МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

по дисциплине «Введение в нереляционные базы данных»

Тема: Умный склад на стройплощадке

Студент гр. 0303 _____ Амежее Ш.К.

Студент гр. 0304 _____ Жиглов Д.С.

Студент гр. 0304 _____ Карабанов Р.Е.

Преподаватель _____ Заславский М.М.

Санкт-Петербург

2023

ЗАДАНИЕ

Студенты

Амежее Ш.К.

Жиглов Д.С.

Карабанов Р.Е.

Группа 0303-0304

Тема задания: Умный склад на стройплощадке

Исходные данные:

Задача - сделать сервис для учета, отпуска и поступления материалов, нужных для строительства дома. Пользователи - рабочие, прорабы, кладовщики.

Необходимые (но не достаточные фичи) - аккаунты пользователей, страницы позиций, страница “Склад”, статистика , страница для формрования накладных, страница для анализа потребления и прогноза позиций.

Содержание пояснительной записки:

“Содержание”

“Введение”

“Сценарий использования”

“Модель данных”

“Разработанное приложения”

“Заключение”

“Приложение А. Документация по сборке и развертыванию приложения”

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 21.12.2023

Дата сдачи реферата: 29.03.2024

Дата защиты реферата: 29.03.2024

Студент 0303	_____	Амежее Ш.К.
Студент 0303	_____	Жиглов Д.С.
Студент 0303	_____	Карабанов Р.Е.
Преподаватель	_____	Заславский М.М.

АННОТАЦИЯ

В рамках данного курса предполагалось разработать в команде приложение на одну из предложенных тем. Была выбрана тема: Умный склад на стройплощадке. Для выполнения задания предлагается использовать СУБД MongoDB.

Найти исходный код и всю дополнительную информацию можно по ссылке: https://github.com/moevm/nosql2h23-construction

SUMMARY

Within the framework of this course it was supposed to develop in a team an application on one of the proposed topics. The topic was chosen: Smart Warehouse on a construction site. To perform the task it is suggested to use MongoDB DBMS.

You can find the source code and all additional information at the link: https://github.com/moevm/nosql2h23-construction.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – создать высокопроизводительное и удобное решение для для учёта материалов в стройплощадке.

Было решено разработать веб-приложение, которое позволит хранить информацию о материалах, информацию о накладных и о пользователей, создать статистику и т.п.

1. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

1.1. Макет UI

Разработанный макет приложения доступен по ссылке: https://www.figma.com/file/sSeZrKHpKsqohmrDdgi01z/Figma-Basics?type=design&node-id=330-2&mode=design&t=CfJIEW9FBMOXLjRC-0

