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

Пояснительная записка
по дисциплине «Введение в нереляционные базы данных»
Тема: Хаб данных умной фермы

Студенты гр. 0383 Позолотин К.С.

Рудакова Ю.В.

Желнин М.Ю.

Преподаватель Заславский М.М.

Санкт-Петербург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

1. Актуальность решаемой проблемы:

Умные фермы представляют собой важную составляющую современного сельского хозяйства, обеспечивая эффективное и экологически устойчивое производство. Автоматизированный мониторинг тепличной фермы помогает оптимизировать процессы, повышать урожайность, снижать издержки и обеспечивать более качественное управление всеми аспектами производства.

2. Постановка задачи:

Необходимо создать приложение для мониторинга и управления тепличной фермой. Пользователи приложения включают в себя рабочих, бригадиров и владельцев бизнеса. Основные задачи включают в себя следующие:

- Создание и управление заданиями для рабочих и бригадиров.
- Ведение учета собранной продукции и расходников на складе.
- График дежурств для эффективного планирования и использования трудовых ресурсов.
- Отображение информации об инфраструктуре фермы.

3. Предлагаемое решение:

Для решения поставленных задач предлагается использовать комбинацию баз данных InfluxDB и MongoDB. InfluxDB подходит для хранения временных рядов, что позволяет эффективно отслеживать изменения в показателях, таких как температура, влажность и другие параметры окружающей среды в тепличной ферме. MongoDB будет использоваться для хранения структурированных данных о пользователях, нарядах, складе и инцидентах.

4. Качественные требования к решению:

- Интуитивный интерфейс: Приложение должно иметь простой и интуитивно понятный интерфейс для обеспечения удобства использования пользователями с различным уровнем технической грамотности.
- Многоролевость: Система должна поддерживать разные уровни доступа и роли для рабочих, бригадиров и владельцев бизнеса.
- Мониторинг параметров фермы: Приложение должно обеспечивать мониторинг и отображение данных о параметрах фермы, таких как температура, влажность, освещение и другие.
- Управление заданиями: Возможность создания, назначения и отслеживания заданий для рабочих и бригадиров.
- Учет склада: Система должна поддерживать учет собранной продукции и расходников на складе, а также предоставлять отчеты о запасах.
- График дежурств: Возможность создания и отображения графика дежурств для эффективного распределения трудовых ресурсов.
- Отчетность: Генерация отчетов о производственной деятельности, урожайности, расходах и других ключевых показателях.
- Масштабируемость и надежность: Архитектура приложения должна быть масштабируемой и надежной для обеспечения стабильной работы в условиях изменяющихся объемов данных и нагрузок.

СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ.

Макет UI.

https://www.figma.com/file/HeXxBHSoMTLj3TiryZWB5m/SmartFarm-UseCase?type=design&mode=design&t=Ib3ooyJV46KEXzMc-0

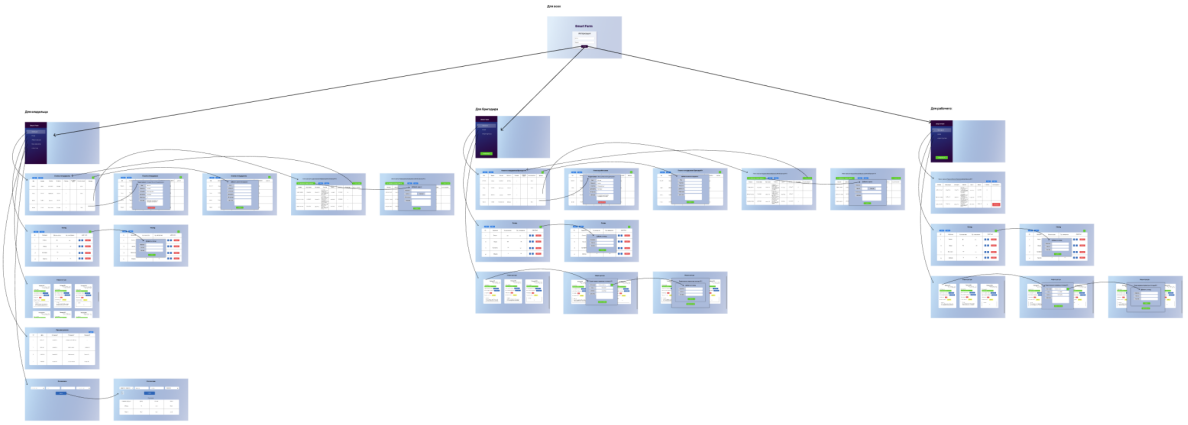

Рисунок 1 - Скриншот Макета UI

Сценарий использования:

1. Сценарий использования - «Просмотреть историю событий, чтобы выявить человека, который собрал урожай в теплице №2»:

Действующее лицо: Владелец

Основной сценарий:

- Пользователь авторизуется.
- Попадает на экран с открытым боковым меню.
- Пользователь выбирает из меню “Просмотр логов”.
- Появляется страница просмотра логов.
- Пользователь сортирует таблицу, чтобы отображались только действия со складом и с теплицей №2
- Пользователь находит кто собрал урожай в теплице № 2

Альтернативный сценарий:

- После сортировки таблица пустая
- Пользователь понимает, что урожай в теплице № 2 еще не собран

2. Сценарий использования - «Добавить нового пользователя в систему»:

Действующее лицо: Владелец

Основной сценарий:

- Пользователь авторизируется.
- Попадает на экран с открытым боковым меню.
- Пользователь выбирает из меню “Работники”.
- Появляется страница со списком сотрудников.
- Пользователь нажимает на кнопку +.
- Появляется диалоговое окно “Добавить нового сотрудника”
- Пользователь заполняет поля и нажимает на кнопку “Добавить”
- Пользователь видит в списке сотрудников только что добавленного им нового сотрудника.

Альтернативный сценарий:

- Пользователь заполняет поля и нажимает на крестик
- Введенные данные не сохраняются, таблица с сотрудниками остается неизменной

3. Сценарий использования - «Проверить все ли сотрудники вышли на свою смену»:

Действующее лицо: Бригадир

Основной сценарий:

- Пользователь авторизируется.
- Попадает на экран с открытым боковым меню.
- Пользователь выбирает из меню “Работники”.
- Появляется страница со списком сотрудников.
- Пользователь видит только сотрудников из его бригады, оценивает столбец “На рабочем месте”
- Если отображены все галочки, значит все сотрудники на месте

Альтернативный сценарий:

- Если отображены и галочки, и крестики, значит все не все сотрудники на месте, или еще не успели отметиться.

