації робочого часу працівника та ефективності виконання поставлених завдань.

РОЗДІЛ 1.
ЗАГАЛЬНІ ВІДОМОСТІ ПРО АДМІНІСТРУВАННЯ

1.1. Аналіз методів управління

Методи управління - це сукупність прийомів і способів цілеспрямованого впливу на виробничий колектив або окремого працівника з метою спонукати їх зробити певні дії в інтересах підприємства.

Методи управління відрізняються один від одного своєю мотиваційної характеристикою, тобто тим, на активізацію яких мотивів поведінки людей вони орієнтовані.

Методи поділяються на:

1. Адміністративні
2. Економічні
3. Соціально-психологічні

Адміністративні методи управління. Вступаючи в виробничий колектив, людина приймає на себе певні обов'язки і відповідальність за якісне виконання відповідної роботи і за результати роботи колективу, в цілому (в якійсь мірі). За допомогою адміністративних методів визначаються: місце колективів і окремих працівників в системі виробництва і управління; їх права, обов'язки і міра відповідальності; способи координації їх дій і взаємозв'язку в процесі виробництва і управління.

Адміністративні (організаційно-розпорядчі) методи управління мають наступні особливості:

1) Мають прямий вплив на волю підлеглих, що проявляється в однозначності віддаються розпоряджень і в обов'язки виконання будь-якого адміністративного акта;
2) Носять безоплатний характер, який не передбачає стимулювання;
3) Вимагають наявності і використання контролю виконання;
4) Вимагають не найкращого рішення проблем, а виконання суворо визначених дій.

Переваги адміністративних методів управління:

1) вони ефективні в примітивних ситуаціях;
2) дозволяють встановити сувору дисципліну;
3) забезпечують обрану технологію виробництва і управління.

До недоліків даного методу відносяться:

1) Не сприяють розвитку творчого початку особистості;
2) Вимагають обов'язкового оформлення всіх прийнятих рішень, що негативно впливає на час їх реалізації
3) Досить часто негативно оцінюються персоналом

Адміністративні методи управління поділяються на: організаційні (регламентування, нормування, інструктування), розпорядчі та дисциплінарні методи.

Організаційні методи орієнтовані на використання в типових ситуаціях.

Регламентування:

Сутність регламентування полягає у встановленні статусу і цілей функціонування, повноважень, прав і відповідальності, правил і критеріїв оцінки діяльності об'єкта регламентування.

Як об'єкт регламентування може розглядатися: організація в цілому (основний регламентує документ - статут організації); структурний підрозділ

(положення про підрозділ); посаду в апарат управління (посадова інструкція); відносини між організацією і персоналом (правила внутрішнього розпорядку, правила прийому на роботу); технологія виконання управлінських робіт (технологічний паспорт, схеми документообігу в організації і підрозділах).

Нормування:

Нормування як метод управління використовує норми, що мають конкретне числове вираження, і нормативи, які носять загальний, типовий характер і є основою для розробки норм.

Основні види нормування:

1) Чисельності, тобто визначення кількості людей, необхідних для виконання певної роботи;
2) Вироблення і обслуговування, тобто визначення кількості виконуваних виробничих операцій в одиницю часу, кількості одиниць виробленої продукції, кількості обслужених клієнтів або наданих послуг за певний час;
3) Управління, тобто визначення кількості підлеглих у одного керівника;
4) Витрати ресурсів, тобто визначення кількості ресурсів, що витрачаються при виконанні будь-якої роботи.

Інструктування:

Метою інструктування є ознайомлення працівників з умовами роботи, прийнятими рішеннями, що стоять перед ними завданнями, наслідками невиконання будь-якого завдання.

Види інструктування: ознайомлення, рада, пояснення, застереження, роз'яснення.

За формою здійснення інструктування поділяють на усне і письмове, індивідуальне і колективне.

Розпорядчий метод:

Розпорядчі методи застосовуються при необхідності втрутитися в процес виробництва і управління для усунення існуючих відхилень або для реалізації представилися можливостей.

Вони здійснюються в наступних формах:

1) наказу - документа, в якому сформульовані цілі, зміст, обсяг і терміни виконання завдань, вказані виконавці і умови виконання цих завдань, а також визначено посадова особа, яка здійснює виконання; видає наказ керівник організації або посадова особа, якій делеговані відповідні повноваження;
2) постанови, яке приймається на рівні всього підприємства спільно адміністрацією і громадськими організаціями;
3) розпорядження - усної або письмової вимоги до підлеглих виконати певні види робіт з метою вирішення будь-яких питань (видаються, як правило, функціональними керівниками або лінійними керівниками середнього рівня);
4) вказівки, які здійснюються в усній формі, використовується на нижчому рівні управління.

Дисциплінарний метод:

Дисциплінарний вплив регламентується КЗпП (Кодекс Законів про Працю) і застосовується в разі невиконання працівником своїх функціональних обов'язків. Право на його здійснення має тільки керівник, який виступає в ролі роботодавця.

1.2. Робочий час та способи контролю

Робочий час - це час, протягом якого працівник виконує свої професійні обов'язки за певну плату. Саме цей ресурс «купує» роботодавець.

Облік ведеться різними способами - від простих відміток в паперовому табелі до комп'ютерних програм, які збирають статистику, запам'ятовують відвідані сайти, роблять знімки екрану і самі формують вартість проекту.

Для вибору системи обліку власник повинен вирішити - який фактор для нього головний. Якщо мова йде про вантажоперевезення, оптимальним рішенням буде моніторинг переміщень, на виробництві - відеоспостереження. Важливо не просто фіксувати показники, а й аналізувати їх: скільки часу витрачає співробітник на робочу задачу, наскільки часто відволікається і чому, чи є у нього переробки. Ці дані потрібні не тільки для оцінки роботи персоналу, а й для планування, прийняття управлінських рішень, вибудовування ефективної системи роботи.

Основою для планування робочого графіка повинен виступати виробничий календар. У ньому відображені всі державні вихідні, переноси робочих днів, нормативне робочий час для різних режимів роботи. Навіщо це потрібно? Робота у вихідні та святкові дні оплачується в подвійному розмірі, понаднормова робота також підлягає додатковій оплаті. На основі виробничого календаря формується режим роботи: графік змінності, табелі обліку робочого часу.

Способи контролю робочого часу:

1) **Звіти про виконану роботу** – співробітники самостійно описують, чим вони займалися протягом дня, які виконали завдання, скільки на них пішло часу. Звіти можуть оформлятися в текстовому редакторі,

але більш вдалий варіант - ведення обліку в таблицях Excel або Google, дані з яких об'єднуються в один звіт. Перевага цього методу - він практично безкоштовний.

Недоліки – співробітник може бути не зовсім чесний в оцінці, а постійне написання звітів вимагає дисципліни.

2) **Система контролю доступу** – це пристрої, які проводять ідентифікацію співробітника для надання можливості входу в офіс або на територію підприємства. Ця система обліку робочого часу фіксує час входу і виходу, присутність людини на роботі. Розпізнавання може проходити за біометричними параметрами (відбитку пальця, голосу або сітківці ока) або за допомогою безконтактних магнітних карток. Пристрої зі скануванням карток дешевше, але менш стійкі до обходу.

Недоліки - враховується виключно час присутності на робочому місці, вона не відображає те, чим людина займається в робочий час.

3) **Відеоспостереження** – встановлення обладнання, яке фіксує те, що відбувається. Це проста, але ефективна система контролю. Камери дають можливість подивитися, чим зайняті співробітники, з ким вони спілкуються, чи приходять вчасно на роботу. Є думка, що відеоспостереження дисциплінує працівників, але, з часом, люди до нього звикають, перестають звертати увагу на зйомку і працюють як зазвичай. Відеозапис може стати доказом у спірній ситуації, з її допомогою можна відновити картину події. Спостереження може здійснюватися в режимі реального часу співробітником відділу безпеки. Але частіше все, що відбувається просто записується і переглядається в разі потреби.

Недоліки – більшості працівників не подобається працювати під прицілом камер, це може негативно позначитися на психологічному кліматі в колективі, також для аналізу відеозаписів потрібно багато часу. У камер є «сліпі» зони, обладнання іноді виходить з ладу. Не завжди зрозуміло, чим займається співробітник - розкладає пасьянс або складає звіт.

4) **Аудіоконтроль** – запис і прослуховування телефонних розмов. Може використовуватися для співробітників, робота яких полягає в спілкуванні з клієнтами. Він показує: скільки часу витрачається на консультацію, чи слід фахівець заданим алгоритмом розмови, наскільки він ввічливий і компетентний.
5) **Моніторинг пересувань** – контроль переміщень співробітників з роз'їзним характером роботи. Здійснюється за допомогою GPS-трекера, встановленого в автомобілі. Роботодавець отримує інформацію про маршрут, оцінює витрачений на поїздки час і ефективність логістики. Схожу послугу зараз надають стільникові оператори - за невелику абонентську плату можна отримувати дані про переміщення мобільного телефону і працівника разом з ним.
6) **Збір даних про роботу за допомогою спеціалізованих сервісів** – ця система обліку робочого часу може оцінити, чим займався співробітник і як довго він працював над завданням.

Співробітники працюють **продуктивно** близько трьох годин в день, з'ясували британські вчені. Решту часу витрачається на перегляд соціальних мереж, читання новин і спілкування з колегами. Щоб дізнатися, чим конкретно зайнятий співробітник в робочий час, на його комп'ютер встановлюють спеціальну програму.

Сервіси і програми обліку автоматизують контроль. Вони враховують час роботи на комп'ютері, час простою, що минув з останнього руху мишкою, відкриті документи і запущені програми, не відволікаючи співробітника. З їх допомогою можна сформувати звіти по використанню часу у вигляді діаграм або таблиць. Керівнику стає простіше аналізувати завантаженість підлеглих і планувати роботу. Дані зберігаються стільки, скільки потрібно, в будь-який момент можна подивитися, хто чим займався. Деякі програми дозволяють зробити скріншот робочого столу або знімок веб-камерою.

Kickidler – віддалений моніторинг комп'ютерів в реальному часі дозволяє побачити, чим зайняті співробітники зараз, які сайти відкривають, які програми використовують. Програма для відстеження комп'ютера автоматично сигналізує про скоєних порушень робочого розпорядку за встановленими заздалегідь правилами.

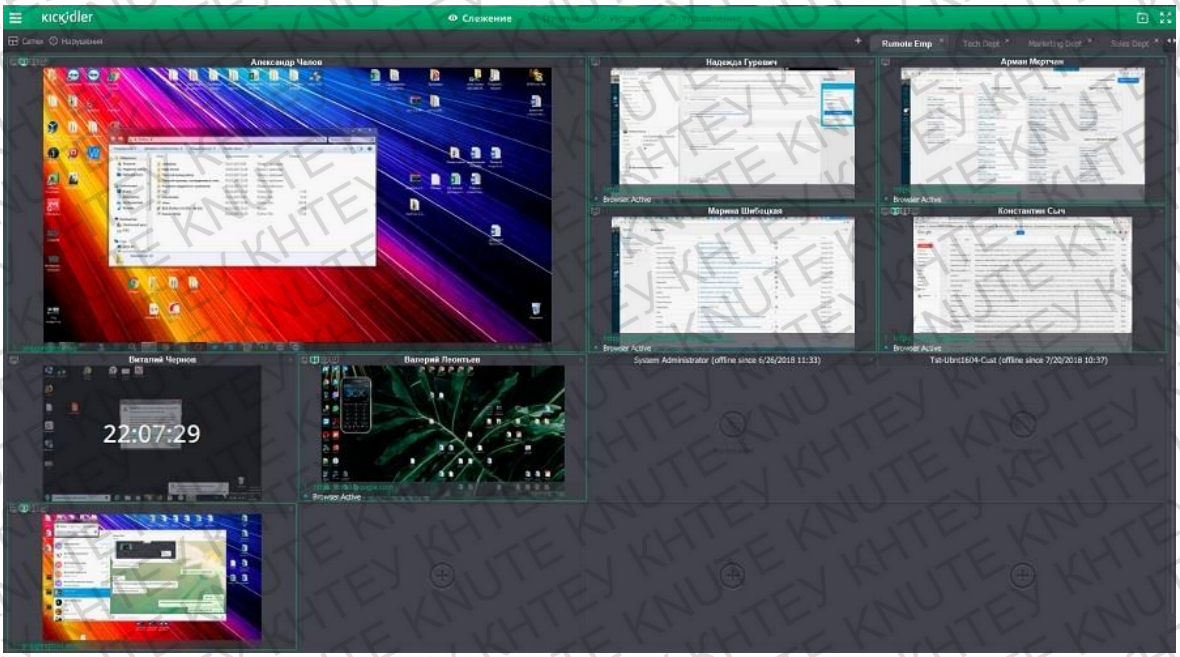

Рис. 1.1. Інтерфейс програми – Kickidler

Таблиця 1.1.