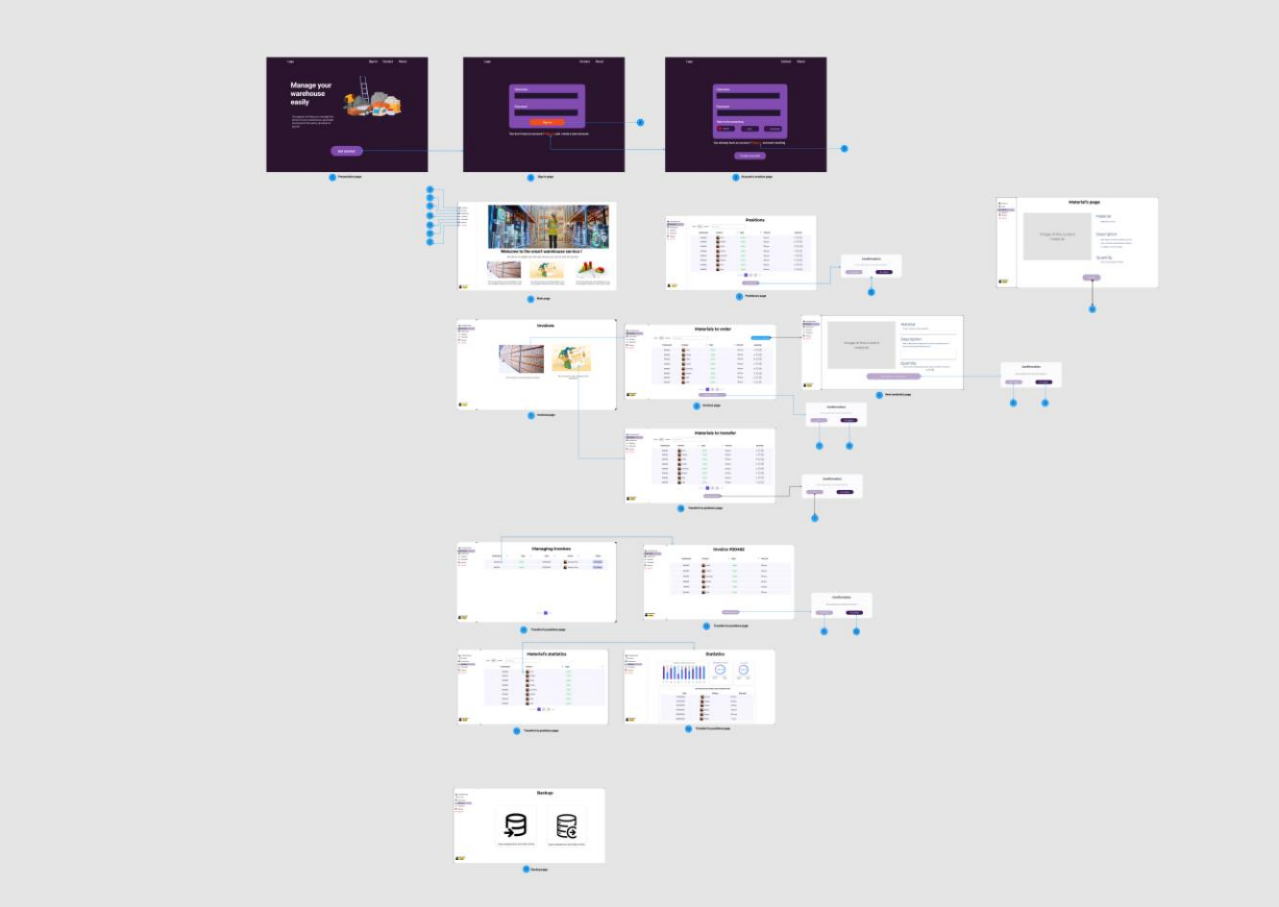


Рисунок 1 – Макет UI

1.2. Сценарии использования

Сценарий использования - « Использование материалов »

Действующее лицо: Рабочий

Основной сценарий:

1. Пользователь заходит в приложении.
2. Пользователь вводит свой логин и пароль и нажимает на кнопку «Sign In».
3. Пользователь нажимает на кнопку «Positions» на левом панеле.
4. Пользователь выбирает в списке нужные материалы с необходимым количеством.
5. После выбора, пользователь нажимает на кнопку «Use materials» и потверждает использование материалов.

Сценарий использования - « Генерация накладных »

Действующее лицо: Прораб

Основной сценарий:

1. Пользователь заходит в приложении.
2. Пользователь вводит свой логин и пароль и нажимает на кнопку «Sign In».
3. Пользователь нажимает на кнопку «Invoices» на левом панеле.
4. Пользователь выбирает какой тип накладных генерировать: для пополнения запасов или перемещения материалов со склада на позиции
5. Пользователь выбирает в списке нужные материалы с необходимым количеством.
6. В случае заказа материалов, пользователь может нажимать на кнопку «Add new material» для создания нового материала чтобы его заказать.

7. После выбора, пользователь нажимает на кнопку «Validate invoice» и потверждает создание накладных.

Сценарий использования - « Посмотр статистики »

Действующее лицо: Прораб

Основной сценарий:

1. Пользователь заходит в приложении.
2. Пользователь вводит свой логин и пароль и нажимает на кнопку «Sign In».
3. Пользователь нажимает на кнопку «Statistics» на левом панеле.
4. Пользователь выбирает из списка материалов тот, который будет объектом статистики.

Сценарий использования - « Управление накладными »

Действующее лицо: Кладовщик

Основной сценарий:

1. Пользователь заходит в приложении.
2. Пользователь вводит свой логин и пароль и нажимает на кнопку «Sign In».
3. Пользователь нажимает на кнопку «Warehouse» на левом панеле.
4. Пользователь выбирает из списка нужный ему накладный.

5. Пользователь для подверждения обработки накладного нажимает на кнопку «Validate invoice»

Сценарий использования - « Импорт - Экспорт данных »

Действующее лицо: Прораб

Основной сценарий:

1. Пользователь заходит в приложении.
2. Пользователь вводит свой логин и пароль и нажимает на кнопку «Sign In».
3. Пользователь нажимает на кнопку «Import/Export» на левом панеле.
4. Пользователь выбирает вид операции бэкупа.
5. База данных в зависимости от выбранной операции либо экспортируется, либо импортируется указав путь к файлу.

2. МОДЕЛЬ ДАННЫХ

2.1. Нереляционная модель данных

Нереляционная модель

Графическое представление

Коллекции

users

```
    username: String,
    password: String,
    role: String,
    avatar: {
      _id: false,
      name: String,
      content: Buffer,
      type: String
```

materials

```
    name: String,
    type: String,
    quantity: [Number],
    unity: String,
    creation: String,
    description: String
```

invoices

```
  type: String,
```

```
    creation: String,
    author: String,
    contain: [
        {
            _id: false,
            material_id: {
                type: mongoose.Schema.Types.ObjectId,
                ref: "Materials",
                require: true
            },
            quantity:{
                type: Number,
                require: true
            },
        }
    ]
statistics
    date: Date,
    person: {
        type: mongoose.Schema.Types.ObjectId,
        ref: "Users",
        require: true
    },

    contain: [
    {
        _id: false,
        material_id: {
```

```
            type: mongoose.Schema.Types.ObjectId,
            ref: "Materials",
            require: true
        },
        quantity:{
            type: Number,
            require: true
        },
    }
]
```

Описание назначений коллекций, типов данных и сущностей

users – коллекция, для хранения данных пользователей.

_id – уникальный идентификатор пользователя

username – имя пользователя

password – пароль

role – роль в компании (рабочий, прораб или кладовщик)

avatar – изображение пользователя

name – название изображения

content – бинарный массив изображения

type – формат изображения

materials – коллекция для хранения данных о материалов.