4. Сценарий использования - «Проверить статус теплицы № 2»:

Действующее лицо: Бригадир

Основной сценарий:

- Пользователь авторизируется.
- Попадает на экран с открытым боковым меню.
- Пользователь выбирает из меню “Инфраструктура”.
- Появляется страница с карточками теплиц.
- Пользователь ищет карточку с нужной ему теплицей
- Пользователь находит карточку и в окошке со статусом видит состояние теплицы № 2

Альтернативный сценарий:

- Если пользователь не находит карточку с теплицей № 2, значит уход за данной теплицей не входит в его обязанности

5. Сценарий использования - «Поменять статус теплицы №5 на “пустая”, занести в склад 10 кг помидор, собранных в этой теплице»:

Действующее лицо: Рабочий

Основной сценарий:

- Пользователь авторизируется.
- Попадает на экран с открытым боковым меню.
- Пользователь выбирает из меню “Инфраструктура”.
- Появляется страница с карточками теплиц.
- Пользователь ищет карточку с нужной ему теплицей
- Пользователь находит карточку и нажимает кнопку редактирования
- Открывается диалоговое окно “Редактировать параметры теплицы № 5”
- Пользователь меняет статус на “пустая”, затем нажимает +
- Открывается диалоговое окно “Добавить в склад”

- Пользователь заполняет поля и нажимает кнопку “Добавить”
- Диалоговое окно “Добавить в склад” закрывается
- В окне “Редактировать параметры теплицы № 5” пользователь нажимает кнопку “Сохранить изменения”
- Диалоговое окно закрывается
- Пользователь видит в карточке теплицы №5 измененный статус
- Пользователь открывает боковое меню и выбирает “Склад”.
- Пользователь видит запись с 10 кг помидоров.

Альтернативный сценарий:

- Если пользователь не находит карточку с теплицей № 5, значит уход за данной теплицей не входит в его обязанности.
- Если пользователь заполняет поля и нажимает на крестик в любом диалоговом окне, то введенные данные не сохраняются, таблица с сотрудниками остается неизменной

МОДЕЛЬ ДАННЫХ.

Нереляционная модель.

Графическое представление.

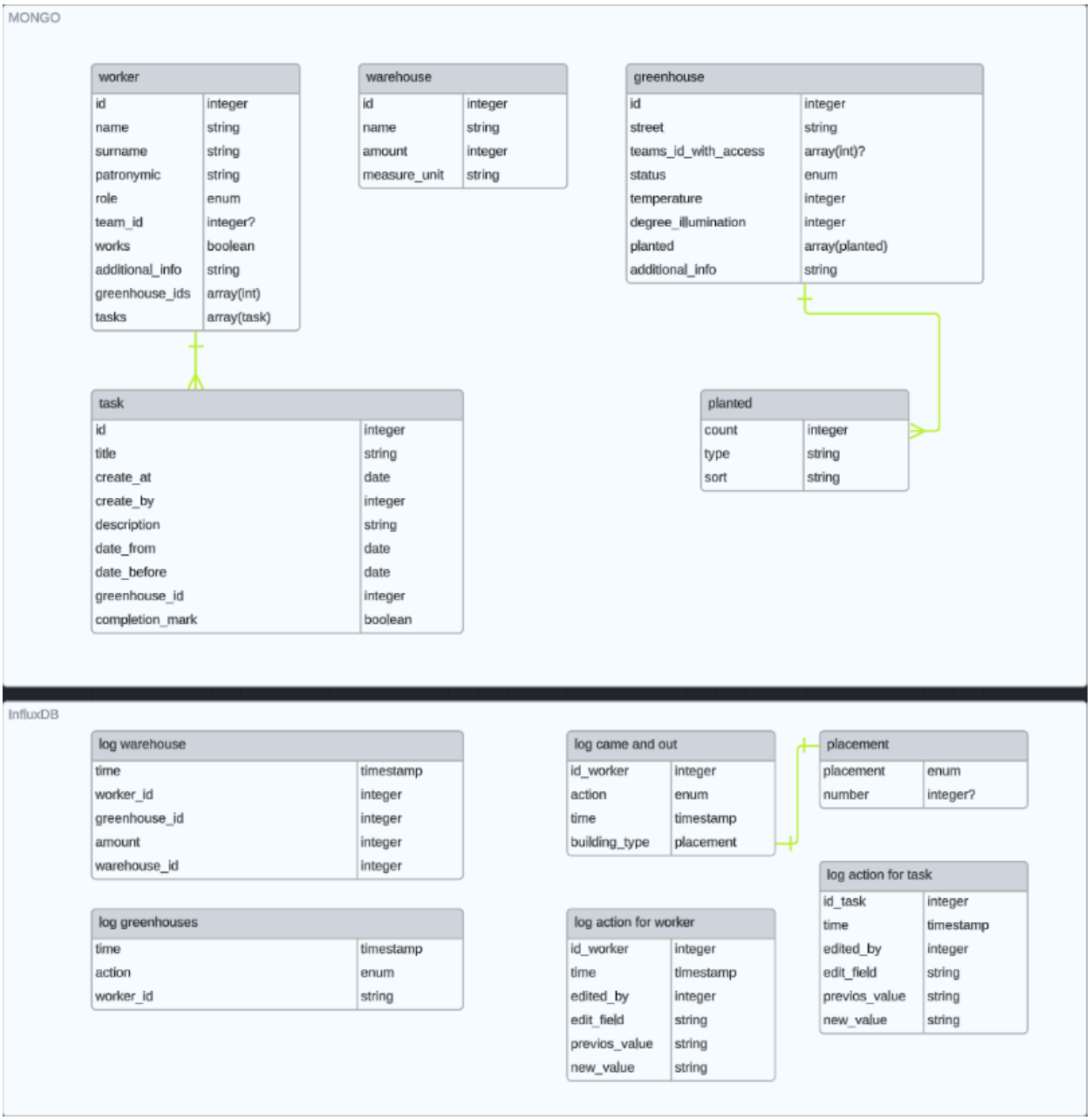

Рисунок 2 - ER-диаграмма модели данных

Подробное описание коллекций и сущностей:

Для идентификаторов в MongoDB, из-за неспособности правильно интегрировать тип Object ID в InfluxDB, был использован тип Integer.

