Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота бакалавра

«СИСТЕМА ЕЛЕКТРОННОЇ КОМЕРЦІЇ ВЕБ-САЙТУ ТЕХНІКИ З СИСТЕМОЮ УПРАВЛІННЯ ДЛЯ АДМІНІСТРАТОРІВ»

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проекту

№	П.І.Б. Група
1	Волков Юрій Ігорович КН-П-202
2	Чермалих Олексій Олександрович КН-П-202
3	Бондаренко Данило Олександрович КН-П-202

Автор звіту

_____ Бондаренко Данило Олександрович

Одеса – 2024

АНОТАЦІЯ

УДК 004.9
Б-81

Бондаренко Д. О. Платформа електронної комерції , серверна частина : Автореферат випускної кваліфікаційної роботи на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп’ютерні науки». - Одеса: ОТУШ, 2024 - 21 с.

Дипломний проект присвячений розробці та впровадженню платформи електронної комерції для продажу електронної техніки, а також системи керування магазином для власника бізнесу. У проекті реалізовано серверну частину, що забезпечує зв'язок між інтернет-магазином, адміністративним додатком та базою даних для передачі даних.

Серверна частина проекту, побудована на Node.js , сервер містить моделі бази даних для структурування даних та забезпечує маршрути для взаємодії з клієнтською частиною додатків , каталоги, такі як PhoneSpecs та ProductImages, зберігають інформацію про специфікації пристроїв та відповідні зображення. Компонент ServerData містить додаткові файли даних, необхідні для роботи сервера, а конфігураційні файли в каталозі Configs дозволяють гнучко налаштовувати сервер під різні потреби .

ЗМІСТ

ВСТУП

У світі інформаційних технологій електронна комерція стала невід'ємною частиною життя багатьох людей. Онлайн-торгівля проникає у різні сфери бізнесу, полегшуючи процес купівлі та продажу товарів та послуг. З розвитком інтернету та мобільних технологій стало можливим створення ефективних платформ для реалізації товарів та автоматизації процесів управління підприємством.

Метою даного проекту є створення комплексного рішення, яке спростить процеси управління рухом та обліку комерційних елементів (товарів, послуг тощо).

Наш проект ділиться на 3 підпроекти :

1. TechX Store - інформаційна система е-комерції, що надає інтуїтивно зрозумілий інтерфейс, для вибору, перегляду, пошуку товарів, а також замовлення їх з використанням кошика. має картки з представленими товарами в магазині, на які можна натиснути і перейти до секції фільтр товарів, у слайдері представлено актуальні товари в магазині.
2. TechX Compass - це кросплатформений Desktop додаток для адміністраторів, який дає змогу керувати продуктами на сайті. В додатку реалізовано меню, що надає можливості додавання товару, перегляду замовлень, відстеження активності та відгуків клієнтів про товари. Також доступні налаштування, де можна змінити мову, перемкнути тему та змінити пароль.
3. TechX Server - це сервер, який виступає сполучною ланкою між клієнтом та базою даних, обробляє запити клієнтської сторони, а також надає API для взаємодії з фронтендом. Сервер забезпечує надійну роботу програм, керуючи всіма необхідними операціями та забезпечуючи безпеку даних. Крім того, він відповідає за автентифікацію та авторизацію користувачів, обробку сесії та зберігання зображень продуктів тощо.

TECHX SERVER

TechX Server - це сервер, який виступає сполучною ланкою між клієнтом та базою даних, обробляє запити клієнтської сторони, а також надає API для взаємодії з фронтендом. Сервер забезпечує надійну роботу програм, керуючи всіма необхідними операціями та забезпечуючи безпеку даних. Крім того, він відповідає за автентифікацію та авторизацію користувачів, обробку сесії та зберігання зображень продуктів. Архітектура сервера побудована за принципом багатошарової моделі, що включає рівні даних, бізнес-логіки та представлення.

Node.js - це серверна платформа, заснована на JavaScript-рушії V8 від Google. Вона дозволяє створювати високопродуктивні та масштабовані серверні додатки завдяки неблокуючій моделі введення/виведення та асинхронній обробці запитів. Використання Node.js забезпечує високу продуктивність і дозволяє розробникам використовувати одну мову програмування, JavaScript, як на серверній, так і на клієнтській стороні.