Характеристика програми – Kickidler

Характеристика	Опис характеристики
1	2
Принцип роботи	На комп'ютери співробітників встановлюються приховані агенти, які виробляють запис відео з екрану, натискань клавіатури і запущених додатків. Для кожної програми задається ефективність. Адміністратор може дистанційно керувати комп'ютерами користувачів. Основне вікно програми являє собою вид на монітори співробітників в реальному часі.
Гнучкий графік	У програмі відсутня можливість задавати графік
Заявки на відпустку відгул. Табель	Заявки та звіти для бухгалтерії відсутні
Звіти	Звіти відображають найбільш використовувані програми і сайти. Звіти неінформативні, вони не розраховані на контроль графіка роботи співробітників. При перегляді моніторів співробітників на екрані одночасно поміщається до 16 моніторів, що буде проблемою при необхідності моніторингу роботи співробітників в реальному часі в компаніях з великим штатом
ОС і БД	Сервер Kickidler працює під ОС Windows і з БД PostgreSQL. Є версія сервера під Linux, але в ній реалізований обмежений функціонал
Налаштування	Для роботи системи необхідно встановити сервер і агенти на комп'ютери користувачів. Для доступу до записаним відео використовується окрема програма «Viewer», яку необхідно встановити співробітникам, які будуть стежити за зібраними даними.

Продовження табл. 1.1

1	2
Робота через інтернет мережі	Для роботи програми необхідний доступ до my.kickidler.com, через який відбувається синхронізація сервера, грабер і вьюверов і перевірка ліцензії. При локальній установці, дану адресу - my.kickidler.com, потрібно ввести на локальний сервер ліцензування. Також присутня підтримка ssl.
Доступ до статистики	Надати користувачам доступ до статистики для самоконтролю неможливо. Доступ до статистики і записаним відео відбувається через програму Viewer. Для встановлення доступу до статистики для користувачів, потрібно в адмін панелі Viewer-а, вибрати користувача(ів) та встановити йому (їм) необхідній доступ.
Ціна	Хмарна версія: 182.00 грн на місяць за одного співробітника. 3640.03 грн за одного співробітника, без обмеження за часом. Локальна версія: Ціна за запитом.

ManicTime - дозволяє бути в курсі роботи своїх співробітників, надсилати точні звіти про хід роботи та керувати своїм часом краще, ніж будь-коли раніше. Автоматично записує період часу використання комп'ютера. Він запам'ятовує, які програми ви використовували та як довго. Також запам’ятовує, які веб-сайти відвідав працівник та над якими документами працював робітник. Усі ці дані допоможуть відстежувати робочий час.

Рис. 1.2. Інтерфейс програми – ManicTime

Таблиця 1.2.

Характеристика програми – ManicTime

Характеристика	**Опис характеристики**
1	2
Принцип роботи	Встановлений агент збирає дані про час роботи за ПК і використовуваних програмах (тільки з графічною оболонкою) і сайтах. Так само агент вміє робити скріншоти.
Гнучкий графік	У розкладі вказується час початку та кінця або кількість годин роботи в день. Можна створити кілька розкладів

Продовження таблиці 1.2

1	2
Заявки на відпустку відгул. Табель	Заявок немає, але є мітки якими можна відзначати свій час. З їх допомогою можна відзначати відгули і т.д. Є звіт «Відвідуваності», що відображає кількість відпрацьованих днів співробітника.
Звіти	Звітів не багато. Вони поділені на два типи: час роботи (початок / кінець, переробка ...), і продуктивність (використовувані веб сайти, програми, документи, статистика продуктивності).
Налаштування	При установці сервера необхідно вибрати з якою БД працювати. СУБД повинна бути налаштована до установки сервера (крім SQLite). В агента необхідно вказати адресу сервера куди відправляти статистику.
Робота через інтернет мережі	Для доступу до сервера необхідний відкритий порт 8080 (можна змінити). Для захисту з'єднання можна включити https, в якому необхідно буде задати своє сертифікат або використовувати стандартний.
Доступ до статистики	Є можливість надати доступ до статистики співробітникам через веб-інтерфейс. Так само є доступ до персональної статистики через клієнтську програму, якщо воно не в прихованому режимі. Є можливість давати права на доступ до статистики всього відділу.
Ціна	Ціна однакова для локальної та хмарної версії: 67$ за користувача, безстрокова.

1.3. Висновки до розділу 1

В першій частині першого розділу було визначено, що таке метод управління, які основні види. Описано як саме працює адміністративний

метод управління, на що цей метод поділяється і дано короткий опис кожного із цих методів, а також виділено плюси і недоліки даних методів.

В другій частині першого розділу було визначено, що таке робочий час на підприємстві і за допомогою яких засобів його можна контролювати. Було виділено 6 засобів контролю робочого часу та описано їх плюси та недоліки.

Для одного із цих засобів було наведено приклади програм та їх опис характеристик, також можливо помітити той факт, що всі програми різні, але принцип роботи фактично ідентичний. Різниця тільки в тому як ці програми встановлювати і наскільки об'ємний функціонал, який період експлуатації пропонує програма.

РОЗДІЛ 2
ПРОЕКТУВАННЯ БАЗИ ДАНИХ ТА СЕРВЕРУ

2.1. Вибір бази даних та її проектування

В ході аналізу технічного завдання, була визначена необхідність у створені бази даних для зберігання необхідної інформації. Було прийняте рішення використовувати таку базу даних, як MongoDB.

MongoDB – це документоорієнтована система управління базами даних, яка не потребує опису схеми таблиць. Застосовується в веб-розробці, зокрема, в рамках JavaScript-орієнтованого стека MEAN (MERN).

В цій БД – відсутнє таке поняття, як таблиця, замість неї використовується таке поняття, як колекція. Колекція зберігає в собі документи з визначеною структурою даних. Документ – це запис в колекцію, і він має JSON-но («Ключ» - «Значення») подібну структуру зберігання даних, тобто, строки.

Різниця між MySQL і MongoDb на даному етапі не велика, але в MongoDB – можливо зберігати, таким чином – «документ» в «документі», що дозволяє швидко отримати доступ до цих даних і не робити зайві запити до БД, тим самим збільшуючи продуктивність додатку (запит робиться один – отримуємо всі необхідні дані – віддаємо ці дані до клієнта).

Ключові переваги NoSQL баз в розподілених системах полягають в процедурах шаринга і реплікації.

Реплікація - це копіювання даних при їх оновленні на інші сервера.

Цей механізм дозволяє домогтися більшої відмовостійкості і масштабованості системи.

Прийнято виділяти два види реплікації: master-slave і peer-to-peer.

Шаринг - це поділ масиву інформації з різних вузлів мережі - коли кожен вузол відповідає тільки за певний набір даних і обробляє запити на читання і запис, що відносяться тільки до цього набору даних. Ця технологія використовувалася при роботі з реляційними базами даних в досить сирому вигляді - на рівні додатку створювалися незалежні бази даних, за якими розподілялися запити користувачів. Концепція NoSQL передбачає, що відповідальність за цей механізм буде лежати на базі даних, і шарінг буде здійснюватися автоматично.

Також, щоб робити запити до MongoDB, не потрібно знати мову SQL, тому що це NoSQL база даних. Для взаємодією з БД – можливо використовувати Mongo Shell (це інструмент для управління базою даних за допомогою консолі и консольних команд).

Існують також драйвери для взаємодії з БД, деякі з них представлені нижче:

1. Mongo-C#, Officlal C# - для розробників, які використовують мову для написання серверу C#.
2. Mongoose - для розробників, які використовують мову для написання серверу Node JS.
3. MongoDB PHP - для розробників, які використовують мову для написання серверу PHP.

Проектування бази даних починається з того, що потрібно визначити, які сутності потрібні для роботи веб-сайту:

1. **Користувачі** - сутність, яка включає в себе інформацію про користувача, який буде реєструватися.
2. **Організації (підприємства)** - сутність, яка включає в себе інформацію про організацію або підприємство.
3. **Ролі** - сутність, яка включає в себе інформацію про роль користувача,

який зареєструвався в системі.

4. **Відділи** - сутність, яка включає в себе інформацію про відділи (які будуть створюватися користувача в ході реєстрації нових робітників).
5. **Завдання** - сутність, яка включає в себе інформацію про завдання, які призначаються для робітників організації (підприємства).
6. **Файли** - сутність, яка включає в себе інформацію про файли, які зберігаються на сервері.
7. **Звіти** - сутність, яка включає в себе інформацію про звіти, які підкріпляються до певного завдання.
8. **Лист блокування** - сутність, яка зберігає в собі всі токени, які відробили свій термін.
9. **Токени** - сутність, яка зберігає в собі всі токени, які призначені для верифікації користувача у системі.
10. **Коментарі** - сутність, яка зберігає в собі всі коментарі, які призначені для певного завдання.

Після розробки даних сутностей та їх детального аналізу, було визначено, що потрібно створити такі колекції, як:

- ✓ *users* – колекція для зберігання даних про користувачів та робітників, також вміщує в себе масив *tasks*, в якому будуть знаходитися завдання працівників.
- ✓ *organizations* – колекція для зберігання даних про організації (підприємства)
- ✓ *roles* – колекція для зберігання даних про ролі користувачів.
- ✓ *departments* – колекція для зберігання даних про відділи, які існують в організації (підприємстві)
- ✓ *files* – колекція для зберігання даних файли.
- ✓ *reports* – колекція для зберігання даних про звіти, які створюються

робітниками.

- ✓ *tasks* – колекція для зберігання даних про завдання, вміщує в себе колекції: *comments*, *reports*.
- ✓ *blocklist* – колекція для зберігання токенів.
- ✓ *tokens* – колекція для зберігання токенів, які потрібні будуть для верифікації користувача.

Фізична модель даних (або проектування бази даних) — подання дизайну даних як реалізованого чи призначеного для реалізації у системі керування базами даних.

Так як використовується нереляційна БД, тому фізична модель (Рис.2.1.) буде відрнятися від фізичної моделі реляційної БД. Відмінність буде у зв'язках, також будуть присутні типи даних, які не входять до реліаційної БД. Всі таблички, які зображені у цій моделі, будуть описані в таблицях, яка будуть представлені нижче під рисунком.

