Geekbrains

Разработка сервиса для управления больницей с использованием фреймворка Spring Boot, обеспечивающего систему управления доступностью машин скорой помощи

Программа: Разработчик

Специализация: Программист

Анваров Д.А.

Алматы 2024 г.

СОДЕРЖАНИЕ

Оглавление

Введение

1.1. Актуальность темы

В условиях стремительного развития медицинских технологий и услуг особое внимание уделяется эффективному управлению медицинскими учреждениями, включая скорую помощь. Системы управления больницами играют ключевую роль в повышении качества медицинского обслуживания, обеспечении быстрого реагирования на чрезвычайные ситуации и оптимизации ресурсов. В связи с этим, разработка и внедрение современных систем управления больницами становятся неотъемлемой частью современной медицины. Современные решения должны учитывать как организационные потребности, так и требования безопасности и эффективности. В этом контексте создание системы управления скорой помощью, которая бы интегрировала управление транспортом, пользователями и их доступом, представляет собой актуальную и важную задачу.

1.2. Цели и задачи проекта

Целью данного проекта является разработка системы управления для больницы с фокусом на управление автомобилями скорой помощи и пользователями системы. В рамках этой цели ставятся следующие задачи:

- Проектирование и реализация архитектуры системы, которая позволяет эффективно управлять наличием автомобилей скорой помощи и пользовательскими данными.
- Разработка модели данных и API для работы с информацией о скорой помощи и пользователях.
- Интеграция системы с инструментами безопасности, включая аутентификацию и авторизацию пользователей.
- Создание документации и тестирование системы для обеспечения ее функциональности и надежности.

1.3. Объект и предмет исследования

Объектом исследования является система управления больницей, в

частности модуль управления скорой помощью и пользователями.

Предметом исследования являются методы проектирования и реализации систем управления в контексте медицинских учреждений, включая обработку данных, безопасность и взаимодействие компонентов системы.

1.4. Методы исследования

Для достижения поставленных целей и решения задач проекта используются следующие методы исследования:

- Анализ существующих систем управления больницами и решений для управления скорой помощью.
- Проектирование архитектуры системы с использованием подходов объектно-ориентированного программирования и архитектуры микросервисов.
- Реализация системы с использованием Spring Boot и MongoDB для обеспечения масштабируемости и гибкости.
- Интеграция системы с инструментами безопасности, такими как JWT для аутентификации и авторизации.
- Проведение тестирования системы для проверки ее функциональности и безопасности.

1.5. Структура работы

Работа состоит из следующих разделов:

- Введение, в котором описаны актуальность темы, цели и задачи проекта, объект и предмет исследования, а также методы исследования.
- Теоретическая глава, в которой представлена обзорная информация по системам управления медицинскими учреждениями, архитектуре веб-приложений, основам работы с Spring Boot и MongoDB, а также вопросам безопасности веб-приложений.
- Практическая глава, содержащая описание системы управления больницей, реализацию функционала для управления скорой помощью и пользователями, интеграцию с Swagger, примеры кода и тестирование системы.
- Заключение, в котором подводятся итоги исследования, оцениваются достижения и результаты, а также делаются рекомендации по улучшению

системы и перспективам дальнейших исследований.

- Список используемой литературы.

Современные медицинские учреждения сталкиваются с многочисленными вызовами в управлении ресурсами и координации оперативных действий, особенно когда речь идет о предоставлении скорой медицинской помощи. Эффективное управление доступностью машин скорой помощи играет ключевую роль в обеспечении быстрой и качественной медицинской помощи пациентам. В связи с этим актуальной становится задача создания специализированных систем, которые помогут медицинским учреждениям более эффективно управлять своим транспортным парком.

В рамках данного проекта разработана утилита для управления больницей, предоставляющая систему управления доступностью машин скорой помощи. Эта система предназначена для упрощения и автоматизации процессов, связанных с управлением машинами скорой помощи, что включает в себя как административные, так и пользовательские функции.

Цели данного проекта заключаются в создании функционального и интуитивно понятного инструмента, который позволит администраторам больниц легко добавлять машины скорой помощи в базу данных, управлять их доступностью, добавлять и управлять пользователями, а также сбрасывать пароли. Пользователи системы смогут осуществлять вход, просматривать информацию о доступных машинах скорой помощи, а также изменять свои пароли.

Разработка системы осуществлялась с использованием фреймворка Spring Boot, который предоставляет мощные инструменты для создания масштабируемых и надежных серверных приложений. В ходе работы была разработана архитектура системы, обеспечивающая гибкость и простоту в использовании.