БД содержит 8 коллекций:

1. worker - информация о работнике и его задачах
2. warehouse - информация о складе
3. greenhouse - информация о теплице и посаженных в ней растений
4. log warehouse - лог склада
5. log greenhouse - лог теплицы
6. log came and out - информация о времени прибытии и ухода работника
7. log action for worker - хранения действий над работниками
8. log action for task - хранения действий над задачами

Коллекция worker:

- id - уникальный номер
- name - имя
- surname - фамилия
- patronymic - отчество
- role - должность
- team_id - номер бригады
- works - наличие сотрудника на рабочем месте
- additional_info - дополнительная информация о сотруднике
- greenhouse_ids - теплицы в которых выполняет работу
- task - информацию о задании
    - id - уникальный номер
    - title - название
    - create_at - дата создания
    - create_by - уникальный номер создателя
    - description - описание
    - date_from - срок выполнения(начало)
    - date_before - срок выполнения(конец)
    - completion_mark - марка выполнения

    - worker_id - уникальный номер работника

Коллекция warehouse:

- id - уникальный номер
- name - имя
- amount - количество предмета на складе
- measure_unit - единица измерения

Коллекция greenhouse:

- id - уникальный номер
- street - название улицы
- teams_id_with_access - команды которые имеют доступ к теплице
- status - статус урожая
- temperature - температура
- degree_illumination - количество света
- additional_info - дополнительная информация
- planted - информация о посаженных растениях
    - count - количество
    - type - тип растения
    - sort - сорт растения

Коллекция log warehouse:

- time - время изменения
- worker_id - номер работника
- greenhouse_id - номер теплицы
- warehouse_id - номер склада
- amount - новое значения количества

Коллекция log greenhouse:

- time - время
- action - действие

- worker_id - номер работника
- greenhouse_id - номер склада

Коллекция log come and out:

- id_worker - номер работника
- action - действие
- time - время
- building_type - информация о локации
- placement - название локации
- number - номер склада

Коллекция log action for worker:

- id_worker - номер работник
- time - время
- edited_by - номер редактора
- edit_field - поле которые было отредактировано
- previos_value - значение до редактирования
- new_value - значение после редактирования

Коллекция log action for task:

- id_task - номер задачи
- time - время
- edited_by - номер редактора
- edit_field - поле которые было отредактировано
- previos_value - значение до редактирования
- new_value - значение после редактирования

worker role(enum):

- "Бригадир"
- "Рабочий"

- "Администратор"

greenhouse status(enum):

- "Посажены саженцы"
- "Готовы к сбору урожая"
- "Пустая теплица"

log came and out action(enum):

- "Пришел"
- "Ушел"

Оценка удельного объема.

Пусть:

- количество сущностей "worker" - nWork (Предположим, ~50 рабочих)
- количество сущностей "warehouse" - nWar (Предположим, ~70 продуктов)
- количество сущностей "greenhouse" - nG (Предположим, ~20 теплиц)

Если в среднем у каждого рабочего по 2 задания = 50 * 2 = 100 Если в среднем в каждую теплице по 3 продукта = 20 * 3 = 60

Объем всех атрибутов сущности worker = (12(id) + 64(name) + 80(surname) + 80(patronymic) + 100(role) + 8(team_id) + 1(works) + 300(additional_info) + 8(greenhouse_ids) + 500(tasks))* nWork = 89 Кбайт

Объем всех атрибутов сущности warehouse = (12(id) + 64(name) + 8(amount) + 8(measure_unit))* nWar = 80 байт

Объем всех атрибутов сущности greenhouse = (12(id) + 64(street) + 32(teams_id_with_access) + 32(status) + 8(temp) + 8(degree_illumination) + 150(planted) + 300(additional_info))* nG = 11 Кбайт

Общий объем: 110 Кбайт, 641*n+8444 байт, где n - количество работников

Избыточность объема.

Пусть количество посаженных растений равно 20, а количество задач 10, тогда отношение между фактическим объемом модели, и чистыми данными для

n(количество работников):((1141+50010)n+80+59420)/(641n+10500+20150+444) = (6141n+11960)/(641n+8444), следовательно объем БД примерно в 10 раз больше чистого объема данных.

Направление роста модели при увеличении количества объектов каждой сущности:

При создании сущностей worker и greenhouse, автоматически создаются сущности task и planted соответственно. Это ускоряет рост модели при увеличении количества данных сущностей.

Запросы к модели, с помощью которых реализуются сценарии использования:

Добавление нового работника с его задачами:

```
db.workers.insert({
    ‘id’: id,
    ‘name’: name,
    ‘surname’: surname,
    ‘patronymic’: patronymic,
    ‘role’: role,
    ‘team_id’: team_id,
    ‘works’: works,
    ‘additional_info’: additional_info,
    ‘greenhouse_ids’: greenhouse_ids,
    tasks: [{
    ‘id’: id,
    ‘title’: title,
    ‘create_at’: create_at,
    ‘create_by’: create_by,
    ‘description’: description,
    ‘date_from’: date_from,
```

‘date_before’: date_before,
‘greenhouse_id’: greenhouse_id,
‘completion_mark’: completion_mark
}]
})

Добавление склада:
db.warehouse.insert({
‘id’: id,
‘name’: name,
‘amount’: amount,
‘measure_unit’: measure_unit,
}),

Добавление теплицы:
db.greenhouse.insert({
‘id’: id,
‘street’: street,
‘teams_id_with_access’: teams_id_with_access,
‘status’: status,
‘temp’: temp,
‘degree_illumination’: degree_illumination,
‘additional_info’: additional_info,
‘planted’: [
{
‘count’: count,
‘type’: type,
‘sort’: sort
}]}

Работники, у которых закрыто меньше всего задач:

```
db.worker.aggregate([
 {
  $match: {
   works: true
  }
 },
 {
  $project: {
   id: 1,
   name: 1,
   Surname: 1,
   patronymic: 1,
   role: 1,
   team_id: 1,
   works: 1,
   additional_info: 1,
   greenhouse_ids: 1,
   tasks: {
    $filter: {
     input: "$tasks",
     as: "task",
     cond: { $eq: ["$$task.completion_mark", true] }
    }
   },
   numCompletedTasks: {
    $size: {
     $filter: {
      input: "$tasks",
      as: "task",
```

```
          cond: { $eq: ["$$task.completion_mark", true] }
        }
       }
     }
   }
  },
  {
   $sort: {
    numCompletedTasks: 1
   }
  },
  {
   $limit: 1
  }
])
```

Работники, проводящие больше всего времени в теплицах:

```
db.worker_actions.aggregate([
  {
   $match: {
          "building_type.placement": "greenhouse"   // Фильтруем только записи, связанные с теплицами
   }
  },
  {
   $group: {
    _id: "$id_worker",
    totalTimeInGreenhouses: {
     $sum: {
      $cond: {
```

```
            if: { $eq: ["$action", "out"] },
            then: { $subtract: ["$time", "$prevTime"] },
            else: 0
          }
        }
      },
      prevTime: { $last: "$time" }
    }
  },
  {
    $sort: {
      totalTimeInGreenhouses: -1
    }
  },
  {
    $limit: 1
  }
])
```

Реляционная модель.