_id – уникальный идентификатор материала

name – название материала

type – тип материала

quantity – массив для хранения количесвта материала в позициях и в складе

creation – дата создания материала

unit – единица измерения материала

description – описание материала

invoices – коллекция для хранения данных о накладных

type – тип накладных (отпуск или послупление материалов)

creation – дата формирования накладных

author – автор накладных

content – содержание накладных в виде массива:

material_id – идентификатор материала

quantity – количество необходимых материалов

statistics – коллекция для хранения данных о потребления материалов

_id – уникальный идентификатор потребления

date – дата потребления материала

person – идентификатор автора потребления

contain – содержание списка ппотребления:

material_id – идентификатор материала

quantity – количество необходимых материалов

Оценка объема информаций, хранимой в модели

 users

Допустим что имеется x пользователей.

_id – тип ObjectID V = 12 байт

username – String V = 30 байт

password – String V = 15 байт

role – String V = 10 байт

avatar – Array V_i = 20 + 8 + 4000= 4028 байт

name - String V = 20 байт

date - Date V = 8 байт

img - Binary Data (допустим, что средний размер снимка 4 кб) V = 4000 байт

 materials.

Допустим что имеется у материалы

_id – тип ObjectID V = 12 байт

name – String V = 30 байт

type – String V = 15 байт

stock – тип Object V = 8 + 8 = 16 байт

positions – Double V = 8 байт

stock – Double V = 8 байт

unit – String V = 6 байта

creation – Date V = 8 байт

description – String V = 300 байт

 invoices.

Допустим что для каждого накладного в среднем mat = 10 различные материалы. Тогда для хранения х накладных прнадобится:

_id – тип ObjectID V = 12 байт

type – String V = 10 байт

creation – Date V = 8 байт

author – String V = 30 байт

contain – тип Object V = 12 + 8= 20 байт

material_id – ObjectID V = 12 байт

quantity – Double V = 8 байт

 statistics.

Допустим что для каждого потребления в среднем используются mat = 10 различные материалы. Тогда для хранения x потреблений прнадобится:

_id – тип ObjectID V = 12 байт

date – Date V = 8 байт

person – ObjectID V = 12 байт

contain – тип Object V = 12 + 8 = 20 байт

material_id – ObjectID V = 12 байт

quantity – Double V = 8 байт

Тогда получаем следующий объём данных для хранения x пользователей и y материалов:

Избыточность модели (отношение между фактическим объемом модели и “чистым” объемом данных).

В БД дублируются данные о материалов и о накладных, у пользователей может отсутствовать аватарка. Тогда получаем:

Отбросим свободный член, чтобы сравнить с фактическим объемом:

Отношение между фактическим и "чистым" объемом данных:

Направление роста модели при увеличении количества объектов каждой сущности

Как и раньше, выразим через количество пациентов и получим:

Отсюда видно, что рост модели при увеличении количества объектов каждой сущности линеен и зависит от количества материалов и пользователей.

Запросы к модели, с помощью которых реализуются сценарии использования

Добавление нового пользователя

```
await UserModel.insertMany(
    [
        {
            username: username,
            password: password,
            role: role,
            avatar:   avatar
        }]).then(console.log("User added successfully"));
```

Добавление материала в позиции

```
MaterialModel.updateOne(
    {"_id": id},
    {$inc: {[`quantity.${0}`]: qty}}
    )
    .then(result => {console.log(`${result}; Updated document with _id ${id}`);})
    .catch(error => {console.error(`Error updating document with _id ${id}: ${error}`)});
```

Добавление материала в складе

```
MaterialModel.updateOne(
    {"_id": id},
    {$inc: {[`quantity.${1}`]: qty}}
    )
    .then(result => {console.log(`${result}; Updated document with _id ${id}`);})
    .catch(error => {console.error(`Error updating document with _id ${id}: ${error}`)});
```

Создание нового материала

```
 MaterialModel.insertMany([{
    name: material.name,
    type: material.type,
    unity: material.unit,
    quantity: [0, 0],
    description: material.description
  }]).then((result) => {
    console.log("result: ", result);
  })
```

Нахождение материала

```
MaterialModel.findById(id).lean()
```

Список накладных с соответствующими материалами

```
InvoiceModel.aggregate([
      {
        $unwind: "$contain"
       },
       {
        $group: {
          _id: null,
          material_ids: { $addToSet: "$contain.material_id" }
        }
       },

       {
        $lookup: {
          from: "materials",
          localField: "material_ids",
          foreignField: "_id",
          as: "materials"
        }
       },

       {
        $unwind: "$materials"
       },

       {
```

```
    $project: {
      _id: 0,
      material_id: "$materials._id",
      name: "$materials.name"

    }
  }
]);
```

Статистика для посмотра потребления материала рабочими

```
StatisticsModel.aggregate([
    {
      $match:{
         "contain.material_id": new mongoose.Types.ObjectId(material_id)
      }
    },
    {
      $unwind: "$contain"
    },

    {
      $lookup: {
         from: "users",
         localField: "person",
         foreignField: "_id",
         as: "person"
      }
    },
```

```
{
   $unwind: "$person"
},

{
   $project:{
      _id: 0,
      person: "$person.username",
      role: "$person.role",
      date: "$date"
   }
},

{
   $group: {
      _id: { person: "$person", role: "$role", date: "$date" },
      count: { $sum: 1 } // To count occurrences of each distinct document
    }
},

{
   $project: {
      _id: 0, // Exclude _id field from the output
      person: "$_id.person",
      role: "$_id.role",
      date: "$_id.date",
   }
},
```

```
    {
      $sort: {
        "date" : 1,
        "person": 1
      }
    }
  ])
```