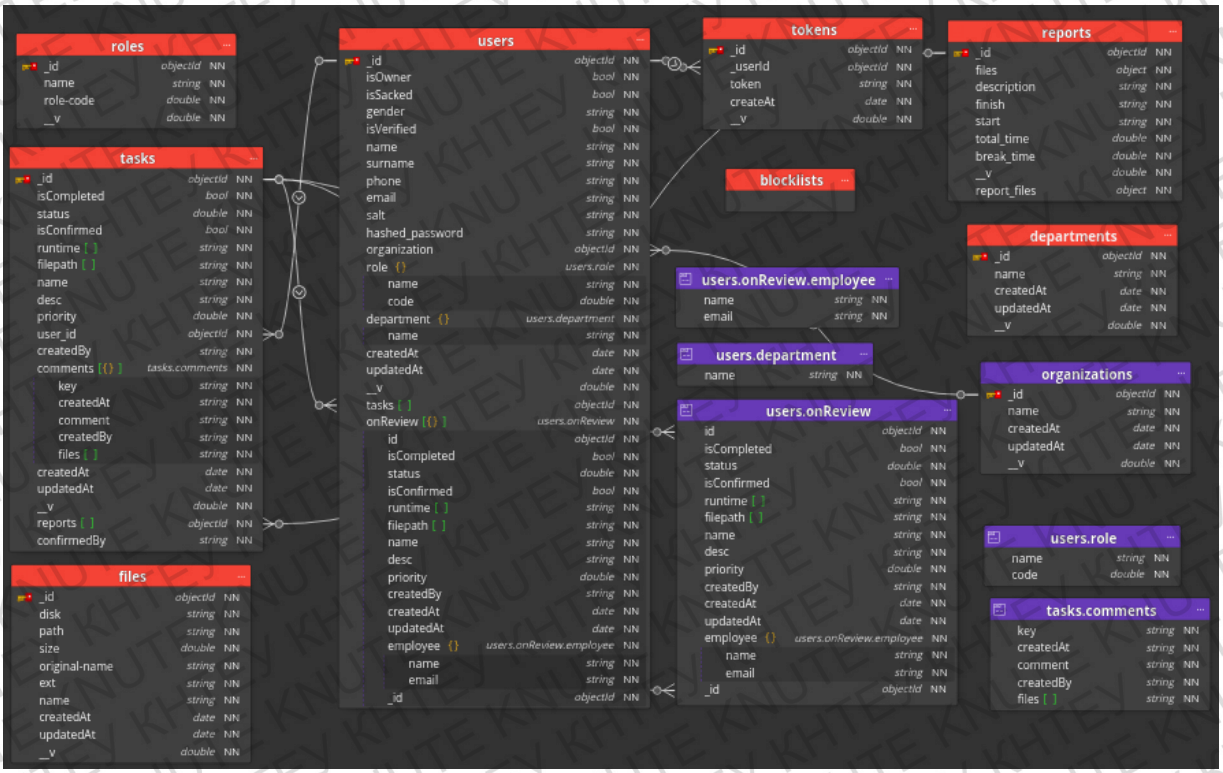

Рис. 2.1. Фізична модель БД

Таблиця 2.1.

Користувачі

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
isOwner	Boolean	-	Це засновник (організації) ?
isSacked	Boolean	-	Користувач звільнений ?
isVerified	Boolean	-	Користувач верифікований ?
email	String	-	E-mail користувача
name	String	-	Им'я користувача
surname	String	-	Прізвище користувача
phone	String	-	Номер телефону
gender	String	-	Стать користувача
salt	Number	-	Унікальний набір цифр (вик. Для шифрування паролю)
hashed_password	String	-	Зашифрований пароль
createdAt	Date	-	Дата створення запису
onReview	Array	-	Зберігає завдання, які були відправлені на перевірку до певного користувача.
organization	ObjectId	«Organization»	Зберігає **id** організації, в якій працює користувач
Tasks	Array [ObjectId]	«Tasks»	Зберігає **id-шники,** завдань, які були призначені певному користувачу.

Таблиця 2.2.

Організації

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
name	String	-	Назва організації

Таблиця 2.3.

Ролі

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
name	String	-	Назва ролі
role-code	Number	-	Код ролі (1-10)

Таблиця 2.4.

Відділи

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
name	String	-	Назва ролі
createdAt	datetime	-	Дата створення

Таблиця 2.5.

Файли

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
original-name	String	-	Справжня назва файлу
name	String	-	Назва файлу
ext	String	-	Розширення
disk	String	-	Місце зберігання
path	String	-	Шлях до файлу
size	Number	-	Розмір
createdAt	datetime	-	Дата створення

Таблиця 2.6.

Звіти

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
start	String	-	Початок роботи
finish	String	-	Завершення роботи
total_time	Number	-	Витрачено годин (на виконування завд.)
break_time	Number	-	Витрачено годин (на відпочинок)
description	String	-	Опис того, що було зроблено
report_files	Array	-	Файли які закріплена за звітом

Таблиця 2.7.

Завдання

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
name	String	-	Назва завдання
desc	String	-	Короткий опис
isCompleted	Boolean	-	Виконаний
status	Number	-	Етап
createdBy	String	-	Ким створенний
isConfirmed	Boolean	-	Підтверджений?
confirmedBy	String	-	Ким підтверджений
runtime	Array	-	Проміжок часу на виконання
priority	Number	-	Пріоритет
filepath	Array	-	Закріплені файли (шляхи до них)
user_id	ObjectId	«User»	За ким закріплене
comments	[Object]	-	Коментарі до завдання
reports	[ObjectId]	«Report»	Звіти до завдання
createdAt	Datetime	-	Дата створення

Таблиця 2.8.

Лист блокування

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
token	String	-	Токен
createdAt	Date	-	Дата створення

Таблиця 2.9.

Токени

Атрибути	Тип	Зв'язок з іншою колекцією	Опис
_id	ObjectId	-	Унікальний ідентифікатор
token	String	-	Токен
createdAt	Date	-	Дата створення

2.2. Опис архітектури додатку

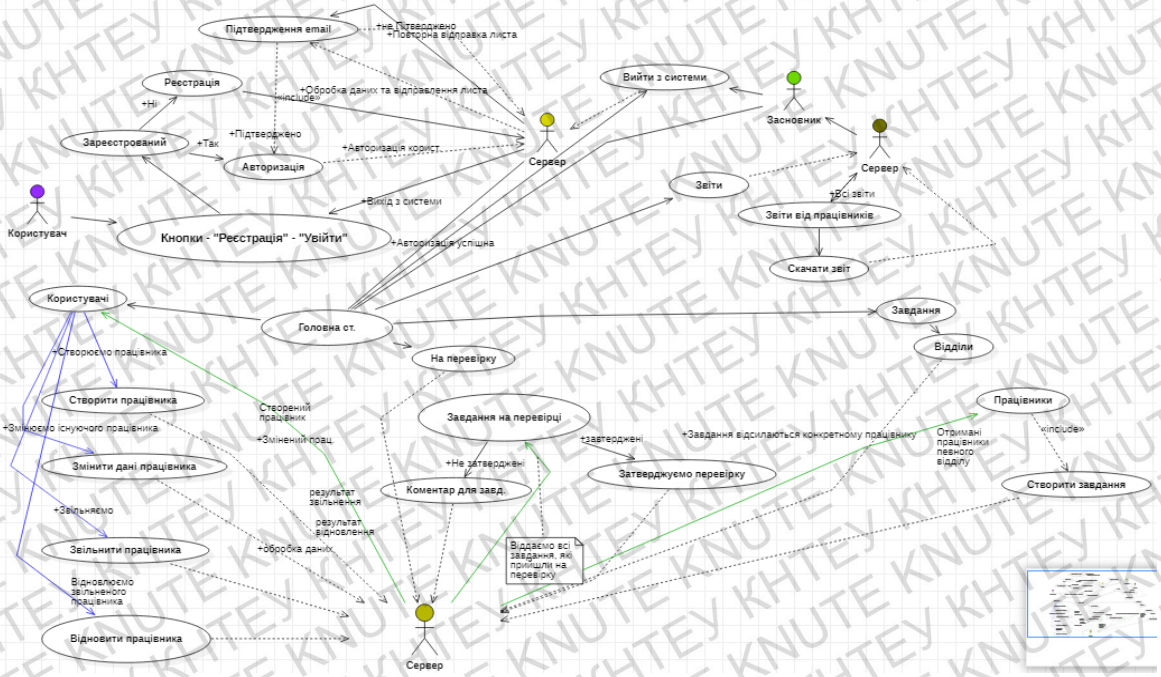

Рис. 2.2. Графічний відображення архітектури

На схемі зображено процедуру реєстрації та авторизації, яку буде проходити користувач. Користувач потрапляє на головну сторінку додатку на якій вказана інформація про сам додаток та деякі його можливості, і далі відбуватимуться такі кроки:

1. Реєстрація користувача та нової організації – користувач заходить на сторінку реєстрації, заповнює форму, вказуючи деякі дані, а також назву його організації (підприємства). Форма перевіряється і дані відсилаються на сервер для подальшої перевірки, збереження цих даних у БД, а також відправлення листа на електрону пошту (користувача) для верифікації його профілю.

2. Підтвердження email – користувач переходить у свою пошту, на яку було надіслано лист. Заходить у лист та підтверджує свій email натисненням на кнопку «Підтвердити». Email підтверджується, якщо у посиланні не вийшов термін придатності, але якщо термін вийшов, то користувач може повторно вислати собі листа для верифікації і підтвердити email.

3. Авторизація – авторизуватися користувач може тільки тоді, коли його email був підтверджений. Якщо email не підтверджений, то його у систему не пустить. Для авторизації потрібно ввести логін і пароль, дані перевіряються і відправляються на сервер для подальшої перевірки.

4. При успішній авторизації користувача перекидає на головну сторінку додатку, на якій відображені дані користувача, кнопка с можливістю редагування даних профілю. Також користувачу надається роль «Засновник».

На сторінці авторизованого користувача будуть відображені такі пункти меню: «Головна», «Користувачі», «Завдання», «Звіти», а також якщо у користувача призначена роль – «Засновник», «Керуючий», «Тимчасовий керуючий», то у нього відображається пункт меню «На перевірку».

На рисунку зображено можливості сторінки «Користувачі». На сторінці відображається загальна інформація про кількість працівників організації (підприємства). А також присутня інформація, яка кількість: «Робітників», «Керуючих» та «Засновників» зареєстровано у системі (даної організації). Для проведення операцій, які будуть розглянуті нижче, потрібно щоб у користувача була присвоєна одна із ролей: «Керуючий», «Тимчасовий керуючий» або «Засновник». Розглянемо, які будуть можливості на цій сторінці:

1. Створити працівника – працівник буде створюватися дуже легко, просто викликаємо модальне вікно с формою, заповнюємо форму, перевіряємо дані та відправляємо їх на сервер для перевірки та зберігання в БД.
2. Створити відділ – можливо в том ж модальному вікні, там де і працівників. Це буде зроблено таким чином, тому що при створені працівника потрібно вибрати відділ, а якщо його нема, то його можливо буде створити і вибрати не закриваючи модальне вікна «Створення працівника».
3. Змінити дані працівника – можливо буде в другому модальному вікні, потрібно буде вибрати потрібного працівника в таблиці та натиснути на кнопку «Змінити», всі дані які можливо буде змінювати, будуть завантажуватися з серверу. Після їх завантаження і відображення в модальній формі, ці дані можливо буде змінити та зберегти зміни.
4. Звільнення – вибрати працівника в таблиці, натиснути на червону кнопку напроти цього працівника і підтвердити його звільнення з організації (підприємства).
5. Відновлення – вибрати працівника в таблиці, натиснути на зелену кнопку напроти цього працівника і підтвердити його відновлення в

організації (підприємстві).

В залежності яка роль у користувача, у на сторінці завдання будуть відображатися такі дані:

1. Роль «Засновник» - на сторінці буде можливість вибрати один відділ із списку. В залежності від відділу, будуть відображатися певна кількість працівників і цим працівникам можливо буде дати завдання.
2. Роль «Керуючий» або «Тимчасовий керуючий» - на сторінці будуть відображатися працівники тільки того відділу, в якому цей працівник знаходиться. Наприклад: «Керуючий» з відділу «Бухгалтерія» - то працівники будуть відображатися для цього керуючий, також с відділу «Бухгалтерія». Обрати другий відділ неможливо так, як буде не вистачати прав. Для кожного із працівників, які будуть відображені, можливо буде створити завдання для них.
3. Роль «Працівник» - не буде можливості: вибрати відділ і створювати завдання для інших працівників. Буде можливість - виконувати завдання, які були надані керівниками; відправляти на перевірку завдання; завершати ці завдання.