Основні компоненти та модулі Node.js в проекті Techx-server:

1. Express.js: Легкий і гнучкий фреймворк для створення веб-додатків і RESTful API. Використовується для обробки HTTP-запитів та маршрутизації.
2. Mongoose: Бібліотека для роботи з MongoDB. Забезпечує зручний інтерфейс для моделювання та управління даними.

Middlewares

Middleware у Node.js - це функції, які виконуються під час обробки HTTP-запитів в Express.js додатках, в нас використовується кілька важливих middleware, які допомагають обробляти запити, забезпечувати безпеку та зручність для користувачів.

express.json() - Цей middleware парсить вхідні запити з JSON-тілами.

CORS-запити що дозволяє контролювати доступ до ресурсів сервера з інших доменів.

```
app.use((req, res, next) =>
{
  const allowedOrigins = ["http://localhost:3000", "https://next-techx.vercel.app"];
  const origin = req.headers.origin;

  if (allowedOrigins.includes(origin))
    res.setHeader("Access-Control-Allow-Origin", origin);

      res.setHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS, PUT, DELETE");
  res.setHeader("Access-Control-Allow-Headers", "Content-Type, Authorization");
  res.setHeader("Access-Control-Allow-Credentials", true);

  next();
});
```

Middleware для вилучення токенів з заголовків авторизації. Токен витягується з заголовка Authorization і додається до об'єкту запиту для подальшої обробки

```
app.use((req, res, next) =>
{
   const auth_header = req.headers["authorization"];
   const utoken = auth_header && auth_header.split(" ")[1];

   req.token = utoken;

   next();
 });
```

Надсилання коду на пошту

Для відправки коду підтвердження на електронну пошту використовується бібліотека Nodemailer. Це дозволяє легко налаштувати та відправляти електронні листи через різні поштові сервіси.

ReadHTMLFile функція зчитує HTML-шаблон листа з файлу.

```
async function ReadHTMLFile()
{
 try
 {
  const html = await fs.readFile(_postcard_path, "utf-8");

  return html;
 }
 catch (error)
 {
  console.error("Error reading HTML file:", error);
  throw error;
 }
}
```

Потім асинхронна функція SendMail налаштовує транспортер Nodemailer для відправки електронної пошти через Gmail і відправляє лист з HTML-шаблоном параметр postcard_content містить HTML-контент повідомлення.

```
async function SendMail(postcard_content)
{
 try
 {
  const transporter = nodemailer.createTransport(
  {
   service: "gmail",
   auth: { user: process.env.GMAIL, pass: process.env.PASSWORD_GMAIL },
  });

  const mail_options =
  {
   from: process.env.GMAIL,
   to: _user_gmail,
   subject: "Confirm your email on TechX",
   html: postcard_content,
  };

  await transporter.sendMail(mail_options);
 }
 catch (error) { console.error("Error sending email:", error); }
}
```

Функція SEND_CODE_VERIFICATION приймає електронну адресу користувача та код підтвердження. Вона зчитує HTML-шаблон, замінює плейсхолдер XXXX на фактичний код і викликає SendMail для відправки листа.

```
function SEND_CODE_VERIFICATION(user_gmail, code)
{
 _user_gmail = user_gmail;
 _code = code;

 (async () =>
 {
  try
  {
   let postcard = await ReadHTMLFile();

   postcard = postcard.replace("<p>XXXX</p>", `<p>${_code}</p>`);

   await SendMail(postcard);
  }
  catch (error) { console.error("Error:", error); }
 })();
}
```