Статистика для посмотра потребления на год

```
await StatisticsModel.aggregate([
    {
      $match: {
        "contain.material_id": new mongoose.Types.ObjectId(material_id),
        date: {
          $gte: startDate,
          $lte: endDate
        }
      }
    },
    {
      $unwind: "$contain"
    },
    {
      $group: {
        _id: "$date",
        totalUsage: { $sum: "$contain.quantity" }
      }
    },
```

```
    {
      $sort: {
        _id: 1
      }
    }


  ]);
```

Статистика для построения графика об использовании материала

```
StatisticsModel.aggregate([

    {
      $match: {
        "contain.material_id": new mongoose.Types.ObjectId(material_id)
      }
    },


    {
      $unwind: "$contain"
    },
    {
      $group: {
        _id: "$person",
        totalQuantity: { $sum: "$contain.quantity" }
      }
    },
    {
```

```
        $lookup: {
            from: "users",
            localField: "_id",
            foreignField: "_id",
            as: "worker"
        }
    },
    {
        $unwind: "$worker"
    },
    {
        $project: {
            _id: 0,
            worker_id: "$worker._id",
            worker_username: "$worker.username",
            totalQuantity: 1
        }
    }
]);
```

Реляционная модель

Графическое представление

users

```
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(255) NOT NULL,
    avatar_name VARCHAR(255),
```

```
    avatar_content LONGBLOB,
    avatar_type VARCHAR(255)
);
```

materials

```
CREATE TABLE materials (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    type VARCHAR(255) NOT NULL,
    quantity DECIMAL(10,2),
    unity VARCHAR(255),
    creation_date DATE,
    description TEXT
);
```

invoices

```
CREATE TABLE invoices (
    id INT AUTO_INCREMENT PRIMARY KEY,
    type VARCHAR(255) NOT NULL,
    creation_date DATE,
    author VARCHAR(255),
    FOREIGN KEY (author) REFERENCES users(username)
);
```

invoices_contain

```
CREATE TABLE invoices_contain (
    invoice_id INT,
    material_id INT,
    quantity INT,
    PRIMARY KEY (invoice_id, material_id),
    FOREIGN KEY (invoice_id) REFERENCES invoices(id),
```

```
    FOREIGN KEY (material_id) REFERENCES materials(id)
);
```

statistics

```
CREATE TABLE statistics (
    id INT AUTO_INCREMENT PRIMARY KEY,
    date DATE,
    person_id INT,
    FOREIGN KEY (person_id) REFERENCES users(id)
);
```

statistics_contain

```
CREATE TABLE statistics_contain (
    statistic_id INT,
    material_id INT,
    quantity INT,
    PRIMARY KEY (statistic_id, material_id),
    FOREIGN KEY (statistic_id) REFERENCES statistics(id),
    FOREIGN KEY (material_id) REFERENCES materials(id)
);
```

Оценка объема информаций, хранимой в модели

 users

Допустим что имеется x пользователей.

_id – тип ObjectID V = 12 байт

username – VARCHAR V = 30 байт

password – VARCHAR V = 15 байт

role – VARCHAR V = 10 байт

avatar_name – VARCHAR V = 20 байт

avatar_content – LONGBLOB = 4000 байт

avatar_type – VARCHAR V = 8 байт

 materials.

Допустим что имеется у материалы

_id – тип ObjectID V = 12 байт

name – VARCHAR V = 30 байт

type – VARCHAR V = 15 байт

stock – тип Object V = 8 + 8 = 16 байт

positions – Int V = 8 байт

stock – Int V = 8 байт

unit – VARCHAR V = 6 байта

creation – Date V = 8 байт

description – VARCHAR V = 300 байт

 invoices.

Допустим что для каждого накладного в среднем mat = 10 различные материалы. Тогда для хранения х накладных прнадобится:

_id – тип ObjectID V = 12 байт

type – VARCHAR V = 10 байт

creation – Date V = 8 байт

author – VARCHAR V = 30 байт

contain – тип Object V = 12 + 8= 20 байт

material_id – ObjectID V = 12 байт

quantity – Int V = 8 байт

 statistics.

Допустим что для каждого потребления в среднем используются mat = 10 различные материалы. Тогда для хранения x потреблений прнадобится:

_id – тип ObjectID V = 12 байт

date – Date V = 8 байт

person – Int V = 12 байт

contain – тип Object V = 12 + 8 = 20 байт

Тогда получаем следующий объём данных для хранения x пользователей и y материалов:

Запросы к модели, с помощью которых реализуются сценарии использования

Запрос для статистики использования материалов между датами, отсортированную по дате

```
SELECT
    u.username AS person,
    u.role,
    s.date,
    COUNT(*) AS count
FROM
    statistics s
JOIN
```

```
    users u ON s.person_id = u.id
JOIN
    statistics_contain sc ON s.id = sc.statistic_id
WHERE
    sc.material_id = :material_id
GROUP BY
    u.username,
    u.role,
    s.date
ORDER BY
    s.date ASC,
    u.username ASC;
```