«На перевірку» - ця кнопка буде відображена тільки у користувачів, які матимуть роль «Тимчасовий керуючий» і вище. При переході на цю сторінку, будуть відображатися завдання, які були відправлені на перевірку іншими працівниками. Принцип роботи з цією сторінкою буде такий:

1. Керуючий продивився якість виконання завдання.
2. В залежності наскільки правильно було виконано завдання, буде писати коментар та підкріплювати до цього коментаря деякі файли, які допоможуть працівникові виправити помилки.
3. Якщо якість виконаного завдання задовольнило керівника, то він може підтвердити виконання цього завдання.

На сторінці «Звіти», залежно від ролі, будуть відображатися або всі звіти працівників певного відділу, або буде можливість вибрати завдання і підкріпити до нього звіт з файлами.

2.3. Вибір інструментарію для розробки

Виходячи з того, яка стоїть задача, було прийняте рішення використовувати такі технології:

Front-end частина:

- React JS – декларативна, гнучка, javascript бібліотека для розробки користувацьких інтерфейсів. Ця бібліотека використовує компонентний підхід для вирішення конкретних завдань. Бібліотека проста у використані.
- Redux JS – бібліотека для javascript з відкритим вихідним кодом, призначена для керування станом додатку. Ця бібліотека дозволяє централізовано зберігати дані, які потрібні для того щоб відобразити компоненти з даними усередині. Також дозволяє відділити логіку від самого компонента, логіка окремо знаходиться і змінює дані в централізованому сховищі, які передаються в компоненти.
- Webpack – це статичний модуль bundler для сучасних програм javascript. Коли цей інструмент обробляє програму, то створюється внутрішній графік залежностей, який відображає кожен модуль, необхідний вашому проекту, і генерує один або більше пакетів

Backend частина:

- Node Js – javascript, який виконується на стороні сервера. Використовує керовану подіями, неблокуючу модель вводу-виводу, яка робить її легкою і ефективною. Також присутня пакетна екосистема, npm, яка є найбільшою екосистемою бібліотек с відкритим вихідним кодом в світі.

- Express JS – це платформа веб-додатків, яка допомагає організовувати веб-додаток в архітектурі MVC на стороні сервера. Також це фреймворк, який має в собі все необхідне для запуску сервера.
- JSON Web Tokens – це система авторизації, яка визначає компактний і автономний спосіб безпечної передачі інформації. Важлива інформація знаходиться в самому токені. Токен – набір символів – результатом якого, є шифрування даних. Розшифрування токену проходить на сервері і дані, які з нього витягуються і є важливими даними, які були зашифровані в цьому токені.

Інший сервіс:

- MailDev – це локальний сервіс для перевірки правильності згенерованих електронних листів, під час розробки проекту.

2.4. Висновки до розділу 2

В даному розділі:

1. Була описана база даних, яка буде використовуватися в подальшому для розробки проекту. Була сформована архітектура бази даних, також було визначено, які дані потрібно зберігати і в якому вигляді вони повинні бути у колекціях.
2. Була описана архітектура додатка, висвітлені основні процеси, які повинні бути враховані при розробці.
3. Було обрано технології для успішного виконання поставленого завдання.

РОЗДІЛ 3
РОЗРОБКА СЕРВЕРА. РОЗРОБКА КОМПОНЕНТІВ КЛІЄНТСЬКОЇ ЧАСТИНИ

3.1. Створення структури серверної частини

Як Розробка серверу починається з того, що потрібно встановити допоміжні бібліотеки. Встановлюються вони за допомогою пакетного менеджера, який йде в комплекті з Node JS – npm. Було встановлено наступні бібліотеки:

1. *express* - бібліотека для побудови сервера.
2. *express-validator* – бібліотека, яка використовується для валідації даних, які надходять з клієнта. Валідація підкріплюється до *route* у вигляді 2-х *middlewares,* 1-й *middleware* провіряє дані і якщо є помилки валідації, то вони надходять у 2-й *middleware*. Другий *middleware* виконує роль сповіщувача, який сповіщає користувача про те що сталася помилка валідації і потрібно ввести дані знову у потрібно форматі.
3. *express-zip* – ця бібліотека потрібна, щоб брати декілька файлів с сервера, поміщати їх в архів та віддавати цей архів користувачу.
4. *mongoose* – це драйвер для роботи з базою даних MongoDB.
5. *multer* – бібліотека, яка валідує, розміщує файли на сервері в папці.
6. *nodemailer* – бібліотека для генерування та відправлення листів до користувача.
7. *nodemon* – зручна бібліотека, яка встановлюється лише на етапі розробки. Вона слідкує за станом файлів і якщо десь щось.

змінилося, то ці всі зміни зберігаються и сервер автоматично перезапускається.

8. *Jsonwebtoken* – про цю бібліотеку було сказано, ще в другому розділі.
9. *dotenv* – бібліотека, яка допомагає зчитувати дані у файлах типу *.env*, *.env.local* і т.д.

Бібліотеки вибрані, тепер потрібно структурувати сервер, а точніше файли, які знаходяться в ньому. Це необхідно робити на початковому етапі для того щоб в подальшому, коли проект буде розростатися і мати велику кількість файлів, не виникало питань де що знаходиться, і як це все працює. Правильно структура папок та їх назви допомагають інтуїтивно знати потрібний нам файл.

Для цього проекту була обрана така структура папок:

- **controllers** – будуть приймати запит та відправляти його на сервіс.
- **emails** – будуть містить в собі файли з шаблонами листів, які будуть відправлятися до користувача в ході роботи. А також містить файл конфігурації.
- **files** – будуть зберігатися всі файли користувачів.
- **models** – будуть міститися файли, які описують поля всередині колекції.
- **services** – файли, які будуть приймати дані від контролера, обробляти їх, працювати з базою даних та відправляти відповідь до користувача з деякими даними.
- **validators** – файли, які містять інформацію про поля що потрібно валідувати, яке повідомлення потрібно відправити в разі помилкової валідації.
- **middlewares** – файли, для фільтрації HTTP запитів, наприклад, якщо не зареєстрований користувач спробує перейти по *route* який захищений, то *middleware* не дасть доступ до захищеної сторінки,

тому що користувач не зареєстрований.

- **routes** – файли, які містять в собі маршрути по яким будуть здійснюватися запити з клієнта на сервер.
- **seeds** – файли для генерації випадкових даних і заповнення ними колекції в базі даних.
- **utils** – містять файли з корисними функціями, які по розмірам маленькі, але вони використовуються досить часто.
- **lang** – містить папки з різними мовами.

Наступний крок - це створення файлу з розширенням .env – цей файл буде містить в собі глобальні зміні, які можливо буде використати у всіх файлах серверу, а в разі потреби змінити значення змінної і ці значення зміняться у всіх місцях де використовувалась дана зміна, приклад вмісту файлу (Рис. 3.1.)

```
NODE_ENV= # development or production
SERVER_URL= # http://localhost:5000
CLIENT_URL= # http://localhost:3000
PORT= # 5000

MONGODB_URI= # mongodb://{username}:{password}@127.0.0.1:27017/{database}
ACCESS_TOKEN_SECRET # exapmple --> 65247efe2c9ad08d06d94fbb36661090f7369f7c
EXPIRES_TIME= # 3600

EMAIL_PASS= # your password
EMAIL_FROM= # your email
```

Рис. 3.1. Приклад *.env* файлу

3.2. Поетапне створення реєстрації на сервері

Бібліотеки вибрані, структура побудована, .env файл сформований, тепер потрібно створити файл ініціалізації, для того щоб запустити сервер. Файл містить всі шляхи (routes), підключення до бази даних, глобальні фільтри (middlewares), а дані для конфігурування сервера можливо взяти з

.env файлу (змінна - port, server_url). Детальніше вміст цього файлу можливо буде побачити у Додаток А.

Наступний крок – це створення файлів зі шляхами (routes). Шляхи реєструються для того, щоб користувач міг по ним зробити запит до сервера та отримати деякі дані. Реєстрації шляхів наведено нижче – auth.route.js (Рис. 3.2.).

```
const express = require('express')
const router = express.Router()
const requireLogin = require('../middlewares/auth.middleware')
const { register, login, logout, refreshToken } = require('../controllers/auth.controller')

// validation middleware
const { runValidation } = require('../middlewares/run-validation.middleware')
// validators
const { registerValidator, loginValidator } = require('../validators')

router.post('/register', registerValidator, runValidation, register)
  .post('/login', loginValidator, runValidation, login)
  .post('/logout', requireLogin, logout)
  .post('/refresh', refreshToken)

module.exports = router
```

Рис. 3.2. Реєстрація шляхів для авторизації, реєстрації користувачів

Якщо подивитись на рисунок уважніше, то можливо помітити, що потрібно переходити до наступних кроків – це створення контролерів, сервісів для контролерів, фільтрів та валідаторів.

Спочатку створюється контролер для приймання даних, які прийшли з запиту від клієнта. В контролері дані конвертуються, та передаються на подальшу обробку до сервісу. Створений контролер – *auth.controller.js* (Рис. 3.3.). Детальніше цей файл можливо побачити у Додаток Б.

```
const User = require('../models/user.model')
const { saveNewUserAndOrganization, getUserAndLogin, saveTokenAndSendNewToken } = require('../services/auth.service')
const jwt = require('jsonwebtoken')

const expiresTime = parseInt(process.env.EXPIRES_TIME)

exports.register = async (req, res) => {
  const user = await User.findOne({ email: req.body.email })

  if (user) {
    return res.status(400).json({
      error: `Данный email уже используется!`
    })
  }

  await saveNewUserAndOrganization(req, res)
}
```

Рис. 3.3. Контролер – *auth.controller*

В цьому контролері функція *register* звертається до колекції *User*, перевіряється наявність *email* який прийшов від користувача, і в залежності, якщо *email* вже існує в системі, то контролер відправляє відповідь з повідомленням про те що такий *email* існує у системі і потрібно обрати інший для завершення реєстрації. А якщо *email* в БД не існує, то функцію викликає функцію, яка приходить с сервісу. Також в цьому прикладі можливо побачити приклад використання змінної з *.env* файлу - `process.env.EXPIRES_TIME`.