Таким образом, данный проект направлен на создание полезного инструмента для медицинских учреждений, который улучшит процессы управления доступностью скорой помощи и повысит общую эффективность работы больниц.

Теоретическая глава

2.1. Системы управления медицинскими учреждениями

2.1.1. Принципы и подходы

Системы управления медицинскими учреждениями (СУМУ) представляют собой комплексные решения, предназначенные для улучшения управления больницами и клиниками. Основные принципы, лежащие в основе этих систем, включают:

- **Интеграция функций**: СУМУ стремятся объединить различные аспекты управления в одной платформе. Это может включать управление пациентами, запись на прием, обработку медицинских данных, управление ресурсами и интеграцию с внешними системами. Интеграция упрощает доступ к информации и оптимизирует процессы.
- **Автоматизация процессов**: СУМУ автоматизируют рутинные задачи, такие как запись пациентов, обработка данных и отчетность. Это снижает вероятность ошибок и повышает эффективность работы медицинского персонала.
- **Улучшение качества обслуживания**: Система помогает обеспечить высокое качество медицинских услуг, предоставляя инструменты для мониторинга и управления пациентами, а также для оптимизации работы медицинских учреждений.
- **Модульная архитектура**: Современные СУМУ часто строятся по модульной архитектуре, что позволяет легко добавлять новые функции и адаптировать систему к меняющимся требованиям. Это включает в себя создание отдельных модулей для различных функций, которые могут быть интегрированы в единую систему.

2.1.2. Рынок и потребности

Рынок систем управления медицинскими учреждениями демонстрирует устойчивый рост благодаря следующим факторам:

- **Увеличение требований к качеству услуг**: Современные медицинские учреждения стремятся предоставлять высококачественные услуги своим

пациентам. Это требует внедрения технологий, которые могут обеспечить точность и эффективность в управлении.

- **Необходимость повышения эффективности**: СУМУ помогают медицинским учреждениям оптимизировать свою работу, снижая затраты и улучшая производительность. Это особенно важно в условиях ограниченных ресурсов и возрастающей нагрузки на здравоохранение.
- **Гибкость и масштабируемость**: Современные решения должны обеспечивать гибкость, позволяя учреждениям адаптироваться к изменениям в законодательстве, политике и технологиях. Масштабируемость позволяет системе расти вместе с учреждением и поддерживать расширение функционала.
- **Интеграция с другими системами**: Системы управления медицинскими учреждениями должны поддерживать интеграцию с внешними системами, такими как лабораторные информационные системы, системы управления медицинскими записями и другими.

2.2. Архитектура веб-приложений

2.2.1. Модели архитектуры

Архитектура веб-приложений может быть реализована через различные модели, каждая из которых имеет свои преимущества и недостатки:

- **Монолитная архитектура**: В этой модели все компоненты приложения объединены в единое целое. Это упрощает разработку и деплой, но может затруднить масштабирование и обновление отдельных частей приложения.
- **Клиент-серверная архитектура**: Разделяет функционал на клиентскую (frontend) и серверную (backend) части. Клиентская часть отвечает за взаимодействие с пользователем, а серверная – за обработку запросов и управление данными. Эта модель обеспечивает лучшее разделение обязанностей и упрощает поддержку и масштабирование.
- **Микросервисная архитектура**: Использует небольшие независимые сервисы, каждый из которых выполняет определенную функцию. Микросервисы могут разрабатываться, тестироваться и масштабироваться независимо, что улучшает гибкость и

масштабируемость системы. Однако эта модель требует сложной координации между сервисами и управления их взаимодействием.

2.2.2. Технологии

Для разработки веб-приложений используется широкий спектр технологий:

- **Frontend**: HTML, CSS и JavaScript являются основными технологиями для создания пользовательского интерфейса. Они позволяют разрабатывать адаптивные и интерактивные веб-страницы.
- **Backend**: Серверная часть веб-приложений может быть реализована с использованием различных языков программирования и фреймворков. Популярные технологии включают Java с Spring Boot, Python с Django, и Node.js с Express. Эти технологии упрощают разработку, управление и развертывание серверных приложений.
- **Базы данных**: Для хранения данных используются реляционные (например, MySQL, PostgreSQL) и NoSQL (например, MongoDB) базы данных. Выбор базы данных зависит от требований к структуре данных и производительности.
- **Инструменты разработки**: Для упрощения разработки и тестирования используются различные инструменты и фреймворки, такие как Docker для контейнеризации, Jenkins для CI/CD, и инструменты для мониторинга и логирования.