Запрос для статистики использования материалов по лицу, отсортированную по общему количеству

```
SELECT
    u.id AS worker_id,
    u.username AS worker_username,
    SUM(sc.quantity) AS totalQuantity
FROM
    statistics s
JOIN
    users u ON s.person_id = u.id
JOIN
    statistics_contain sc ON s.id = sc.statistic_id
WHERE
    sc.material_id = :material_id
GROUP BY
    u.id,
```

```
    u.username;
```

Запрос для статистики использования материалов по лицу и роли, отсортированную по дате и лицу

```
SELECT
    u.username AS person,
    u.role,
    s.date,
    COUNT(*) AS count
FROM
    statistics s
JOIN
    users u ON s.person_id = u.id
JOIN
    statistics_contain sc ON s.id = sc.statistic_id
WHERE
    sc.material_id = :material_id
GROUP BY
    u.username,
    u.role,
    s.date
ORDER BY
    s.date ASC,
    u.username ASC;
```

Запрос для статистики использования материалов между датами

```
SELECT
    date,
    SUM(sc.quantity) AS totalUsage
FROM
```

```
    statistics s
JOIN
    statistics_contain sc ON s.id = sc.statistic_id
WHERE
    sc.material_id = :material_id
    AND date BETWEEN :start_date AND :end_date
GROUP BY
    date
ORDER BY
    date ASC;
```

Запрос для статистики использования материалов по лицу, отсортированную по общему количеству

```
SELECT
    u.id AS worker_id,
    u.username AS worker_username,
    SUM(sc.quantity) AS totalQuantity
FROM
    statistics s
JOIN
    users u ON s.person_id = u.id
JOIN
    statistics_contain sc ON s.id = sc.statistic_id
WHERE
    sc.material_id = :material_id
GROUP BY
    u.id,
    u.username
ORDER BY
```

totalQuantity DESC;

Сравнение моделей

NoSQL требует больше памяти, по сравнению с SQL, так в нем дублируются некоторые данные. SQL выигрывает по памяти, так как вместо того, чтобы хранить сами объекты целиком (как NoSQL), он хранит только id определенного элемента из другой таблицы. По удобству запросов выигрывает NoSQL, так как ввиду дублирования данных в некоторых сущностях, нам не приходится JOIN-ить с другими коллекциями. Для некоторых запросов SQL приходится JOIN-ить несколько таблиц, что может сказать в скорости доступа.

Выводы

NoSQL требует больше памяти, но удобен и быстр в запросах, в то время как SQL требует меньше памяти и не совсем удобен в запросах, из-за того что приходится JOIN-ить таблицы. Для данной задачи NoSQL подходит лучше, чем SQL.

3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

3.1. Описание

Backend реализован при помощи фреймворка express на базе nodejs на языке программирования JavaScript с использованием базы данных MongoDB. Frontend реализован с помощью Ejs, CSS и Javascript .

3.2. Использованные технологии

База данных: MongoDB

Backend: Nodejs, express, Javascript.

Frontend: Ejs, CSS и Javascript.