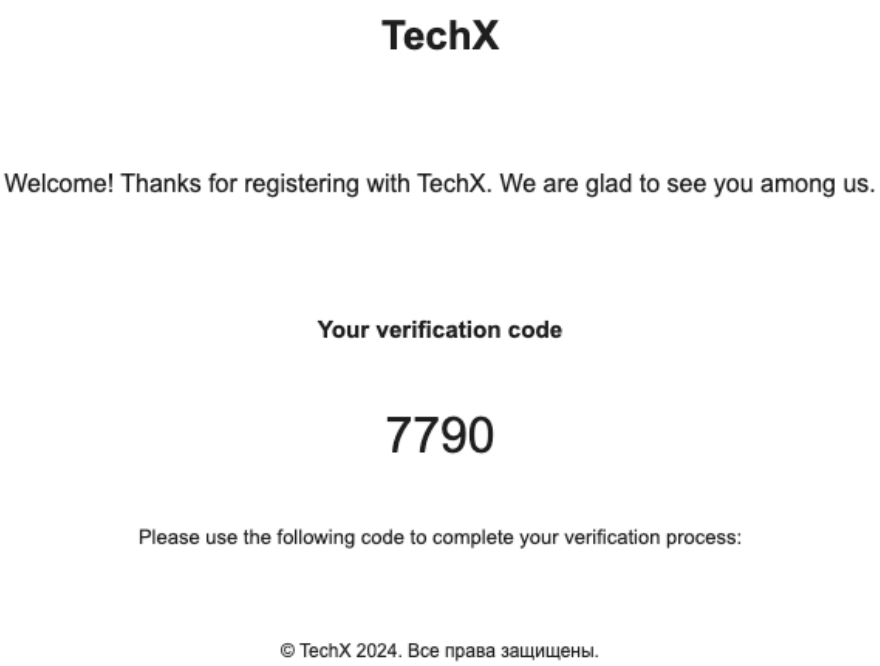
Маршрут обробляє запити на відправку коду підтвердження, генерує випадковий код і викликає SEND_CODE_VERIFICATION

```
app.post("/SendConfirmationCodeEmail", async (req, res) =>
{
 try
 {
  const { email } = req.body;
  const confirmation_code = Math.floor(1000 + Math.random() * 9000).toString();

  SEND_CODE_VERIFICATION(email, confirmation_code);

  res.status(200).json({ conf: confirmation_code });
 }
 catch (error)
 {
  console.error(error);
  res.status(500).json({ message: "Internal Server Error" });
 }
});
```

Для роботи з файловою системою та змінними оточення використовуються модулі fs/promises та dotenv відповідно. Змінні оточення, такі як адреса електронної пошти та пароль від пошти Gmail, зберігаються у файлі .env.



Малюнок 1 - Код підтвердження

Реєстрація користувачів

Процес реєстрації користувачів у проекті Techx-server включає кілька важливих кроків: перевірка існування користувача, відправка коду підтвердження на електронну пошту та додавання нового користувача до бази даних.

1. Перевірка існування користувача

   Клієнт відправляє POST-запит на маршрут /CheckUserExists з електронною адресою користувача у тілі запит потім сервер отримує запит і використовує модель UserModel, щоб знайти користувача з вказаною електронною адресою у базі даних, перевіряє чи користувач існує, сервер відповідає { existing_user: true } , якщо ні, то { existing_user: false }. У разі помилки сервер відповідає з кодом 500 і повідомленням про внутрішню помилку сервера

```
app.post("/CheckUserExists", async (req, res) =>
{
 try
 {
  const { email } = req.body;
  const existing_user = await UserModel.findOne({ email });
```

```
    if (existing_user)
      res.status(200).json({ existing_user: true });
    else
      res.status(200).json({ existing_user: false });
  }
  catch (error)
  {
    console.error(error);
    res.status(500).json({ message: "Internal Server Error" });
  }
});
```

2. Відправка коду підтвердження на електронну пошту
3. Додавання нового користувача

Клієнт відправляє POST-запит на маршрут /NewUser з ім'ям, електронною адресою та паролем користувача у тілі запиту , сервер створює новий об'єкт користувача, використовуючи модель UserModel, з отриманими даними. Новий об'єкт користувача зберігається у базі даних , сервер відповідає клієнту повідомленням про успішне додавання користувача: { message: "User added successfully" }.

```
app.post("/NewUser", async (req, res) => {
        try {
                const { name, email, password } = req.body;
                const new_user = new UserModel({
                        name, email, password,
                        phone_number: "null",
                        delivery_address: "null",
                        favourites: []
                });
                await new_user.save();
                res.status(200).json({ message: "User added successfully" });
        } catch (error) {
                console.error(error);
                res.status(500).json({ message: "Internal Server Error" });
         }
});
```

Авторизація користувачів

Авторизація користувачів складається з перевірки пароля користувача, генерації токена доступу та перевірки дійсності токена.