Графическое представление модели.

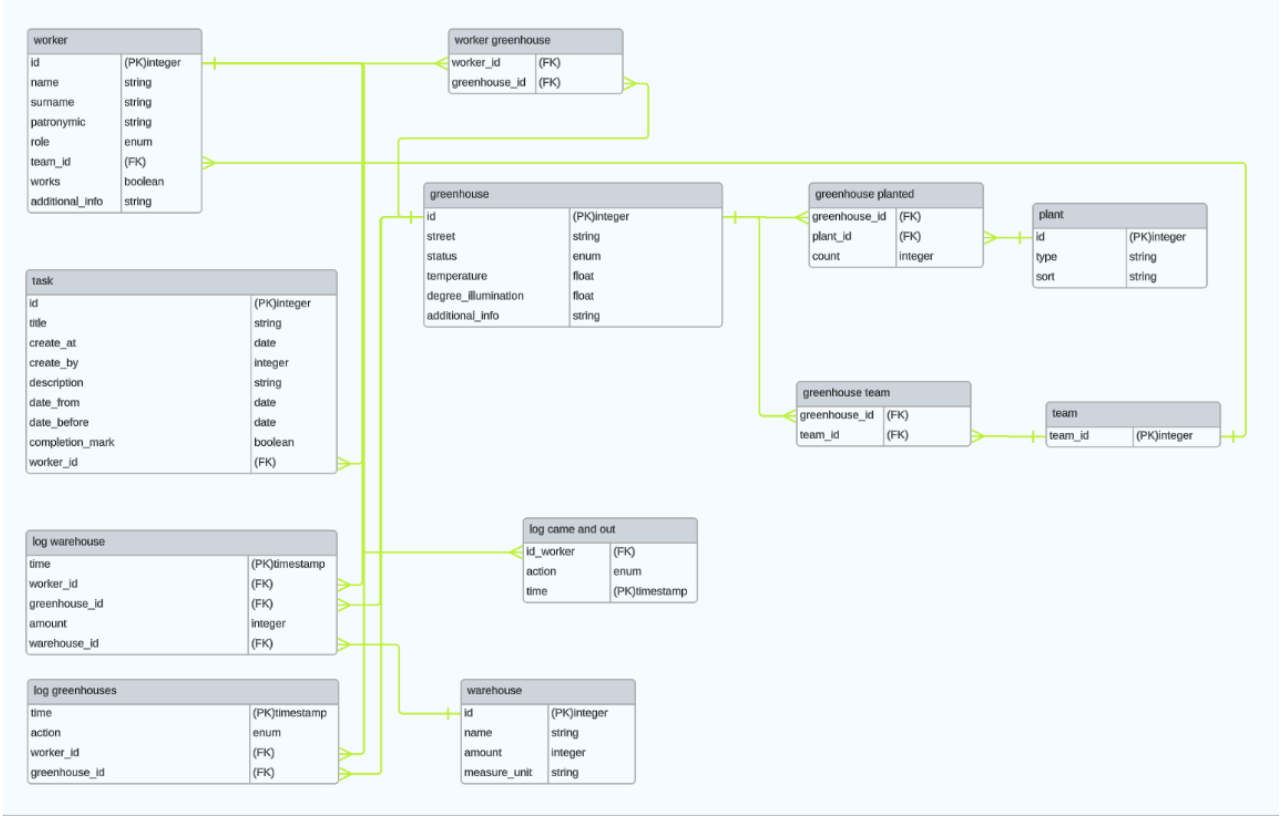

Рисунок 3 - ER-диаграмма модели данных

Подробное описание коллекций и сущностей.

Бд содержит 12 таблиц:

Таблица worker - информацию о работниках:

- id - уникальный номер
- name - имя
- surname - фамилия
- patronymic - отчество
- role - должность
- team_id - номер бригады
- works - наличие сотрудника на рабочем месте
- additional_info - дополнительная информация о сотруднике

Таблица task - информация о задании:

- id - уникальный номер
- title - название
- create_at - дата создания
- create_by - уникальный номер создателя
- description - описание
- date_from - срок выполнения(начало)
- date_before - срок выполнения(конец)
- completion_mark - марка выполнения
- worker_id - уникальный номер работника

Таблица greenhouse - информация о теплице:

- id - уникальный номер
- street - название улицы
- status - статус урожая
- temperature - температура
- degree_illumination - количество света
- additional_info - дополнительная информация

Таблица plant - информация о растении:

- id - уникальный номер
- type - тип растения
- sort - сорт растения

Таблица log warehouse - информация о логах склада

- time - время изменения
- worker_id - номер работника
- greenhouse_id - номер теплицы
- warehouse_id - номер склада
- amount - новое значения количества

Таблица log greenhouse - информация о логах теплицы

- time - время
- action - действие
- worker_id - номер работника
- greenhouse_id - номер склада

Таблица log come and out - информация о времени прибытия и ухода работника:

- id_worker - номер работника
- action - действие
- id_worker - время

Таблица greenhouse planted - соответствие теплицы и растения посаженного на нем:

- greenhouse_id - номер теплицы
- palnt_id - номер растения
- count - количество

Таблица greenhouse team - соответствие теплицы и команды:

- greenhouse_id - номер теплицы
- team_id - номер команды

Таблица team - информация о команде:

- team_id - уникальный номер команды

Таблица worker greenhouse - соответствие работника и теплицы:

- worker_id - номер работника
- greenhouse_id - номер теплицы

Таблица warehouse - информация о складе:

- id - уникальный номер

- name - имя
- amount - количество предмета на складе
- measure_unit - единица измерения

Оценка удельного объема:

Объем worker (307 байт):

- id - int(8 байт)
- name - string(20 байт)
- surname - string(30 байт)
- patronymic - string(20 байт)
- role - string(20 байт)
- team_id - int(8 байт)
- works - bool(1 байт)
- additional_info - string(200 байт)

Объем task (269 байт):

- id - int(8 байт)
- title - string(20 байт)
- create_at - datetime(8 байт)
- create_by - int(8 байт)
- description - string(200 байт)
- date_from - datetime(8 байт)
- date_before - datetime(8 байт)
- completion_mark - bool(1 байт)
- worker_id - int(8 байт)

Объем greenhouse (272 байт):

- id - int(8 байт)
- street - string(20 байт)
- status - string(20 байт)

- temperature - float(12 байт)
- degree_illumination - float(12 байт)
- additional_info - string(200 байт)

Объем plant (48 байт):

- id - int(8 байт)
- type - string(20 байт)
- sort - string(20 байт)

Объем log warehouse (40 байт):