3.3. Снимки экрана приложения

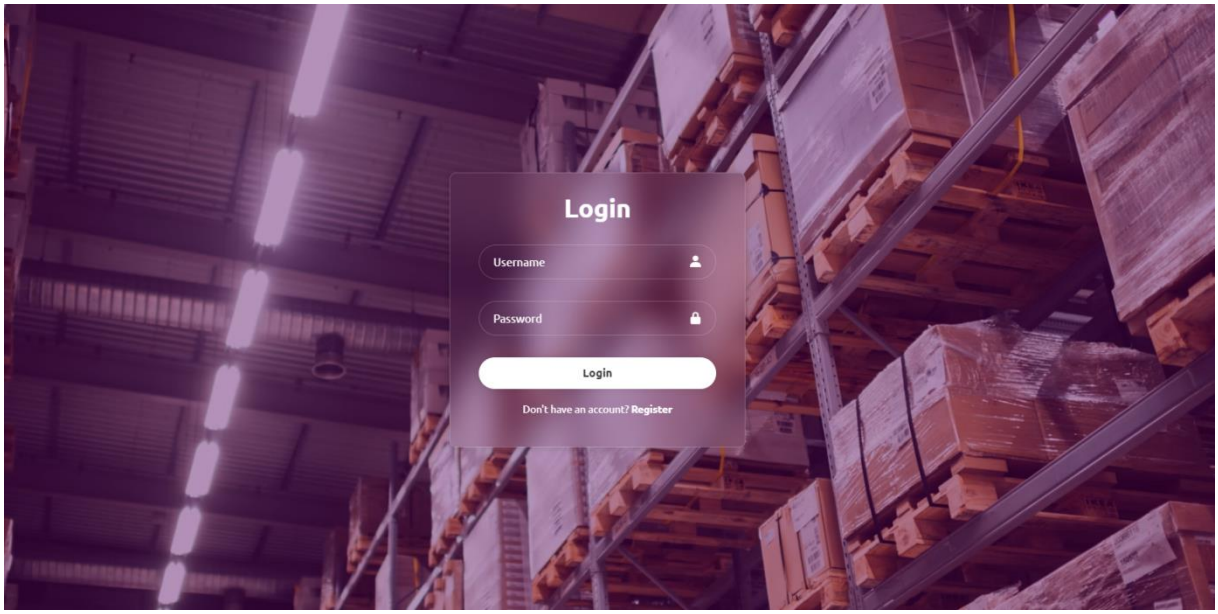


Рисунок 6 – Страница входа

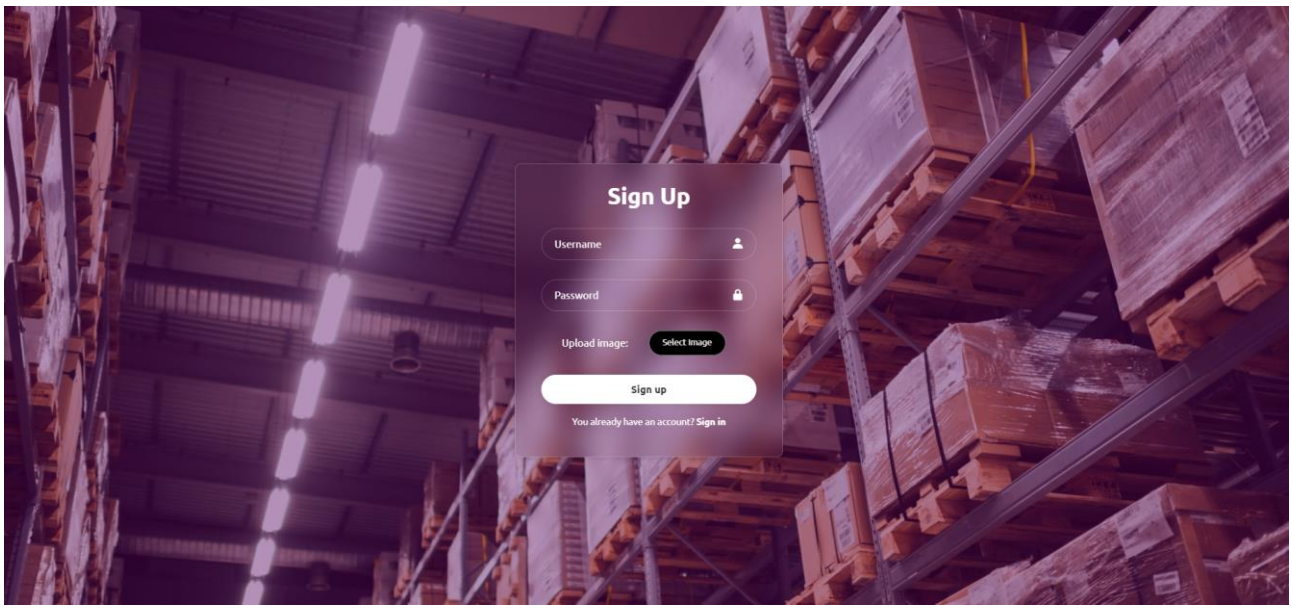
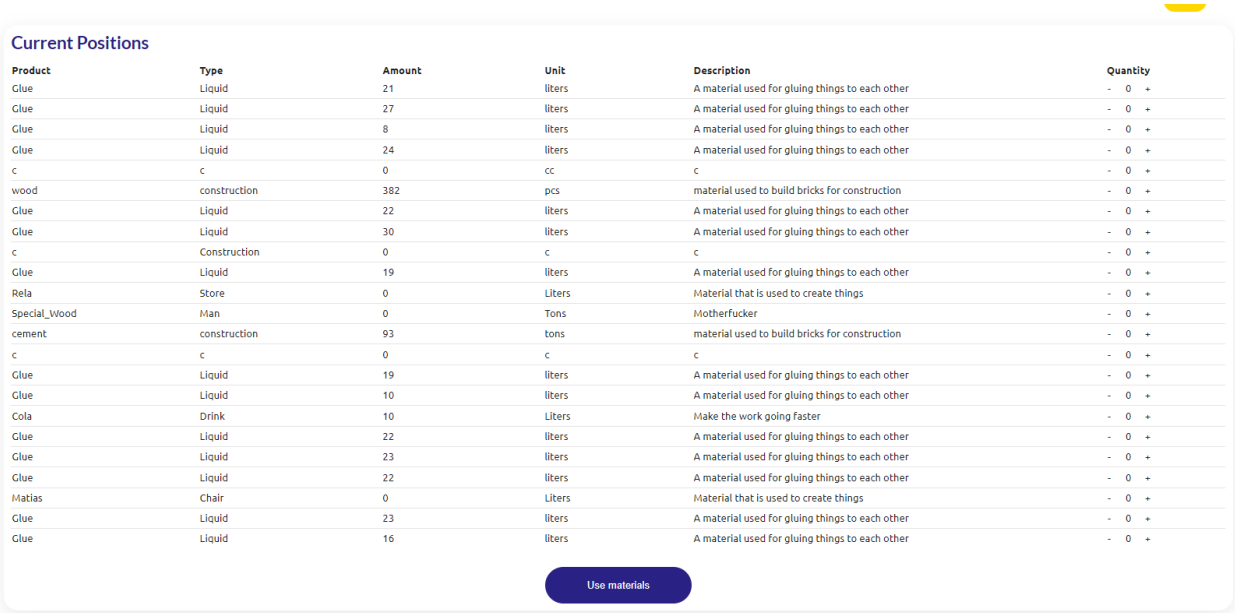