1. ProofPass перевіряє, чи правильно вказаний пароль для користувача з такою електронною адресою . Отримує запит і використовує модель UserModel, щоб знайти користувача з вказаною електронною адресою у базі даних , якщо користувач існує, сервер використовує bcrypt, щоб порівняти вказаний пароль з хешованим паролем у базі дани , у разі паролі співпадають, сервер відповідає { success: true }, інакше { success: false }

```
app.post("/ProofPass", async (req, res) =>
{
 try
 {
  const { email, password } = req.body;
  const existing_user = await UserModel.findOne({ email });

  if (existing_user)
  {
            const is_password_correct = await bcrypt.compare(password,
existing_user.password);

    if (is_password_correct)
     res.status(200).json({ success: true });
    else
     res.status(200).json({ success: false });
  }
  else
   res.status(200).json({ success: false });
 }
 catch (error)
 {
  console.error(error);
  res.status(500).json({ message: "Internal Server Error" });
 }
});
```

2. GenerateToken генерує JWT токен для користувача, що пройшов аутентифікацію. Клієнт відправляє POST-запит на маршрут

/GenerateToken з електронною адресою у тілі запиту , Сервер генерує JWT токен, використовуючи секретний ключ _token_secret_key, і встановлює термін дії токена на 1 годину , відповідає клієнту згенерованим токеном у форматі { token: "JWT токен" }.

```
app.post("/GenerateToken", async (req, res) =>
{
 try
 {
  const { email } = req.body;
  const token = jwt.sign({ user: email }, _token_secret_key, {expiresIn: "1h" });

  res.status(200).json({ token });
 }
 catch (error)
 {
  console.error(error);
  res.status(500).json({ message: "Internal Server Error" });
 }
});
```

3. CheckToken перевіряє дійсність JWT токена, переданого в заголовку авторизації. Клієнт відправляє POST-запит на маршрут /CheckToken з JWT токеном у заголовку авторизації . Сервер витягує токен з заголовку запиту , функція jwt.verify використовується для перевірки дійсності токена за допомогою секретного ключа _token_secret_key . Якщо токен дійсний, сервер відповідає { success: true }, інакше { success: false }.

```
app.post("/CheckToken", async (req, res) =>
{
 try
 {
  const token = req.token;

  jwt.verify(token, _token_secret_key, (err, decoded) =>
  {
   if (err)
    res.status(200).json({ success: false });
   else
    res.status(200).json({ success: true });
  });
 }
 catch (error)
 {
  console.error(error);
  res.status(500).json({ message: "Internal Server Error" });
 }
});
```

Запити сесії

Запити сесії складаються з створення сесії, отримання даних сесії, перевірка дійсності токена та завершення сесії

1. Створення сесії створює нову сесію для користувача після успішної аутентифікації . Сервер знаходить користувача у базі даних за електронною адресою , створюється новий об'єкт сесії з ідентифікатором користувача та токеном, витягнутим із заголовка запиту , об'єкт сесії зберігається у базі даних .

```
app.post("/CreateSession", async (req, res) =>
{
 try
 {
  const { email } = req.body;
  const id = await UserModel.findOne({ email });
  const new_session = new SessionModel({ user_id: id.id, token: req.token })

  await new_session.save();

  res.status(200).json({ create_session_data: true });
 }
 catch (error)
 {
  console.error(error);
  res.status(500).json({ create_session_data: false });
 }
});
```

2. Отримання даних сесії отримує дані сесії для користувача за вказаним токеном , знаходить сесію у базі даних за токеном , якщо сесія знайдена, сервер знаходить відповідного користувача за ідентифікатором користувача з сесії , відповідає клієнту даними користувача у форматі JSON , якщо користувач або сесія не знайдені, сервер відповідає з кодом 404.

```
app.post("/PullOutOfSession", async (req, res) =>
{
 try
 {
  const token = req.token;
  const session = await SessionModel.findOne({ token });

  if (session)
```

```
    {
      const user = await UserModel.findOne({ _id: session.user_id });

      if (user)
      {
         const user_data = { name: user.name, email: user.email, phone_number: user.phone_number, delivery_address: user.delivery_address, favourites: user.favourites };

        res.status(200).json({ user_data });
      }
      else
        res.status(404).json({ message: "User not found" });
    }
    else
      res.status(404).json({ message: "Session not found" });
  }
  catch (error)
  {
    console.error(error);
    res.status(500).json({});
  }
});
```

3. Перевірка дійсності токена перевіряє дійсність JWT токена, переданого в заголовку авторизації . Клієнт відправляє POST-запит на маршрут /CheckToken з JWT токеном у заголовку авторизації , потім витягує токен з заголовка запиту , функція jwt.verify використовується для перевірки дійсності токена за допомогою секретного ключа _token_secret_key , у разі якщо токен дійсний, сервер відповідає { success: true }, інакше { success: false }.

```
app.post("/CheckToken", async (req, res) =>
{
  try
  {
    const token = req.token;

    jwt.verify(token, _token_secret_key, (err, decoded) =>
    {
      if (err)
        res.status(200).json({ success: false });
      else
        res.status(200).json({ success: true });
    });
  }
  catch (error)
  {
    console.error(error);
    res.status(500).json({ message: "Internal Server Error" });
  }
});
```

Запити для клієнта

Запити для клієнта включає такі основні операції: пошук продуктів, отримання зображення продукту та отримання даних для каруселі на головній сторінці.