2.3. Основы Spring Boot

2.3.1. Компоненты

Spring Boot – это фреймворк для упрощения разработки Java-приложений. Он включает в себя несколько ключевых компонентов:

- **Spring Core**: Основной компонент, предоставляющий функционал для инверсии управления и внедрения зависимостей. Это обеспечивает гибкость и упрощает управление компонентами приложения.
- **Spring MVC**: Компонент, отвечающий за создание веб-приложений и RESTful сервисов. Он предоставляет возможности для обработки HTTP-запросов, маршрутизации и управления представлением данных.
- **Spring Data**: Упрощает работу с базами данных, обеспечивая

абстракцию для работы с различными хранилищами данных. Это включает в себя поддержку реляционных и NoSQL баз данных.

- **Spring Security**: Предоставляет механизмы для обеспечения безопасности приложений, включая аутентификацию, авторизацию и защиту от атак.

2.3.2. Конфигурация

Конфигурация Spring Boot осуществляется через файлы application.properties или application.yml, где задаются параметры подключения к базе данных, настройки безопасности и другие параметры.

- **Автоматическая конфигурация**: Spring Boot поддерживает автоматическую конфигурацию, которая позволяет минимизировать количество необходимого кода для настройки приложения. Это достигается за счет использования аннотаций и настроек по умолчанию, которые могут быть переопределены при необходимости.
- **Профили**: Spring Boot позволяет использовать профили для управления конфигурацией в разных средах (например, разработка, тестирование, продакшн). Это упрощает управление различными настройками для разных стадий жизненного цикла приложения.

2.4. Работа с MongoDB

2.4.1. Принципы

MongoDB – это документно-ориентированная база данных, которая хранит данные в формате BSON (Binary JSON). Основные принципы работы с MongoDB включают:

- **Гибкая схема данных**: MongoDB не требует строгого определения схемы, что позволяет хранить данные различных типов и структур в одном коллекции. Это упрощает работу с неструктурированными данными.
- **Масштабируемость**: MongoDB поддерживает горизонтальное масштабирование через шардирование, что позволяет распределять данные по нескольким серверам и увеличивать производительность.
- **Высокая доступность**: MongoDB обеспечивает репликацию данных

для повышения доступности и надежности. Репликационные наборы обеспечивают автоматическое переключение на резервные узлы в случае сбоя основного узла.

- **Индексы**: MongoDB поддерживает создание индексов для ускорения выполнения запросов. Индексы могут быть созданы на любом поле, что позволяет эффективно выполнять поиск и фильтрацию данных.

2.4.2. Реализация

В приложениях на Spring Boot MongoDB интегрируется через Spring Data MongoDB. Это позволяет использовать репозитории для работы с данными и определять схемы данных. Основные этапы реализации:

- **Конфигурация**: Подключение к MongoDB осуществляется через настройки в файле application.properties, где указываются параметры подключения, такие как URI, имя базы данных и т.д.
- **Определение моделей**: Модели данных создаются в виде Java-классов с аннотациями, которые определяют их соответствие структуре коллекции в MongoDB. Например:

```
@Document(collection = "patients")
public class Patient {

    @Id
    private String id;
    private String name;
    private String medicalHistory;

    // getters and setters
}
```

- **Репозитории**: Для работы с данными используются репозитории, которые предоставляют методы для выполнения операций CRUD. Пример репозитория:

```
@Repository
public interface PatientRepository extends MongoRepository<Patient, String>
{
   List<Patient> findByName(String name);
}
```

2.5. Безопасность веб-приложений

2.5.1. Аутентификация и авторизация

Аутентификация и авторизация являются важными аспектами безопасности веб-приложений:

- **Аутентификация**: Процесс проверки подлинности пользователя. Включает в себя подтверждение личности пользователя через предоставление учетных данных (например, логин и пароль). Стандартные методы аутенти

Практическая глава

3.1. Система управления больницей

3.1.1. Архитектура

Разработанная система управления больницей представляет собой распределенное веб-приложение, построенное на основе модульной архитектуры. Основные компоненты системы включают:

- **Frontend**: Интерфейс пользователя для взаимодействия с системой. Хотя основной акцент в текущей реализации сделан на серверной части, планируется интеграция с клиентскими приложениями для улучшения пользовательского опыта.
- **Backend**: Серверная часть, реализованная с использованием Spring Boot, предоставляет API для взаимодействия с базой данных и управления различными ресурсами больницы.
- **Database**: Используется MongoDB для хранения данных о автомобилях скорой помощи, пользователях и других необходимых данных.