Рисунок 7 – Страница регистрации



Current Positions

Product	Type	Amount	Unit	Description	Quantity
Glue	Liquid	21	liters	A material used for gluing things to each other	- 0 +
Glue	Liquid	27	liters	A material used for gluing things to each other	- 0 +
Glue	Liquid	8	liters	A material used for gluing things to each other	- 0 +
Glue	Liquid	24	liters	A material used for gluing things to each other	- 0 +
c	c	0	cc	c	- 0 +
wood	construction	382	pcs	material used to build bricks for construction	- 0 +
Glue	Liquid	22	liters	A material used for gluing things to each other	- 0 +
Glue	Liquid	30	liters	A material used for gluing things to each other	- 0 +
c	Construction	0	c	c	- 0 +
Glue	Liquid	19	liters	A material used for gluing things to each other	- 0 +
Rela	Store	0	Liters	Material that is used to create things	- 0 +
Special_Wood	Man	0	Tons	Motherfucker	- 0 +
cement	construction	93	tons	material used to build bricks for construction	- 0 +
c	c	0	c	c	- 0 +
Glue	Liquid	19	liters	A material used for gluing things to each other	- 0 +
Glue	Liquid	10	liters	A material used for gluing things to each other	- 0 +
Cola	Drink	10	Liters	Make the work going faster	- 0 +
Glue	Liquid	22	liters	A material used for gluing things to each other	- 0 +
Glue	Liquid	23	liters	A material used for gluing things to each other	- 0 +
Glue	Liquid	22	liters	A material used for gluing things to each other	- 0 +
Matias	Chair	0	Liters	Material that is used to create things	- 0 +
Glue	Liquid	23	liters	A material used for gluing things to each other	- 0 +
Glue	Liquid	16	liters	A material used for gluing things to each other	- 0 +