Але перед тим як створювати сервіс, необхідно створити модель даних *users*, точніше колекцію з відповідними полями. Створення моделі проводиться таким чином. С драйвера *mongoose* імпортуються – *model, Schema, Types*. Потім створюється нова схема. В схемі вказуються які повинні бути у документі.

```
const { model, Schema, Types } = require('mongoose')
const { hashedPassword } = require('../utils/hashed-password')

const userSchema = new Schema({
  name: { type: String, trim: true, required: true, max: 32 },
  surname: { type: String, trim: true, max: 32 },
  email: { type: String, trim: true, required: true, unique: true, lowercase: true },
  hashed_password: { type: String, required: true
  },
  phone: String,
  isOwner: { type: Boolean, required: true, default: false
  },
  isSacked: { type: Boolean, required: true, default: false },
  gender: { type: String, default: 'unknown' },
  department: { type: Object },
  role: { type: Object },
  isVerified: { type: Boolean, default: false },
  salt: String,
  tasks: [{ type: Types.ObjectId, ref: 'Task' }],
  onReview: [{ type: Object }],
  resetPasswordLink: { data: String, default: '' },
  organization: { type: Types.ObjectId, ref: 'Organization' }
}, { timestamps: true })
```

Рис. 3.4. Схема *user.model.js*

В цій схемі присутні такі поля:

- *name* – тип даних String, поле обов'язкове для заповнення, максимальна кількість символів 32.
- *surname* – тип даних String, максимальна кількість символів 32.
- *email* – тип даних String, поле обов'язкове для заповнення, повинно бути унікальним, приводиться до нижнього регістру.
- *hashed_password* - тип даних String, поле обов'язкове для заповнення
- *phone* – тип даних String
- *isOwner* – тип даних Boolean, поле обов'язкове для заповнення, значення за замовчуванням «*false*»
- *isSacked* - тип даних Boolean, поле обов'язкове для заповнення, значення за замовчуванням «*false*»

- *gender* - тип даних String, значення за замовчуванням «*unknown*»
- *department* – тип даних Object
- *role* - тип даних Object
- *isVerified* - тип даних Boolean, значення за замовчуванням «*false*»
- *salt* - тип даних String
- *tasks* – масив який складається за ObjectId, зв’язок з колекцією «Task»
- *onReview* – масив який складається за об’єктів.
- *resetPasswordLink* – тип даних String, значення за замовчуванням – “”
- *organization* – тип даних ObjectId, зв’язок з колекцією «Organization»

Перед тим як перейти до створення сервісу, потрібно створити файл для валідації даних. Валідація даних проводиться на сервері, тому що в основному валідація на стороні клієнта не дає стовідсоткової гарантії, що дані які вводить користувач будуть правильні. Для щоб зареєструвати користувача, нам потрібні такі дані:

- *name* – ім’я користувача
- *organization* – назва організації (підприємства)
- *email* – електрона пошта
- *password* – пароль користувача

Всі інші дані, такі як: *department, role, gender, isOwner, isSacked, isVerfied* – встановлюються автоматично на сервері.

```
const { check } = require('express-validator')

exports.registerValidator = [
  check('name')
    .not()
    .isEmpty()
    .withMessage('Пожалуйста, введите ваше имя!'),
  check('organization')
    .not()
    .isEmpty()
    .withMessage('Пожалуйста, введите название организации!'),
  check('email')
    .isEmail()
    .withMessage('Введен неверный E-mail'),
  check('password')
    .isLength({ min: 6 })
    .withMessage('Пароль должен содержать 6 цифр, символов или более')
]

exports.loginValidator = [
  check('email')
    .isEmail()
    .withMessage('Введен неверный E-mail'),
  check('password')
    .isLength({ min: 6 })
    .withMessage('Пароль должен содержать 6 цифр, символов или более')
]
```

Рис. 3.6. Валідатор – *auth.validator.js*

В даному файлі (Рис.3.6.) показано два валідатори – *register, login validators*. Валідатори складаються з масиву, всередині масиву послідовно викликаються такі функції:

- *check('email')* – функція, яка бере значення поля «Email» і передає в наступний метод
- *isEmail()* – отримав значення і валідує його за допомогою регулярного вираження, що перевіряє присутній у значені таких

символів – @, ., .com і т.д. Якщо валідація пройшла успішно то в масив нічого не вертається, а якщо не успішно, то в масив вертається повідомлення, яке буде відправлене до користувача.

- *withMessage* – в цьому методі можливо записати свою версію повідомлення, яке буде відправлятися до користувача в разі помилки валідації даних.

Перейдемо до створення сервісів. Сервіси приймають дані, звертаються до БД, і вертають деяку відповідь до клієнта. Також щоб сервіс міг відсилати листи, потрібно підключити *nodemailer* до проекту, та настроїти конфігурацію.(Рис. 3.6.) Сервіс – *auth.service.js* вказаний (Рис. 3.8. та Рис. 3.9.). Код сервісу знаходиться у Додаток В.

```
const nodemailer = require('nodemailer')

module.exports =  nodemailer.createTransport({
  host: 'localhost',
  port: 25,
  ignoreTLS: true,
  auth: {
    user: '',
    pass: ''
  },
})
```

Рис. 3.6. Конфігурація пошти

Але для відправки листа, цієї конфігурації не достатньо. Потрібно створити шаблон листа, який буде відправлятися кожному користувачу, який пройшов реєстрацію і очікує підтверджувального листа.

```
module.exports = function (email, link, name) {
  return {
    to: email,
    from: process.env.EMAIL_FROM,
    subject: 'Подтверждение почты',
    html: `
     <table style="width: 100% !important;height: 100%; background: #f8f8f8;">
      <tr>
        <td style="display: block !important; clear: both !important; margin: 0 auto !important; max-width: 580px !important;">
            <!-- Message start -->
            <table style="width: 100% !important; border-collapse: collapse; ">
                <tr>
                    <td align="center" style="padding: 80px 0; background: #6c63ff; border-radius: 10px;">
                        <h1 style="
                        line-height: 1.25; font-size: 24px;
                        margin: 0 auto !important; max-width: 90%;
                        text-transform: uppercase; color: #fff !important">
                          Добро пожаловать в приложение Work Time Manager
                        </h1>
                    </td>
                </tr>
```

Рис. 3.7. Фрагмент шаблону – підтверджувальний лист

```
/**
 * saveNewUserAndOrganization. Create new user, organization and save them in DB.
 * @param {object} req
 * @param {object} res
 * @return {*}
 */
exports.saveNewUserAndOrganization = async (req, res) => {
  const { name, email, organization, password } = req.body

  try {
    const findRole = await Role.findOne({ 'role-code': 3 })
    const newUser = new User({ name, email, password, isOwner: true, role: { name: findRole.name, code: findRole['role-code'] } })
    const department = await Department.findOne({ name: 'администрация' })

    if (!department) {
      const newDepartment = new Department({ name: 'администрация' })
      newUser.department = { name: newDepartment['name'] }
      newDepartment.save()
    } else {
      newUser.department = { name: department['name'] }
    }

    const newOrganization = new Organization({ name: organization })
    newUser.organization = newOrganization._id

    const token = new Token({
      _userId: newUser._id,
      token: generateToken({ email }, { expiresIn: 86400 }),
      expireAfterSeconds: 86400
    })
```

Рис. 3.8. Сервіс – *auth.service*. Частина перша

```
    const token = new Token({
      _userId: newUser._id,
      token: generateToken({ email }, { expiresIn: 86400 }),
      expireAfterSeconds: 86400
    })

    await newUser.validate()
    await token.validate()
    await newOrganization.validate()

    const link = `${process.env.CLIENT_URL}/confirm/${token.token}`

    await emailTransporter.sendMail(registrationMail(email, link, name))

    token.save()
    newOrganization.save()
    newUser.save()

    res.status(200).json({ success: 'Регистрация прошла успешно. На Ваш электронный адрес отправлено письмо для подтверждения аккаунта!' })
  } catch (err) {
    return res.status(500).json({ msg: err.message })
  }
}
```

Рис. 3.9. Сервіс – *auth.service*. Частина друга

В сервісі auth – потрібно встановити роль, відділ для користувача, тому проводиться звернення до колекцій roles, departments. Роль і відділ присвоюється новому користувачу, за замовчуванням роль у нового користувача – це «Засновник», а відділ – «Адміністрація». Також створюється нова організація, назву якої ввів користувач у формі «Реєстрація».

Потім генерується новий токен (дані, що шифруються – email) у якого термін придатності 1 день. Токен зберігаються у колекцію tokens. Через 24 години – цей токен видаляється з колекції автоматично, тому що термін придатності вийшов. Також після створення токену, документи, які щойно були створені – валідуються, а також користувачу відправляється лист на електрону скриньку с проханням підтвердити свою електронну адресу. А нові дані, після валідації і відправки листа, зберігаються у колекції, і користувачу відсилається повідомлення про те, що він успішно пройшов реєстрацію, і йому потрібно підтвердити його електронну пошту.

Користувач повинен буде перейти до електронної скриньки і підтвердити свою електрону пошту протягом 1- й доби, з моменту її

реєстрації у системі. Якщо користувач спробує підтвердити свій електрону пошту після того, як у посиланні вичерпався термін придатності, то йому буде вислано повідомлення про те, що термін придатності вичерпано і необхідно вказати свою електрону адресу в спеціальній формі та вислати собі повторно листа для підтвердження.

Після того як користувач підтвердив свою електрону адресу, його перенаправляє на сторінку «Авторизація» і він тепер має право авторизуватися у системі.

3.3. Розробка клієнтської частини

Переходимо до клієнтської частини в якій потрібно також визначитися з бібліотеками:

- *React* – це фундаментальна бібліотека за допомогою, якої буде створюватися весь інтерфейс користувача.
- *Redux* – потрібна для контролювання, оновлення стану додатку.
- *React-redux* – так як *redux,* це окрема JavaScript бібліотека, тому ця бібліотека необхідна для того щоб *react* працював з *redux*
- *react-router-dom* - необхідна для налаштування шляхів на стороні клієнтської частини.
- *prop-types* – необхідна перевірки типів даних які знаходяться у властивостях компонентів.
- *Antd* – бібліотека з готовими компонента, які можливо використовувати у своїх проектах.
- *Axios* - призначена для виконання HTTP-запитів, заснована на Проміс. Вона підходить для використання в середовищі Node.js і в браузерних додатках. Бібліотека підтримує всі сучасні браузери, і, в тому числі, IE8 +.

Головна сторінка - це «обличчя» кожного проекту, важливо, щоб ця сторінка виглядала якомога краще. Для цього використовуються ілюстрації, короткий опис продукту. Також є можливість переходу на сторінки «Реєстрація» та «Авторизація».

Рис. 3.10. Головна сторінка (користувач не авторизований)

Користувач повинен зареєструватися на сайті, для того щоб мати змогу зайти у додаток. Реєстрація нового користувача проводиться наступним чином: заповнюються всі поля у формі, відправляються дані, та отримується відповідь від серверу. Відповідь відображені у вигляді повідомлення. Приклад реєстрації (Рис. 3.11.).

Після успішної реєстрації користувачу на електрону адресу приходить лист с повідомленням про успішну реєстрацію і кнопкою «Підтвердити» електронну пошту. Після підтвердження електронної адреси, користувача перенаправляє на сторінку «Авторизація».

Авторизація проводиться стандартним способом – вводиться email і пароль. Дані відправляються на сервер, вони перевіряються і якщо все вірно і користувача знайшло в БД, то його авторизує, і йому присилається токен.