1. Пошук продуктів здійснює пошук продуктів у базі даних . Клієнт відправляє POST-запит на маршрут /SearchForProducts з текстом пошукового запиту у тілі запиту. Сервер шукає продукти у різних колекціях бази даних, таких як IPhoneModel, AirPodsModel, AppleWatchModel, MacbookModel, IpadModel, ConsoleModel, де атрибути продуктів відповідають пошуковому запиту , сервер форматує знайдені дані і відповідає клієнту масивом об'єктів продуктів .

```
app.post("/SearchForProducts", async (req, res) =>
{
 try бы
 {
  const { query } = req.body;
  const iphones = await IPhoneModel.find(
  {
   $or:
   [
    { brand: { $regex: query, $options: "i" } },
    { category: { $regex: query, $options: "i" } },
    { model: { $regex: query, $options: "i" } }
   ]
  }).select("images model price");

  const formatted_data_iphones = iphones.map(i =>
  {
   return { _id: i.id, images: i.images[0], model: i.model, price: i.price };
  });
```

2. Отримання зображення продукту отримує зображення продукту за його назвою . Сервер декодує назву зображення і шукає його у директорії ProductImages , сервер створює потік для читання файлу і відправляє його клієнту , якщо зображення не знайдено, сервер відповідає з кодом 404.

```
app.get('/GetImage/:ImageName', (req, res) =>
{
 const image_name = decodeURIComponent(req.params.ImageName);
 const image_path = join(__dirname, 'ProductImages', image_name);

 if (fs.existsSync(image_path))
 {
    const image_stream = fs.createReadStream(image_path);
```

```
    image_stream.pipe(res);
  }
  else
    res.status(404).send('Image not found');
});
```

3. Отримання даних для каруселі отримує дані продуктів, які повинні відображатися у каруселі на головній сторінці. Сервер шукає всі продукти у різних колекціях бази даних , потім фільтрує продукти, які повинні відображатися у каруселі , форматовані дані продуктів відповідають клієнту у форматі JSON.

```
app.post("/GettingDataForCarusel", async (req, res) =>
{
  try
  {
    const iphones = await IPhoneModel.find();

    const formatted_data_iphones = iphones.filter(i => i.incarousel ===
true).map(i =>
    {
       return { id: i.id, images: i.images[0], model: i.model, price: i.price,
descont_price: i.descont_price};
    });
 const formatted_data = [...formatted_data_iphones , ...formatted_data_airpods,
...formatted_data_applewatchs, ...formatted_data_macbooks,
...formatted_data_ipads, ...formatted_data_consoles];

    res.status(200).json(formatted_data);
  }
  catch (error)
  {
    console.error(error);
    res.status(500).json({});
  }
});
```

MONGODB

MongoDB – це NoSQL база даних, орієнтована на зберігання та обробку документів, представлених у форматі BSON, бінарного аналога JSON. Документи MongoDB об'єднуються в колекції, а колекції в свою чергу групуються в бази даних. Гнучка схема MongoDB дозволяє документам в одній колекції мати різну структуру, що робить її зручною для розробки та зміни додатків. Завдяки можливості горизонтального масштабування MongoDB дозволяє легко додавати нові вузли для обробки великих обсягів даних, забезпечуючи високу продуктивність та стійкість до відмов. MongoDB пропонує потужну мову запитів для пошуку та агрегації даних, що робить її корисною для різних типів програм, включаючи веб-розробку, аналіз даних, керування вмістом та інші. Екосистема MongoDB включає різні інструменти та драйвери для розробки та адміністрування, забезпечуючи зручність використання і масштабованість додатків.