Use materials

Рисунок 7 – Список материалов на складе для использования

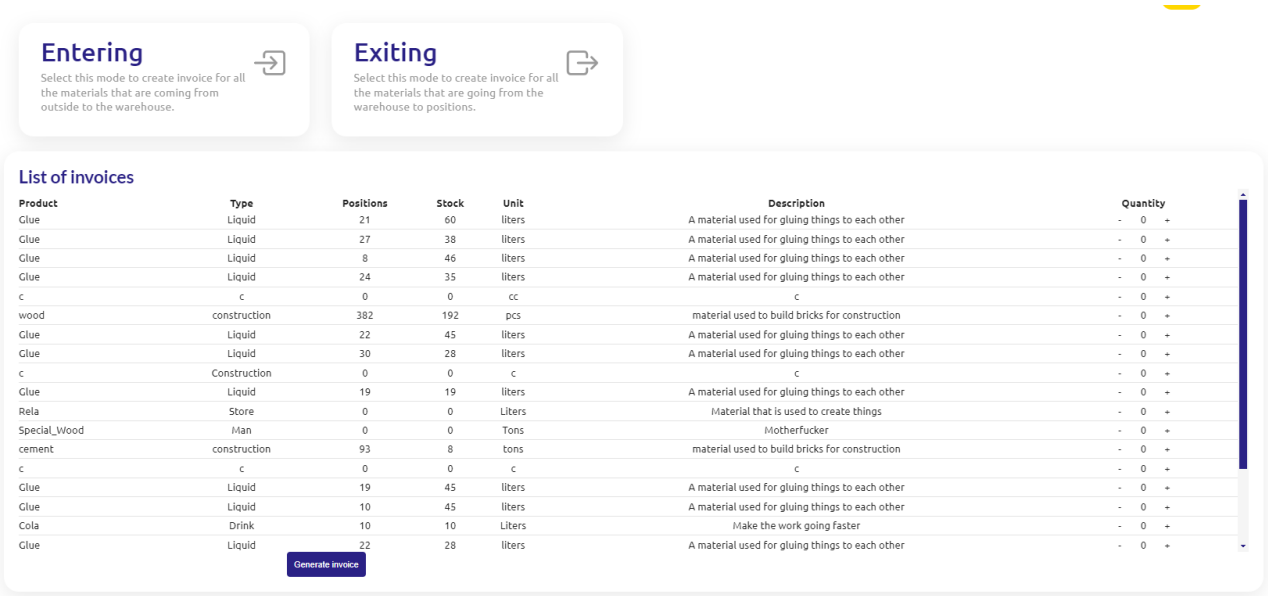


Рисунок 8 – Список материалов для заказа

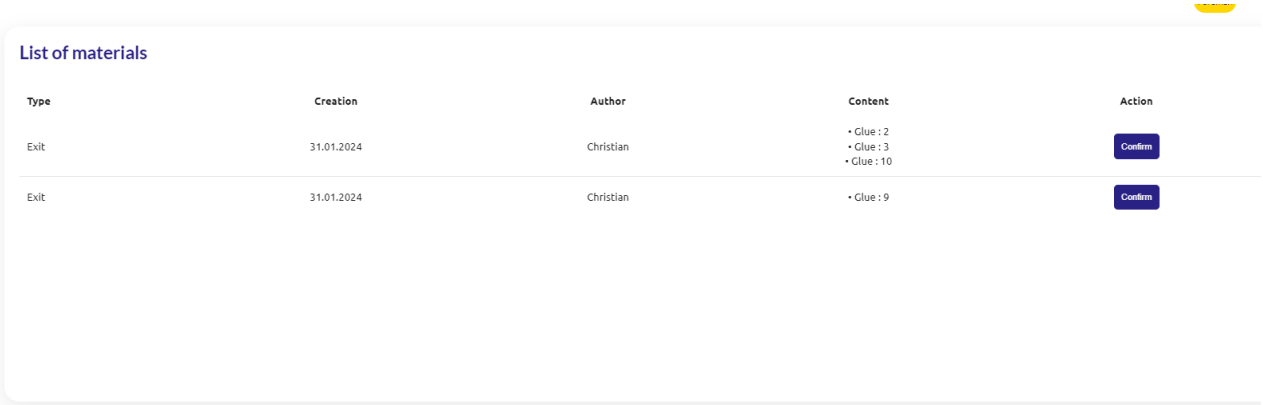


Рисунок 9 – страница подтверждения заказа (Страница админа)

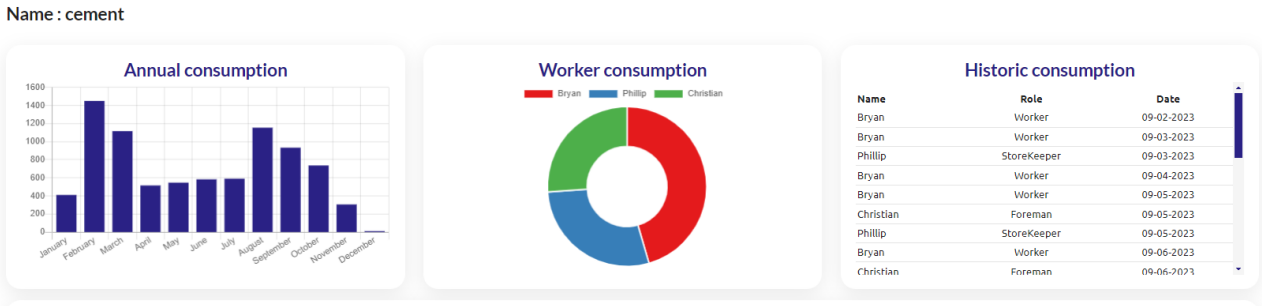


Рисунок 10 – статистика использования материалов

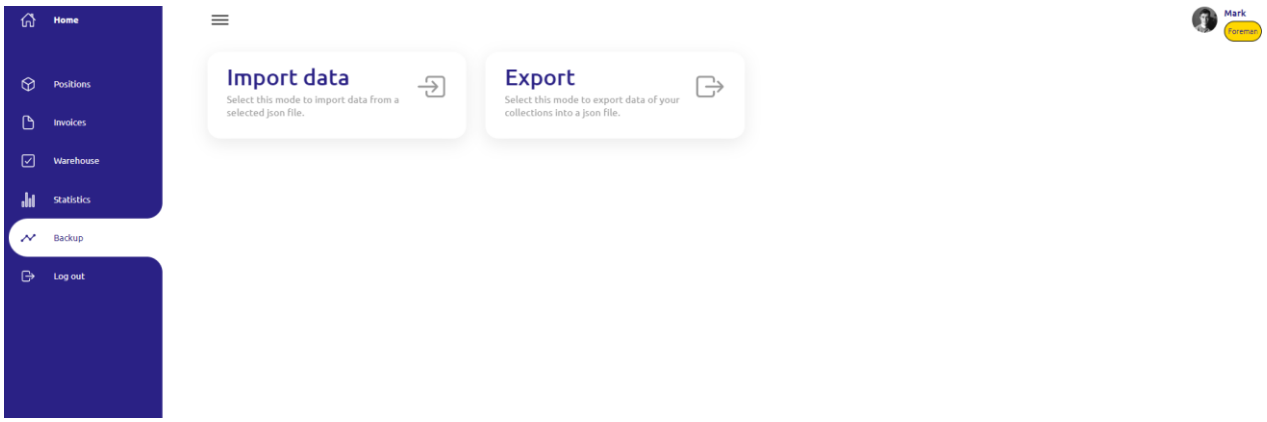


Рисунок 11 – страница загрузки данных из бд

ЗАКЛЮЧЕНИЕ

В результате работы было разработано веб-приложение “ Умный склад на стройплощадке ”, которое позволяет хранить информацию о количестве товаров на стройплощадке, а так же заказывать и распределять товары по складу .

a. Недостатки и пути для улучшения полученного решения

Добавление страницы для регистрации пользователей по ролям

Добавление возможности заказывать товар не по количеству упаковок, а по весу

Добавление возможности распределять товар по складу для быстрого доступа

Будущее развитие решения

Разработка мобильной версии приложения

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий веб-приложения: [электронный ресурс]. URL: https://github.com/moevm/nosql2h23-construction
2. MongoDB The Developer Data Platform: [электронный ресурс]. URL: https://www.mongodb.com/

ПРИЛОЖЕНИЕ А

ДОКУМЕНТАЦИЯ ПО СБОРКЕ И РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ

1. Скачать проект из репозитория (указан в ссылках на приложение)
2. Установить зависимости для клиентской части (в папке frontend) и для серверной части (в папке backend)
3. В папке backend запустить веб-сервер через команду npm run devStart
4. В папке frontend запустить клиентскую часть через команду npm run dev
5. Открыть приложение в браузере по адресу http://localhost:5173/