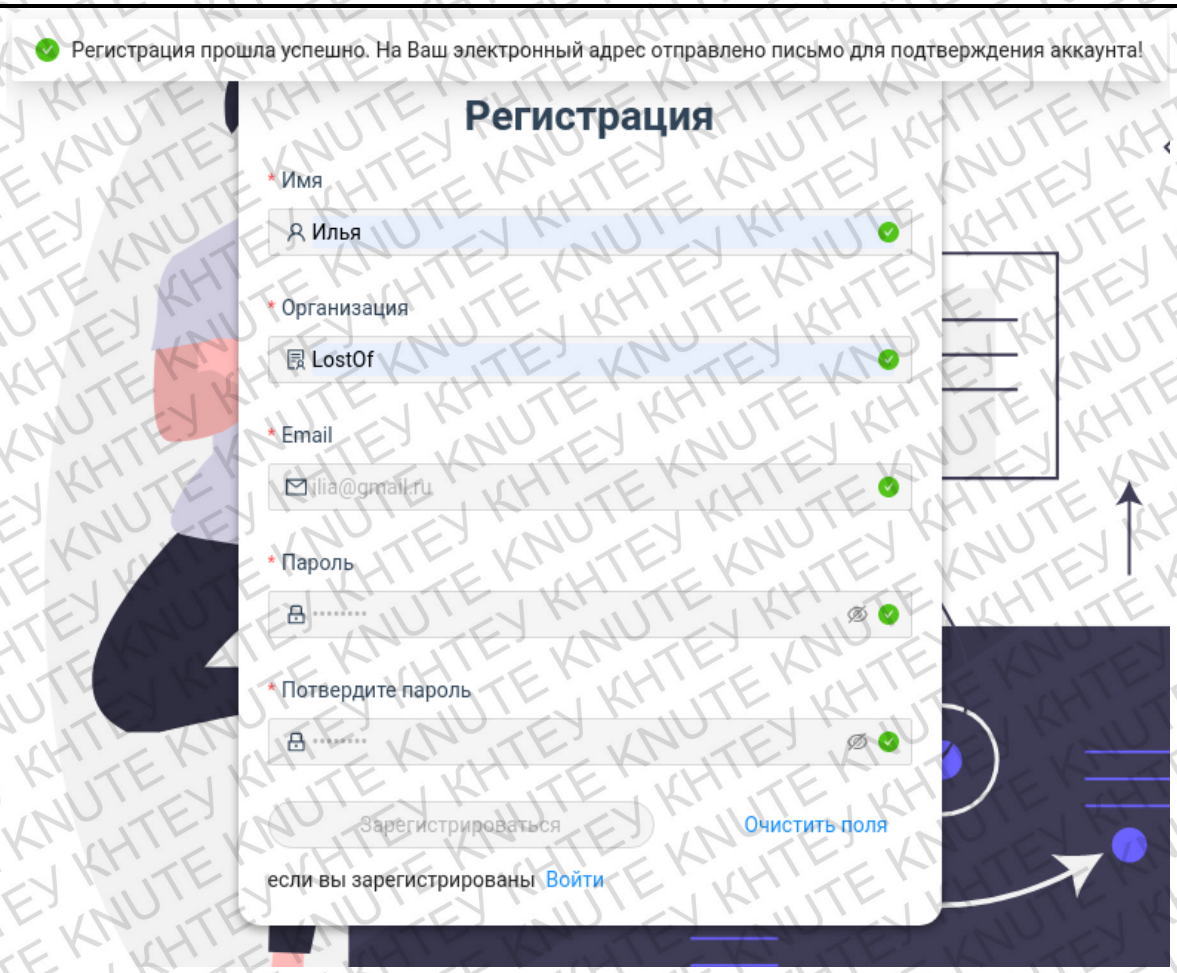

Рис. 3.11. Реєстрація нового користувача

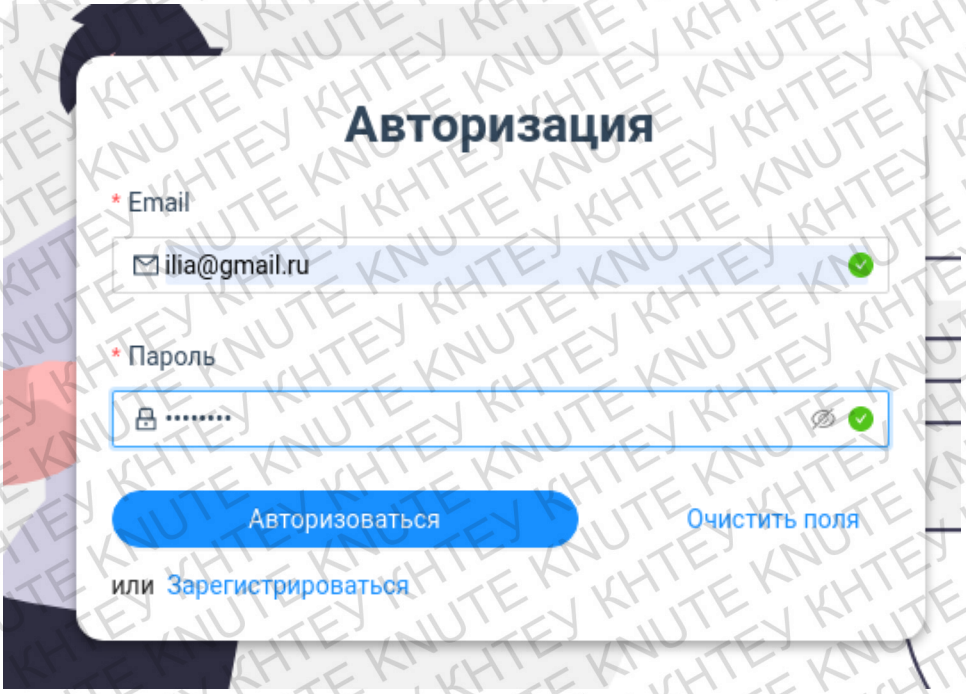

Рис. 3.12. Авторизація

Після успішної авторизації, користувач потрапляє на головну сторінку (Рис.3.13.) додатку. На головній сторінці знаходиться інформація про користувача, та інформація про можливості, які в нього є у цьому додатку. Також є можливість переходу на сторінки «Працівники», «Завдання», «Звіти» та «На перевірку» (на дану сторінку можуть переходити тільки працівники у яких роль: «Засновник», «Керуючий», «Тимчасовий керуючи»).

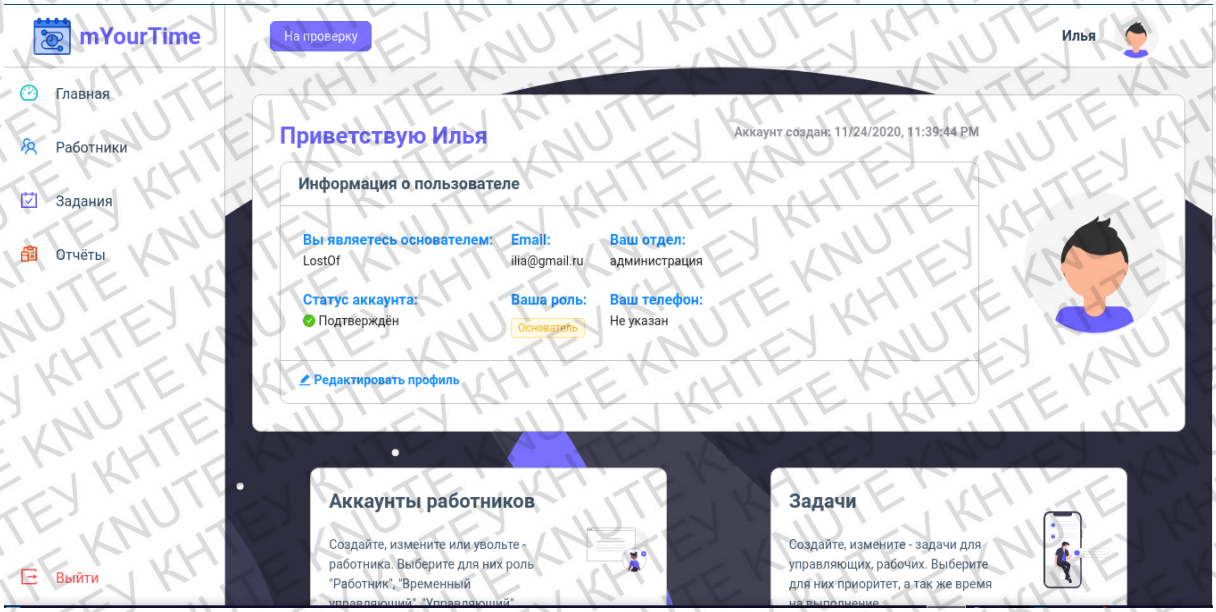

Рис. 3.13. Головна сторінка (Авторизований користувач)

Користувач може змінити деякі свої дані – натиснувши кнопку «Редагувати профіль», його перенаправить на сторінку (Рис. 3.14.) з формою, в якій користувач міняє дані, які він хоче змінити и зберігає їх.

Також на сторінці «Редагування» присутня можливість змінити пароль, для цього потрібно – підтвердити свій старий пароль, при успішному підтверджені активується 2-га форма для вводу нового паролю і користувач має можливість його успішно змінити.

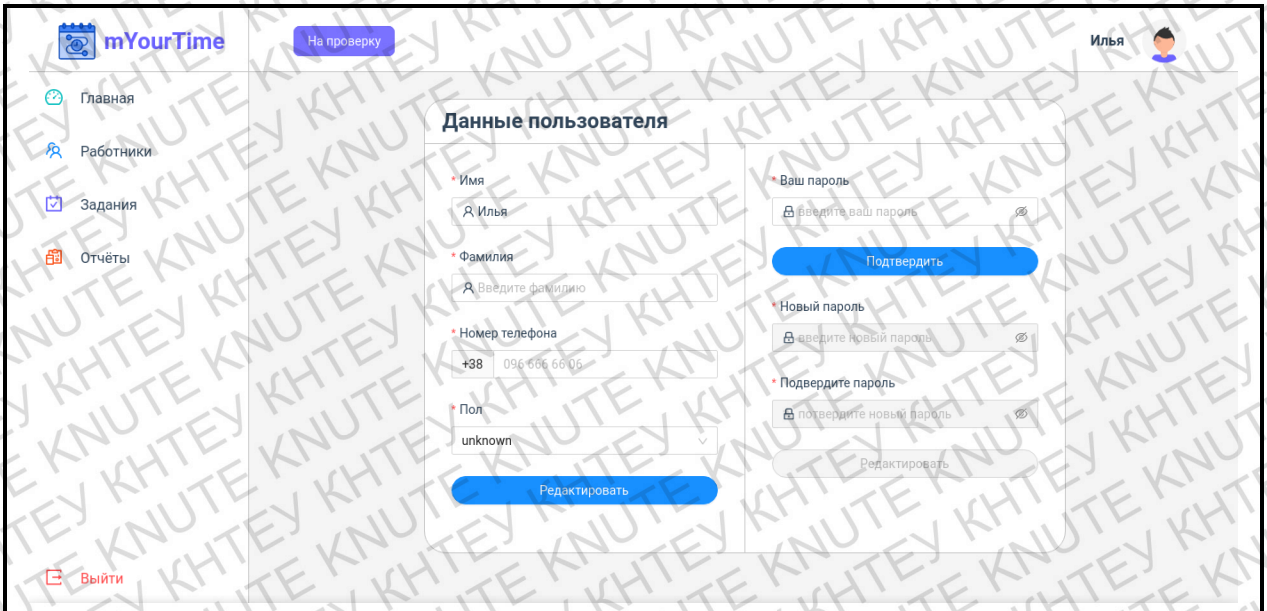

Рис. 3.14. Сторінка «Редагування профілю»

Перейшовши на сторінку «Працівники» (Рис.3.15.), користувач побачить інформацію про те скільки людей зареєстровано в його організації (підприємстві). Також є таблиця в якій відображаються усі працівники. Присутні такі кнопки: Створити працівника, Змінити дані працівника, Звільнити працівника, Поновити працівника (якщо працівника звільнили).

Рис. 3.15. Сторінка «Працівники»

На сторінці «Завдання» (Рис. 3.16.) в залежності від ролі працівника присутні такі можливості:

- Засновник – вибирати відділ та створювати завдання усім працівникам своєї організації.
- Керуючий або Тимчасовий керуючий – створювати завдання усім працівникам, які знаходяться у їх відділі та виконувати свої завдання.
- Працівник – виконувати завдання, які були призначені керуючими або засновником. Також відправляти завдання на перевірку, читати коментарі, які були написані під певним завданням.

Рис. 3.16. Сторінка «Завдання»

На сторінці «Звіти» (Рис. 3.17.) – присутня можливість вибрати одне із завдань. Дані про вибране завдання прийдуть з сервера та відобразяться на сторінці (Рис. 3.18.). Після вибору завдання з’являється форма «Відправка звіту» (Рис. 3.19.). Цю форму потрібно заповнити даними та відправити, сервер їх обробляє та зберігає, після відправки звіту, до нього можливо додатково прикріпити файли.