- time - datetime(8 байт)
- worker_id - int(8 байт)
- greenhouse_id - int(8 байт)
- warehouse_id - int(8 байт)
- amount - int(8 байт)

Объем log greenhouse (44 байта):

- time - datetime(8 байт)
- action - string(20 байт)
- worker_id - int(8 байт)
- greenhouse_id - int(8 байт)

Объем log come and out (36 байт):

- id_worker - int(8 байт)
- action - string(20 байт)
- time - datetime(8 байт)

Объем greenhouse planted (24 байт):

- greenhouse_id - int(8 байт)
- palnt_id - int(8 байт)

- count - int(8 байт)

Объем greenhouse team (16 байт):

- greenhouse_id - int(8 байт)
- team_id - int(8 байт)

Объем team (8 байт):

- team_id - int(8 байт)

Объем worker greenhouse (16 байт):

- worker_id - int(8 байт)
- greenhouse_id - int(8 байт)

Объем warehouse (56 байт):

- id - int(8 байт)
- name - string(20 байт)
- amount - int(8 байт)
- measure_unit - string(20 байт) Объем БД, где n - количество работников: 307n+269+272+48+56+16+16+8+24 = 1016n

Избыточность объема.

Отношение между фактическим объемом модели, и чистыми данными. Избыточность вызвана отсутствием способа связать данные “многие ко многим” без использования дополнительной таблицы.

Объем БД = 1016*n

Чистый Объем БД = 952*n

Объем БД примерно в 1.06 раза больше чистого объема данных.

Направление роста модели при увеличении количества объема каждой сущности.

При создании новой сущности, никакая другая сущность автоматически не создается.

Сценарий использования.

Добавление нового работника с его задачами:

INSERT worker(name,surname, patronymic, role, team_id, works, additional_info)

VALUES(name,surname, patronymic, role, team_id, works, additional_info);

INSERT task(title, create_at, create_by, description, date_from, date_before, completion_mark, worker_id)

VALUES(title, create_at, create_by, description, date_from, date_before, completion_mark, worker_id);

Добавление склада:

INSERT warehouse(name, amount, measure)

VALUES(name, amount, measure);

Добавление теплицы:

INSERT greenhouse(street, status, temperature, degree_illumination, additional_info)

VALUES(street, status, temperature, degree_illumination, additional_info);

Сравнение моделей.

При небольшом объеме noSQL требует меньше места чем SQL. Однако, при увеличении объема данных, особенно при создании дополнительных сущностей, наподобие рабочего и теплиц, использование денормализации в noSQL может оказаться менее выгодным с точки зрения памяти.

В реляционной модели данных SQL часто требуется больше запросов для добавления новых записей. Кроме того, поиск информации о заданиях у рабочего и поиск информации о растениях, посаженных в теплице, может занимать больше времени в сравнении с NoSQL базой данных.

Вывод.

Несмотря на то что при больших данных NoSQL занимает больше места чем SQL, для данного проекта выгоднее использовать NoSQL модель за счёт быстродействия.

Примеры хранения данных:

worker

```
{
    id: 1;
    name: "Константин";
    surname: "Позолотин";
    patronymic: "Сергеевич";
    role: "Бригадир";
    team_id: 1;
    works: true;
    additional_info: "Поступил на работу 11.11.2023 11:11";
    greenhouse_ids: [1];
    tasks: [{
        id: 1;
        title: "Посадка растений";
        create_at: 11.11.2023 11:11;
        create_by: 3;
        description: "Посадить морковь";
        date_from: 11.11.2023 11:11;
        date_before: 12.11.2023 11:11;
```

```
            greenhouse_id: 1;
            completion_mark: false;

    }]
},
warehouse:

{
      id: 1;
      name: "Лопаты";
    amount: 20;
      measure_unit: "шт.";
},
```

РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ.

Краткое описание.

Разработанное приложение для автоматизированного мониторинга тепличной фермы овощей состоит из нескольких модулей, обеспечивающих различные функциональные возможности:

1. Модуль Пользователей:
    - Создание, редактирование и удаление пользователя.
    - Создание, редактирование и удаление заданий.
    - Назначение заданий рабочим и бригадирам.
    - Отслеживание статуса выполнения нарядов.
    - Фильтрация и сортировка данных
2. Модуль Склада:
    - Учет собранной продукции.
    - Отслеживание расходников.
    - Фильтрация и сортировка данных
3. Модуль Теплицы:
    - Отображение информации об инфраструктуре фермы.
    - Мониторинг параметров фермы.
    - Фильтрация и сортировка данных
4. Модуль Логирования:
    - Добавления логов
    - Просмотр логов
    - Фильтрация и сортировка данных

Архитектура:

Архитектура приложения основана на монолитной структуре для упрощение процесса разработки. Общение с базой данных и отображение данных тесно связаны.

Использованные технологии:

1. Фронтенд:
    - Blazor и Radzen для создания динамических представлений и соединения логики и отображения.
    - HTML, CSS для верстки.
2. Бэкенд:
    - C# и фреймворк ASP.NET Core для создания серверной части приложения.
    - MongoFramework для удобного взаимодействия с базой данных MongoDB.
3. База данных:
    - MongoDB для структурированных данных.
    - InfluxDB для временных рядов.

4. Контейнеризация и Оркестрация:
    - Docker для контейнеризации ASP.NET-приложения и баз данных.

Снимки экрана приложения:

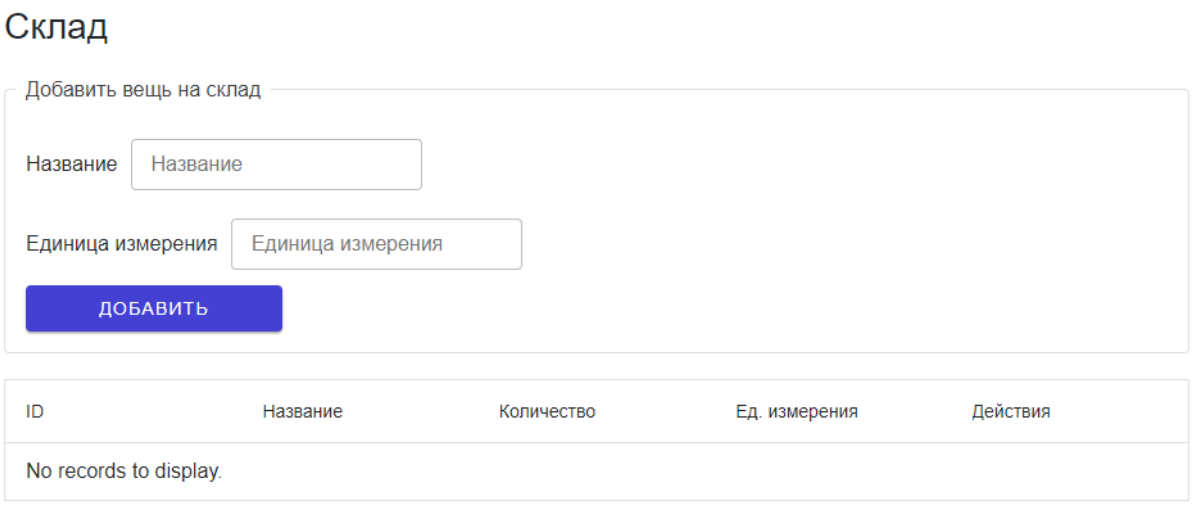

Рисунок 4 - Склад

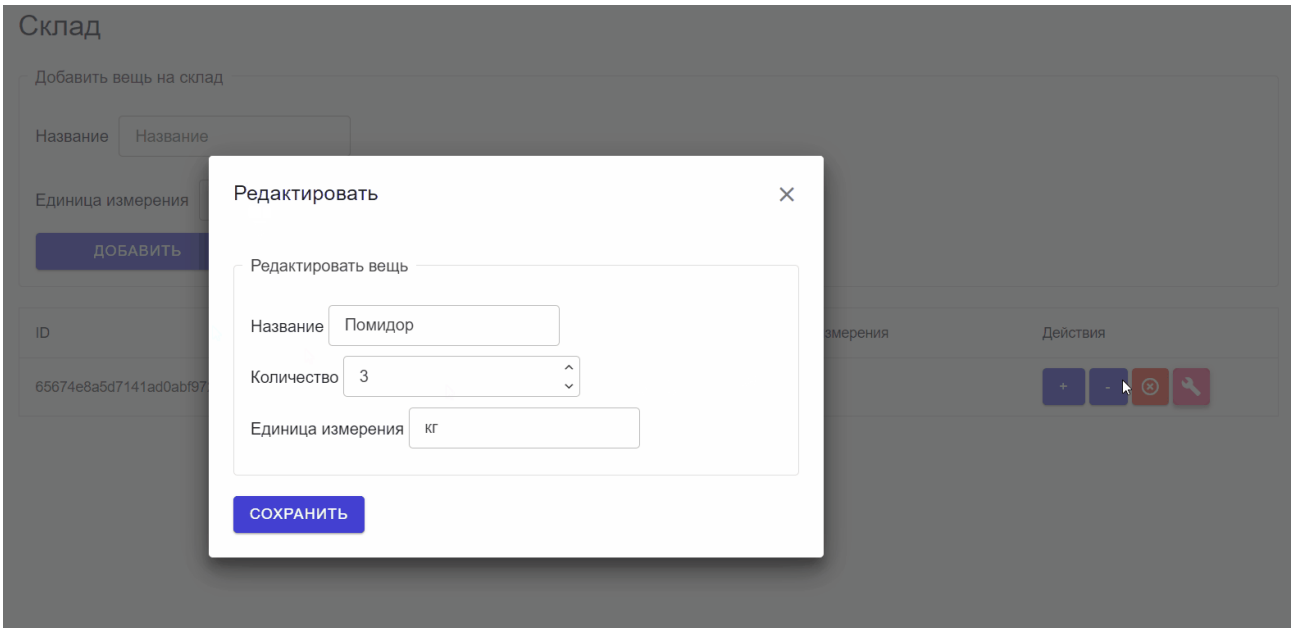

Рисунок 5 - Редактирование вещи на складе

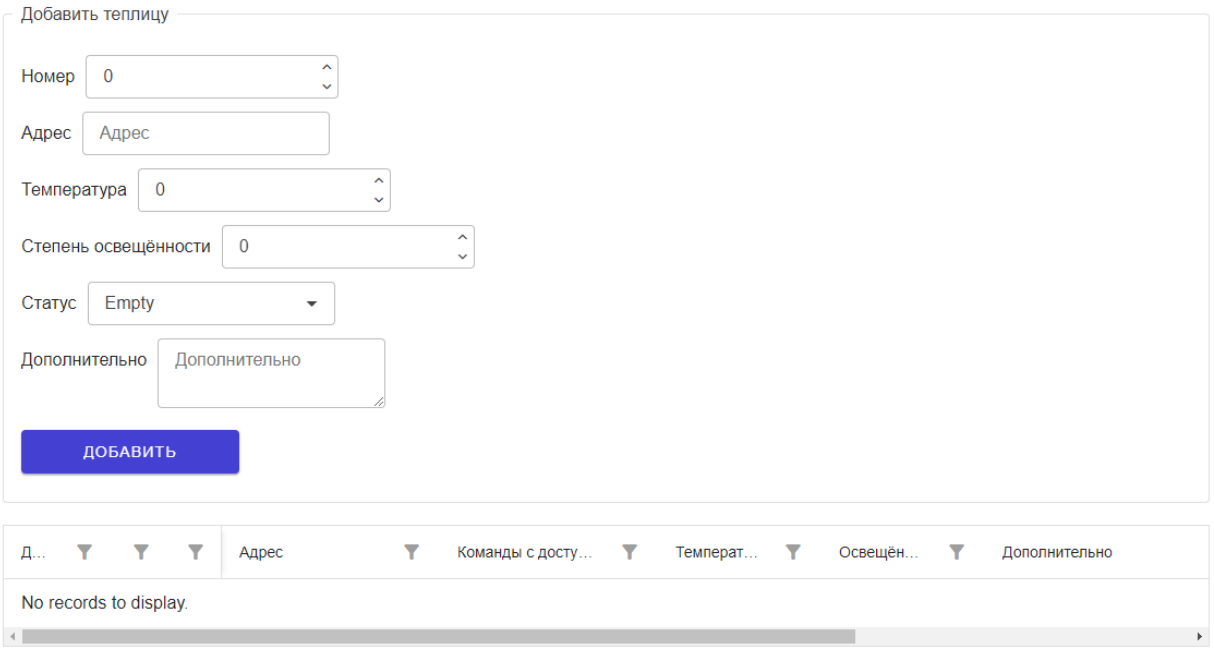

Рисунок 6 - Теплица

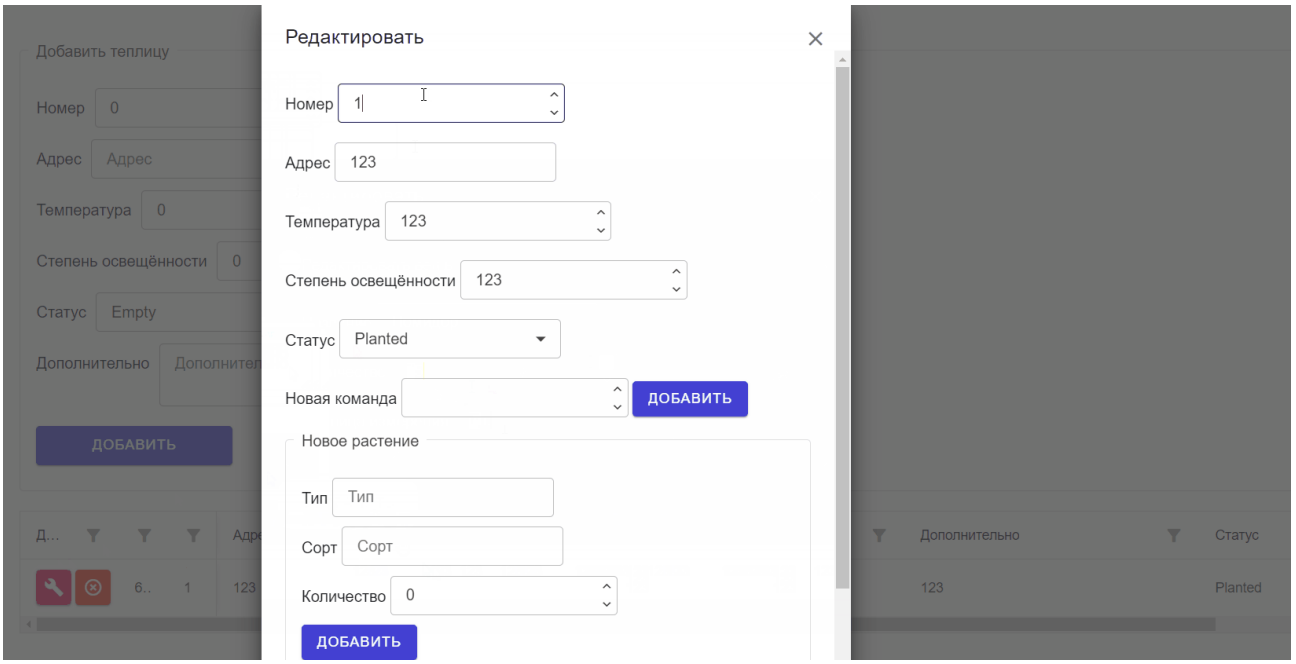

Рисунок 7 - Редактирование теплицы

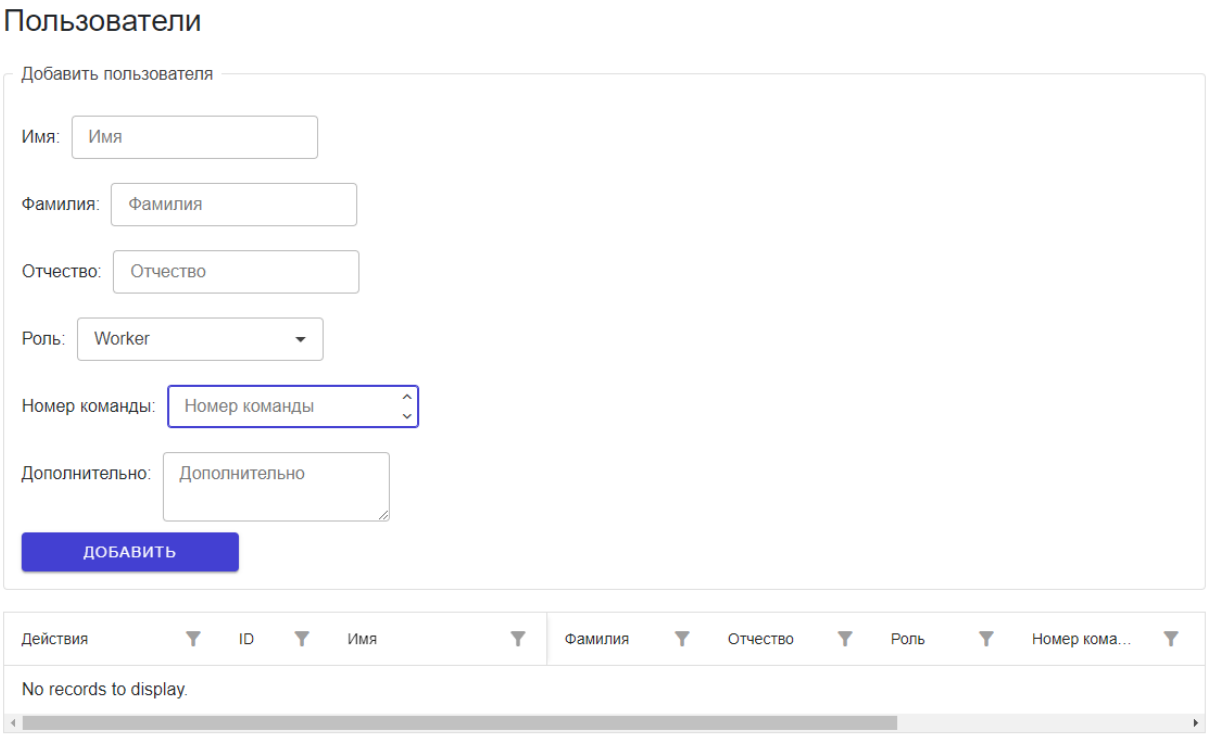

Рисунок 8 - Пользователи

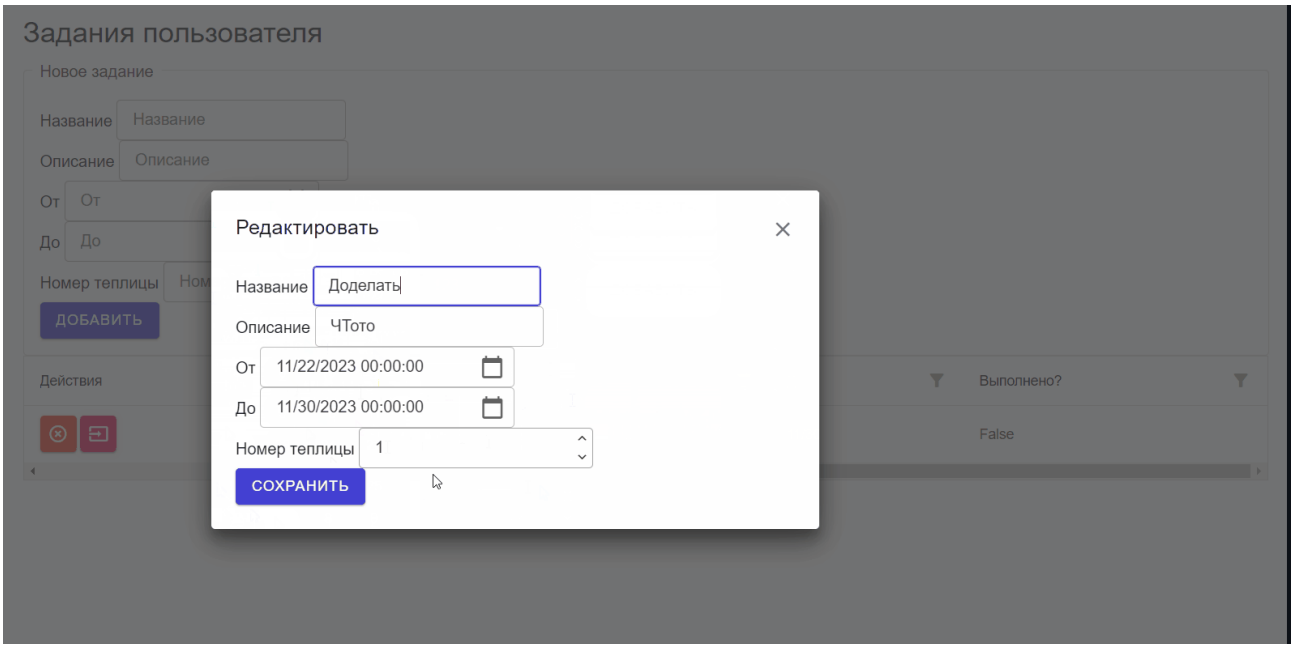

Рисунок 9 - Редактирование задания пользователя

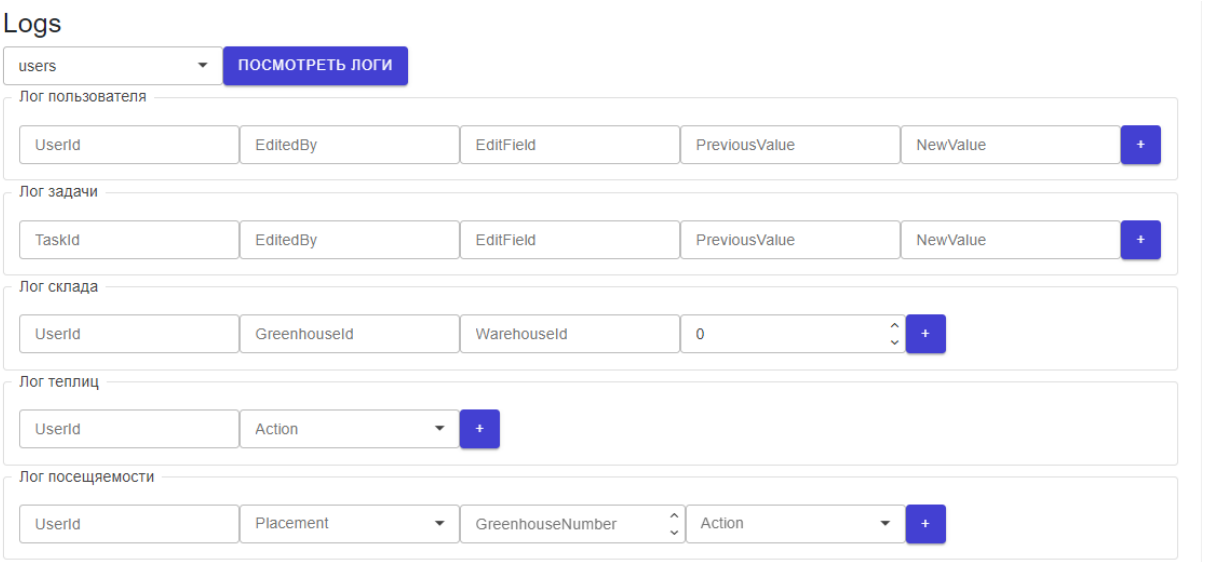

Рисунок 10 - Логи пользователя

ВЫВОДЫ.

Достигнутые результаты:

1. Функциональность:
    - Разработано приложение для автоматизированного мониторинга тепличной фермы овощей.
    - Реализованы модули для управления нарядами, учета склада, фиксации инцидентов, графика дежурств и мониторинга параметров фермы.
2. Технологии:
    - Использованы современные технологии, такие как ASP.NET Core, MongoFramework, MongoDB, InfluxDB и Docker.
    - Применение Blazor и Radzen для создания динамических представлений.
3. Простота в поддержке:
    - Использование монолитной архитектуры обеспечивает легкое понимание происходящего, так как вся логика сосредоточена в файле с страницей и тесно связана с отображением.