У нашому проекті MongoDB використовується як основна база даних для зберігання інформації про продукти, користувачів та сесії. Взаємодія з MongoDB здійснюється за допомогою бібліотеки Mongoose, яка забезпечує зручний API для роботи з документами MongoDB у Node.js .Підключення до MongoDB складається з команди mongoose.connect(DB_URL) де DB_URL рядок підключення до бази даних яка зберігається у файлі connect_db.js . Також у нас також у нас є моделі які використовується для взаємодії з колекціями.

```
const AirPodsSchema = new mongoose.Schema(
  {
    _id: mongoose.Schema.Types.ObjectId,
    category: String,
    brand: String,
    model: String,
    price: Number,
    descont_price: Number,
    color: String,
    description: String,
    processor: String,
    battery: String,
    images: [String],
    incarousel: Boolean,
  },
  { collection: "AirPods" }
);
```

ВИСНОВКИ

Отже TechX-Store реалізований на платформі Next.js, забезпечуючи високопродуктивний та SEO-оптимізований веб-додаток для користувачів. Цей додаток дозволяє користувачам переглядати каталог товарів, здійснювати пошук, додавати товари в улюблені та оформлювати замовлення. Next.js забезпечує швидкий рендеринг сторінок та зручну інтеграцію з іншими сервісами.

Адмін-панель TechX-Compass розроблена на основі Electron, що дозволяє створювати кросплатформені настільні додатки з використанням веб-технологій. Ця панель надає можливості для адміністраторів та менеджерів управляти каталогом товарів, обробляти замовлення та аналізувати дані про продажі. Використання Electron спрощує розробку та розгортання адмін-панелі на різних операційних системах.

Серверна частина TechX-Server написана на Node.js і виступає основним інструментом для обробки запитів, управління базою даних та реалізації бізнес-логіки. Сервер використовує MongoDB для зберігання даних про користувачів, товари та замовлення, забезпечуючи високу швидкість доступу та надійність зберігання. Node.js дозволяє ефективно обробляти велику кількість одночасних запитів, що важливо для масштабованості системи.

У системі впроваджено засоби покращення інформаційної безпеки, зокрема, використання токенів для управління сесіями та відправка коду підтвердження на електронну пошту користувачів. Це забезпечує високий рівень безпеки та зручності для користувачів, зменшуючи ризик несанкціонованого доступу до облікових записів.

Інтеграція з бібліотекою Mongoose спрощує роботу з MongoDB, забезпечуючи зручний API для взаємодії з базою даних. Це дозволяє легко реалізувати CRUD-операції та забезпечити високий рівень організації даних.

Система також включає функціонал для пошуку продуктів, отримання даних

для каруселі на головній сторінці та управління зображеннями продуктів. Це дозволяє створювати динамічний та привабливий інтерфейс для користувачів, підвищуючи їх задоволення від використання сервісу.

Проект TechX-Store забезпечує зручний інтерфейс для кінцевих користувачів, TechX-Compass дозволяє ефективно управляти магазином, а TechX-Server забезпечує надійну серверну інфраструктуру. Разом ці компоненти утворюють потужну та гнучку систему для управління інтернет-магазином, здатну задовольнити потреби як користувачів, так і адміністраторів.

ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ

№ п.п.	Ресурс	Ступінь запозичення
1.	Next.js Розробник: Vercel; Документація: https://nextjs.org/	Використано у якості фреймворку React для платформи для розробки Web-додатку.
2.	React.js Розробник: Facebook; Документація: https://react.dev/	Використано у якості платформи для розробки Web-додатку.
3.	Electron Розробник: GitHub; Документація: https://www.electronjs.org/	Використана як створення системи для адміністраторів.
5.	Node.js Розробник: Ryan Dahl; Документація: https://nodejs.org/en	Використовується як розробка WEB-додатку, системи управління для адміністраторів та сервера
6.	JavaScript Розробник: Brendan Eich; Документація: https://developer.mozilla.org/en-US/docs/Web/JavaScript	Використовувався для написання системи для керування адміністраторів та сервера

7.	JSX Розробник: Jordan Walke; Документація: https://ru.legacy.reactjs.org/docs/introducing-jsx.html	Використано у якості язика програмування для розробки Web-додатку.
8.	MongoDB Розробник: 10gen; Документація: https://www.mongodb.com/	Використовувалося як глобальне сховище, для зберігання даних
9.	Visual Studio Code Розробник: Microsoft; Документація: https://code.visualstudio.com/	Використовувалося як редактор коду