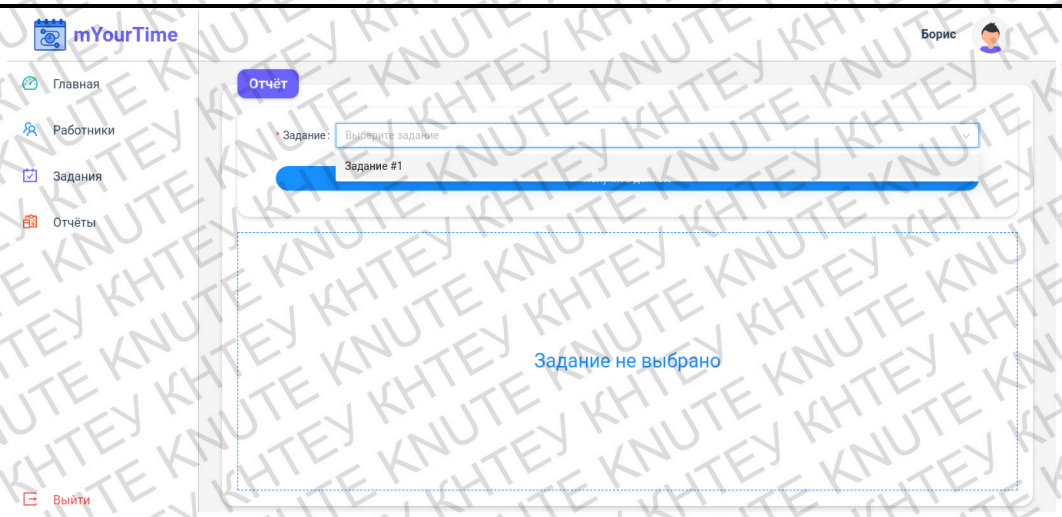

Рис. 3.17. Сторінка «Звіти» - завдання відсутнє

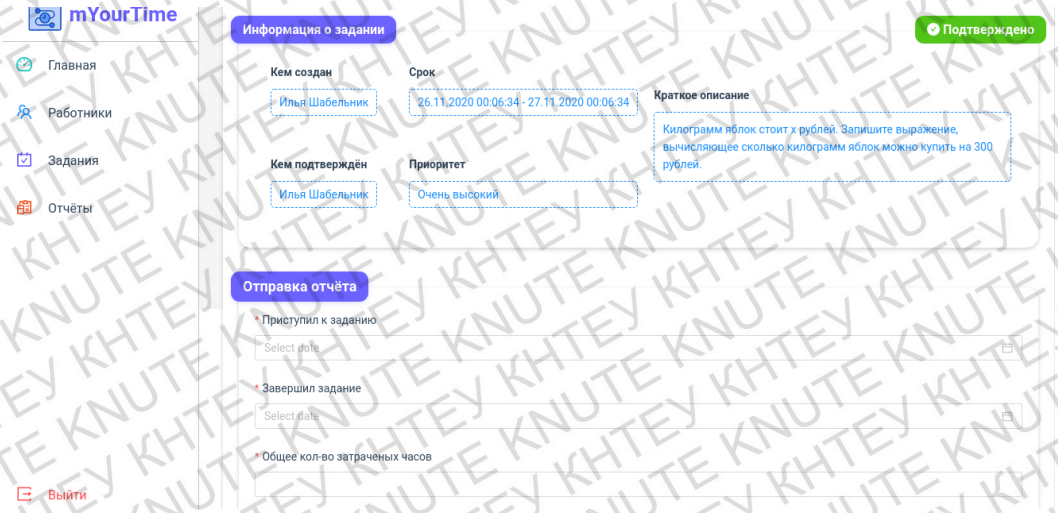

Рис. 3.18. Сторінка «Звіти» - завдання присутнє

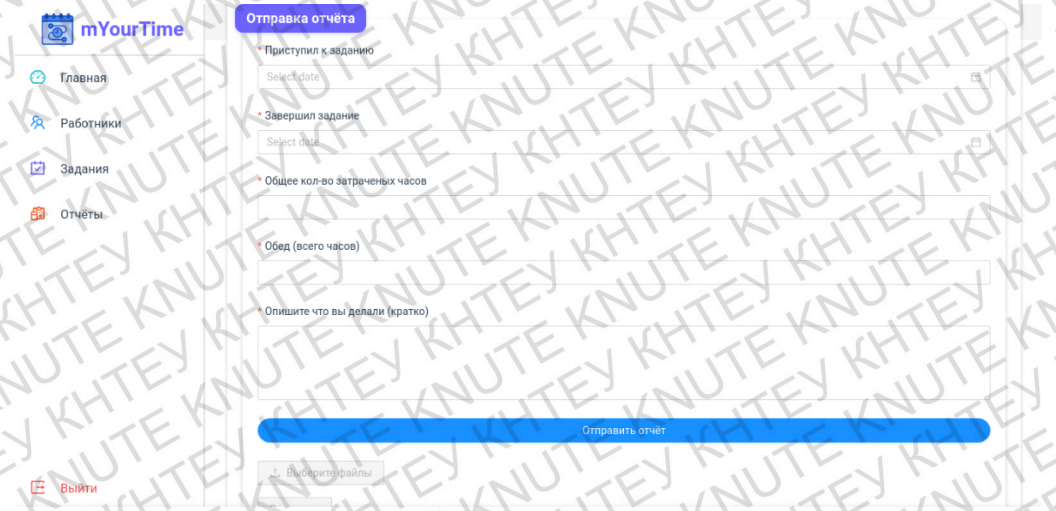

Рис. 3.19. Форма «Відправка звіту»

3.4. Висновок до розділу 3

В даному розділі було:

1. Описано структуру сервера, які були використані допоміжні бібліотеки в ході роботи над сервером.
2. Створено файл конфігурації.
3. Поетапно розказано та проілюстровано хід розробки серверної частини, а точніше розробки реєстрації.

Розроблено клієнтську частину проекту. Розказано про всі можливості, які присутні у проекті.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

З проведених досліджень можна зробити наступні висновки:

1. Був проведений аналіз методів управління, на що вони поділяються. Дана їх коротка характеристика зі вказанням переваг та недоліків цих методів. Було визначено на що ті чи інші методи спрямовані. Виходячи с теми проекту, було проаналізовано робочий час працівників. Що являється основою для планування робочого графіку, а також було визначено способи контролю робочого часу, приведено назви цих способів та їх коротке описання.

2. В якості БД для цього проекту було обрано MongoDB, відповідно до технічного завдання. Був проведений аналіз того, які колекції потрібно створити для зберігання усіх даних. Була спроектована модель цієї БД. Визначено які саме поля потрібні для запису даних. Також була спроектована архітектура додатку. Архітектура відображену у графічному вигляді та було описано поетпано, що відбувається на тій схемі, а також було висвітлено деякі нюанси роботи серверу.

3. Проведений аналіз технологій, що знаходяться на даний момент на ринку і було прийнято рішення використувати такі технології:

   - Клієнтська частина – React JS, Redux Js
   - Серверна частина – Node JS, Express JS, JWT
   - База даних – MongoDB
   - Редактор коду - WebStorm

4. В процесі виконання випускного кваліфікаційного проекту було розроблено веб-сайт, який дає можливість зареєструватися, авторизуватися, створити - працівників, окремі відділи, завдання, перевіряти завдання,

писати звіти по завданням та можливість редагувати, як дані працівників так і свої.

5. Розроблений веб-сайт повністю задовольняє поставлені функціональні вимоги згідно технічного завдання. В подальшому є можливість реалізувати додатковий функціонал, який не вказаний у ТЗ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація Node JS [Електронний ресурс]. - Режим доступу: https://nodejs.org/en/
2. Офіційна документація Express JS [Електронний ресурс]. - Режим доступу: https://expressjs.com/
3. Офіційна документація express-validator [Електронний ресурс]. - Режим доступу: https://express-validator.github.io/docs/
4. Офіційна документація JWT [Електронний ресурс]. - Режим доступу: https://jwt.io/
5. Офіційна документація React JS [Електронний ресурс]. - Режим доступу: https://reactjs.org/docs/getting-started.html
6. Офіційна документація Redux JS [Електронний ресурс]. - Режим доступу: https://redux.js.org/introduction/getting-started
7. Офіційна сторінка сайту StackOverflow [Електронний ресурс]. - Режим доступу: https://stackoverflow.com/

ТЕХНІЧНЕ ЗАВДАННЯ

Формулювання технічного завдання це найголовніша частина у розробці програмного забезпечення, тому що чим якісніше буде сфурмольоване ТЗ тим легше буде спроектувати весь проект. У ТЗ вказуються основні аспекти, які повині бути присутні в проекті.

1. Загальні відомості

Назва проекту: Work Time Manager

Планові строки розробки: липень 2020 – вересень 2020.

Потенційні користувачі: користувачі стаціонарних ПК, планшетів, мобільних телефонів з встановленим браузером: Chrome, Safari, Firefox, Brave.

Призначення програмного рішення: контроль часу працівників, контроль виконання завдань в організації.

Мета створення програмного рішення: застовуті набуті знання і навички у створені цього проекту, навчитися користуватися БД MongoDB.

2. Структура проекту

Оскільки цей сайт буде запускатися у браузері, то нам потрібно розробити основні частини, це:

1. Backend – ця частина буде приймати запити від користувачів, обробляти, фільтрувати їх. Також підключатися до БД і буде з неї брати дані. Сервер матиме можливість змінювати, видаляти дані з БД. Буде присутня фільтрація вхідних користувачів, для того щоб не зареєстровані користувачі не змогли отрмимати дані, які не призначені для них. Зареєстровані користувачі матимуть змогу

					КНТЕУ 121 06-23.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		28.02.20	Розробка інформаційної системи адміністрування робочого часу співробітників підприємства	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Рзаєва С.Л.		28.02.20		*ТЗ*	*53*	*52*
Гарант		Криворучко О.В.		28.02.20		Факультет інформаційних технологій 2м курс, 6 група		
Розробив		Шабельник І.Я.		28.02.20	*Технічне завдання*			

отримувати дані тільки ті, які відносяться до їх оргнанізації, а також відповідно до їх роли в цій організації (підприємства).

2. Front-end частина – це інтерфейс для зручного користування веб-сайтом. На ній будуть присутні різні форми та кнопочки, для здійснення таких операцій:
   - Реєстрація
   - Авторизація
   - Отримання нового листа для підтвердження email
   - Редагувати профіль
   - Підтвердити старий пароль та створити новий
   - Переходити між сторінками: «Головна», «Працівники», «Завдання», «Звіти», «На перевірку».
   - Можливість створювати нових працівників, призначати відділи (також якщо не має потрібного відділу в списку, то мати можливість створити цей відділ і вибрати його для нового працівника). Надавати їм різні ролі, але щоб, наприклад, «Тимчасовий керівник» не міг створити нового працівника з ролю «Керівник» або «Засновник» (тому що ці ролі знаходяться вище по рангу і їх не можуть призначати працівники нижче по рангу).
   - Можливість продивлюватися усіх працівників, які зареєстровані в організації (відображення в табличці).
   - Можливість вибрати працівника в табличці і редагувати його дані
   - Можливість звільнити або поновити працівника.
   - Створювати нові завдання для працівників з можливістю прикріплювати допоміжні файли до призначеного завдання, щоб працівнику було зрозуміло, що саме керівник хоче побачити при виконані даного завдання.

- Також при перевірці завдань надати можливість керівнику підкріпляти деякий коментарій с доп. файлами для завдань, які не пройшли перевірку. А для завдань, які пройшли перевірку успішно, просто «Підтвердити» виконання без написання коментарів.
- Надати користувачу з ролю «Працівник» - можливість виконувати завдання, продивлюватись дані працівників, які зареєстровані в організації (без можливості їх змінювання, видалення з системи).
- Також «Працівник» - повинен мати можливість відправляти виконані завдання до керуючого (вибрати зі списку, потрібного керючого). Продивлюватися коментарі, які присилаються від керуючого та скачувати файли, які підкріплені до нього
- Також «Працівник» - повинен мати можливість відправляти звіти з деякою інформацією. Принцип роботи зі звітами – користувач вибирає завдання зі списку, переходить до форми «Створення звіту», вписує дані (дата коли почав та закінчив виконувати завдання, скільки годин всього було витрачено на виконання, скільки год витрачено на відпочинок, і невеликий опис, що було зроблено по факту + підкріплення файлів до цього звіту).