Недостатки и пути для улучшения:

1. Интерфейс:
    - Может потребоваться дополнительная работа над дизайном интерфейса для улучшения пользовательского опыта.
2. Оптимизация запросов:
    - В зависимости от объема данных и нагрузки, возможно, потребуется оптимизация запросов к базам данных для обеспечения высокой производительности.
3. Дополнительные функциональности:
    - Добавление дополнительных функций, таких как уведомления, аналитика и прогнозирование на основе данных.

Будущее развитие решения:

1. Внедрение сенсоров и системы IoT для сбора более точных данных о параметрах фермы.
2. Использование машинного обучения для анализа данных и предсказания урожайности, оптимизации производственных процессов.
3. Разработка мобильного приложения для удобного доступа к системе непосредственно на ферме.
4. Добавление новых модулей и функциональностей в ответ на потребности пользователей и изменения в бизнес-процессах.
5. Проведение систематического тестирования и оптимизации для обеспечения стабильной работы и эффективного использования ресурсов.
6. Регулярное обновление мер безопасности и внедрение средств защиты от угроз.

Развитие решения должно быть направлено на удовлетворение растущих потребностей бизнеса и использование новых технологий для повышения эффективности и конкурентоспособности фермы.

ПРИЛОЖЕНИЯ

Для запуска контейнеров: переходим в “nosql2h23-smart-farm/SmartFarm.Web” и пишем команду: “docker-compose up --build”, после чего переходим на localhost:8080

Для того, чтобы применились изменения, особенно те, что в Blazor, необходимо пересобрать проект и перезапустить контейнер сервера.

Второй вариант для разработчиков: в appsettings.json меняем ConnectionString на mongodb://localhost:27017/TestDb"

Запускаем контейнер mongo: docker run -d -p 27017:27017 --name mongo mongo:latest

Запускаем через Visual studio сервер.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://github.com/moevm/nosql2h23-smart-farm репозиторий приложения
2. https://blazor.radzen.com/ Radzen Blazor Components
3. https://dotnet.microsoft.com/en-us/apps/aspnet ASP.NET Core
4. https://github.com/TurnerSoftware/MongoFramework Mongo