3. Функціональні вимоги до додатку

Сайт повинен бути доступним тільки для зареєстрованих користувачам. Авторизацію необхідно реалізувати по логіну та паролю, взаємодіяти за допомогою JSON Web Token (JWT) технології або сесій браузера, та зберігати в куках, якщо це сесії, або ж в локальному сховищі — для JWT.

Розробити функціонал в серверній та клієнтській частині, за яким можливо завантажувати та зберігати файли, які в подальшому можливо

підкріплювати до звітів, коментарів або завдань. Файли повинні містити інформацію в БД .

Клієнтська частина повинна бути реалізована за допомогою фреймворків — Vue.js, React або Angular, з роутінгом і глобальним сховищем (використовувати Redux JS або Vuex). Для збору клієнтської частини, необхідно обрати одну з наступних технологій — Webpack.

Спроектувати базу даних MongoDB, яка буде вмістити в собі всю необхідну інформацію по користувачам, завданням, звітам.

На сайті повинен бути реалізована пагінація та інший додатковий функціонал, з яким зручно взаємодіяти та оброблювати інформацію, яка виводиться на сайті.

ТЕСТУВАННЯ ВЕБ-САЙТУ WORK TIME MANAGER

Головною метою такої ретельної перевірки сайту є грамотне налаштування всіх показників, однією або всіх сторінок сайту, оперативне виявлення і усунення всіх дефектних місць на сайті, а також його адаптація під різні пристрої. Що буде корисно і зручно як для самого власника сайту, так і для його відвідувача.

Для тествування веб-сайту буде застосовуватись метод ручного тествування, а також написання тест-кейсів. Тестування буде проводитись по деяким пукнтам:

- Реєстрація користувача
- Підтвердження email
- Авторизація

Таблиця Т.1

Тест-кейси

№	Назва	Кроки	Очікуваний результат	Результат
1.	Реєстрація (успішна)	1.Перехожу на сторінку «Реєстрації» 2.Запонюю форму даними 3.Відправляю дані	Реєстрація пройшла успішно і на електрону адресу відіслався лист для підтвердження Email	**Успішно**
2.	Підтвердження Email (24 години не прошло)	1.Перехожу на пошту 2.Відкриваю листа	Перенаправленя на сторінку підтвердження,	**Успішно**

					КНТЕУ 121 06-23.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		24.01.20	Розробка інформаційної системи адміністрування робочого часу співробітників підприємства	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Рзаєва С.Л.		24.01.20		*ПМТ*	*57*	*52*
Гарант		Криворучко О.В.		24.01.20		Факультет інформаційних технологій 2м курс, 6 група		
Розробив		Шабельник І.Я.		24.01.20	*Програма та методика тестування*			

Проводовження таблиці Т.1

№	Назва	Кроки	Очікуваний результат	Результат
		3.Натискаю на кнопку «Підтвердити»	редірект на сторінку «Авторизація», відображення повідомлення про те, що підтвердження пройшло успішно	
3	Підтвердження Email (прошло 24 години)	1.Перехожу на пошту 2.Відкриваю листа 3. Натискаю на кнопку «Підтвердити»	Перенаправленя на сторінку підтверждження, Редірект на сторінку «Авторизація». Повідомлення про те, що не вдалося підтвердити email, термін придатності посилання вичерпаний.	**Успішно**
4	Авторизація (Email не підтверджений)	1.Заповнюю форму авторизації 2.Натискаю на кнопку «Увійти»	Отримане повідомлення про те, що авторизація не можлива. Причина: Email не підтверджений	**Успішно**
5	Авторизація (Email підтверджений)	1.Заповнюю форму авторизації 2.Натискаю на кнопку «Увійти»	Авторизація успішна. Перенаправлення на «Головну» сторінку авторизованого користувача	**Успішно**

ДОДАТКИ

Додаток А

Код файлу ініціалізації сервера index.js

```
const express = require('express')
const bodyParser = require('body-parser')
const cookieParser = require('cookie-parser')
const cors = require('cors')
const path = require('path')
const debug = require('debug')
const databaseConnect = require('./utils/database-connect')
require('dotenv').config()

const log = debug('http')

// bring routes
const authRoutes = require('./routes/auth.route')
const confirmRoutes = require('./routes/confirm.route')
const userRoutes = require('./routes/user.route')
const employeeRoutes = require('./routes/employee.route')
const departmentRoutes = require('./routes/department.route')
const roleRoutes = require('./routes/role.route')
const taskRoutes = require('./routes/task.route')
const reviewRoutes = require('./routes/review.route')
const reportRoutes = require('./routes/report.route')

// app
const app = express()
debug.enable('*')

//generate static path for images
app.use('/images', express.static(path.join(__dirname, 'images')))
app.use('/files', express.static(path.join(__dirname, 'files')))

// middlewares
app.use(bodyParser.json())
app.use(cookieParser())

// cors
if (process.env.NODE_ENV === 'development') {
  app.use(cors({ origin: `${process.env.CLIENT_URL}` }))
}

app.use((req, res, next) => {
  log(req.method + ' ' + req.url)
  next() })
```

Продовження дод. А

```
app.use('/auth', authRoutes)
app.use('/confirm', confirmRoutes)
app.use('/user', userRoutes)
app.use('/employee', employeeRoutes)
app.use('/department', departmentRoutes)
app.use('/role', roleRoutes)
app.use('/task', taskRoutes)
app.use('/review', reviewRoutes)
app.use('/report', reportRoutes)

// port
const port = process.env.PORT || 5000

// Connect to db
databaseConnect()

app.listen(port, (err) => {
 if (err) throw err
})
```

Додаток Б

Код контролера auth.controller.js

```
const User = require('../models/user.model')
const { saveNewUserAndOrganization, getUserAndLogin, saveTokenAndSendNewToken } = require('../services/auth.service')
const jwt = require('jsonwebtoken')

const expiresTime = parseInt(process.env.EXPIRES_TIME)

exports.register = async (req, res) => {
 const user = await User.findOne({ email: req.body.email })

 if (user) {
  return res.status(400).json({
   error: `Данный email уже используется!`
  })
 }

 await saveNewUserAndOrganization(req, res)
}

exports.login = (req, res) => {
 const { email, password } = req.body

 getUserAndLogin(email, password, expiresTime, res)
}

exports.logout = (req, res) => {
 res.clearCookie('token')
 res.status(200).json({
  success: 'Вы вышли из учётной записи!'
 })
}

exports.refreshToken = (req, res) => {
 // Token must be in request HEADER
 const headerToken = req.headers.authorization.split(' ')[1]
 const refreshTime = parseInt(req.body.refresh)
 const nameOrg = req.body.nameOrg
 const name = req.body.name

 if (!headerToken) {
  return res.status(422).json({ message: 'Token not found' })
 }

 let decodedToken = jwt.verify(headerToken, process.env.ACCESS_TOKEN_SECRET, { ignoreExpiration: true })

 if (refreshTime <= Math.floor(Date.now() / 1000)) {
  return res.status(422).json({ error: 'Не удалось обновить токен, заново авторизируйтесь' })
 }

 saveTokenAndSendNewToken(res, { headerToken, decodedToken, nameOrg, name, expiresTime, refreshTime })}
```

Додаток В

Код сервісу auth.service.js

```
const emailTransporter = require('../emails/email-transporter')
const User = require('../models/user.model')
const Organization = require('../models/organization.model')
const Token = require('../models/token.model')
const Role = require('../models/role.model')
const BlockList = require('../models/blocklist.model')
const Department = require('../models/department.model')
const registrationMail = require('../emails/registration')
const { generateToken } = require('../utils/generate-token')

/**
 * saveNewUserAndOrganization. Create new user, organization and save them in DB.
 * @param {object} req
 * @param {object} res
 * @return {*}
 */
exports.saveNewUserAndOrganization = async (req, res) => {
  const { name, email, organization, password } = req.body

  try {
    const findRole = await Role.findOne({ 'role-code': 3 })
    const newUser = new User({ name, email, password, isOwner: true, role: { name: findRole.name, code: findRole['role-code'] } })
    const department = await Department.findOne({ name: 'администрация' })

    if (!department) {
      const newDepartment = new Department({ name: 'администрация' })
      newUser.department = { name: newDepartment['name'] }
      newDepartment.save()
    } else {
      newUser.department = { name: department['name'] }
    }

    const newOrganization = new Organization({ name: organization })
    newUser.organization = newOrganization._id

    const token = new Token({
      _userId: newUser._id,
      token: generateToken({ email }, { expiresIn: 86400 }),
      expireAfterSeconds: 86400
    })

    await newUser.validate()
    await token.validate()
    await newOrganization.validate()

    const link = `${process.env.CLIENT_URL}/confirm/${token.token}`

    await emailTransporter.sendMail(registrationMail(email, link, name))
```

```
    token.save()
    newOrganization.save()
    newUser.save()

    res.status(200).json({ success: 'Регистрация прошла успешно. На Ваш электронный адрес отправлено письмо для подтверждения аккаунта!' })
   } catch (err) {
    return res.status(500).json({ msg: err.message })
   }
  }

  /**
   * getUserAndLogin. Looking for user in DB and return data for authorized him.
   * @param {string} email
   * @param {string} password
   * @param {number} expiresTime
   * @param {object} res
   * @return {*}
   */
  exports.getUserAndLogin = (email, password, expiresTime, res) => {
   return User.findOne({ email }).populate('organization')
    .exec((err, user) => {
     if (err) {
      return res.status(500).json({ msg: err.message })
     } else if (!user) {
      return res.status(400).json({ error: 'Адрес электронной почты не связан ни с одной учетной записью. Пожалуйста, проверьте и попробуйте еще раз!' })
     } else if (!user.authenticate(password)) {
      return res.status(400).json({ error: 'Введён неправильный пароль!' })
     } else if (!user.isVerified) {
      return res.status(400).json({ error: 'Ваша почта не была подтверждена. Пожалуйста сделайте это!' })
     } else if (user.isSacked) {
      return res.status(400).json({ error: 'Вы не можете войти в систему. Причина: Вас уволили!' })
     }

     const { _id, isOwner, role, name } = user
     let convertRoles = []

     for (let i = 0; i < role.code + 1; i++) {
      convertRoles.push(i)
     }

     const token = generateToken({ _id, isOwner, orgId: user.organization._id, roles: convertRoles, role: role.code }, { expiresIn: expiresTime })
     res.cookie('token', token, { expiresIn: expiresTime })

     res.status(200).json({
      token,
      user: _id,
      name,
      nameOrg: user.organization.name,
```

```
      exp: Math.floor(Date.now() / 1000 + expiresTime)
    })
  })
}

/**
 * saveTokenAndSendNewToken. Save token that is came from params to block list, generate new token and send to user.
 * @param {Object} res
 * @param {Object} data
 * @return {*}
 */
exports.saveTokenAndSendNewToken = (res, data = {}) => {
  const { headerToken, decodedToken: { _id, isOwner, orgId, roles, role }, nameOrg, expiresTime, refreshTime, name } = data

  return BlockList.findOne({ token: headerToken }).exec((err, token) => {
    if (err) {
      return res.status(500).json({ msg: err.message })
    } else if (token) {
      return res.status(422).json({ error: 'Невозможно обновить токен.' })
    }

    const oldToken = new BlockList({ token: headerToken, expireAfterSeconds: 86400 })

    oldToken.save((err) => {
      if (err) {
        return res.status(500).json({ msg: 'Ошибка сохранения старого токена' })
      }

      res.status(200).json({
        token: generateToken({ _id, isOwner, orgId, roles, role }, { expiresIn: expiresTime }),
        refresh: refreshTime,
        nameOrg,
        user: _id,
        name,
        exp: Math.floor(Date.now() / 1000 + expiresTime),
        success: 'Токен успешно обновлён'
      })
    })
  }
  )
}
```