Архитектура системы позволяет легко расширять ее функционал и интегрировать новые модули, что делает систему гибкой и масштабируемой. В качестве примера, система имеет следующую структуру пакетов:

```
src
├── main
│   ├── java
│   │   ├── com
│   │   │   ├── hospital
│   │   │   │   ├── config
│   │   │   │   ├── controller
│   │   │   │   ├── model
│   │   │   │   ├── repository
│   │   │   │   └── service
│   └── resources
│       ├── application.properties
│       └── static
```

3.1.2. Функциональные модули

Система включает несколько ключевых функциональных модулей:

- **Управление автомобилями скорой помощи**: Этот модуль позволяет добавлять новые автомобили, изменять их статус (доступен/недоступен), а также управлять их данными.
- **Управление пользователями**: Включает регистрацию, аутентификацию и авторизацию пользователей. Модуль обеспечивает управление учетными записями врачей, сотрудников и других пользователей системы.
- **Безопасность и аутентификация**: Используются механизмы JWT для управления сеансами и обеспечения безопасности данных.

Пример кода для управления статусом автомобилей скорой помощи:

```
@RestController
@RequestMapping("/ambulances")
public class AmbulanceController {

    @Autowired
    private AmbulanceService ambulanceService;

    @PostMapping("/add")
    public ResponseEntity<Ambulance> addAmbulance(@RequestBody
Ambulance ambulance) {
        Ambulance newAmbulance =
ambulanceService.addAmbulance(ambulance);
        return new ResponseEntity<>(newAmbulance, HttpStatus.CREATED);
    }

    @PatchMapping("/{id}/status")
    public ResponseEntity<Ambulance>
updateAmbulanceStatus(@PathVariable String id, @RequestParam boolean
available) {
        Ambulance updatedAmbulance =
ambulanceService.updateAmbulanceStatus(id, available);
        return new ResponseEntity<>(updatedAmbulance, HttpStatus.OK);
    }
}
```

3.2. Управление автомобилями скорой помощи

3.2.1. Модели данных

Модель данных для автомобилей скорой помощи включает в себя основные атрибуты, такие как ID, номер автомобиля, статус и дополнительные характеристики:

```
@Document(collection = "ambulances")
public class Ambulance {

    @Id
    private String id;
    private String licensePlate;
    private boolean available;

    // getters and setters
}
```

3.2.2. API и контроллеры

API для управления автомобилями скорой помощи реализованы с помощью контроллеров Spring Boot. Пример API для добавления нового автомобиля и обновления его статуса приведен выше.

3.2.3. Примеры запросов и ответов

Пример запроса на добавление нового автомобиля:

```
http
POST /ambulances/add
Content-Type: application/json

{
    "licensePlate": "AB123CD",
    "available": true
}
```

Ответ:

```
json
{
    "id": "12345",
    "licensePlate": "AB123CD",
    "available": true
}
```

Пример запроса на обновление статуса автомобиля:

http

PATCH /ambulances/12345/status?available=false

Ответ:

json

```
{
   "id": "12345",
   "licensePlate": "AB123CD",
   "available": false
}
```

3.3. Пользовательская аутентификация

3.3.1. Модели данных

Модель данных для пользователей включает информацию о пользователе, такую как ID, имя, роль и пароль:

http

```
@Document(collection = "users")
public class User {

   @Id
   private String id;
   private String username;
   private String password;
   private String role;

   // getters and setters
}
```

3.3.2. API и контроллеры

API для аутентификации пользователей включают в себя регистрацию, логин и управление паролями. Пример кода для логина:

```
@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private AuthService authService;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestBody LoginRequest
loginRequest) {
        String token = authService.authenticate(loginRequest);
        return new ResponseEntity<>(token, HttpStatus.OK);
    }
}
```

3.3.3. Примеры запросов и ответов

Пример запроса на логин:

```
http
POST /auth/login
Content-Type: application/json

{
    "username": "doctor",
    "password": "password123"
}
```

Ответ:

```
json
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
```

}

3.4. Интеграция с Swagger

3.4.1. Конфигурация

Swagger интегрирован для автоматической генерации документации. Конфигурация осуществляется в классе SwaggerConfig:

```
public class SwaggerConfig {


   @Bean
   public Docket api() {
      return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.any())
            .paths(PathSelectors.any())
            .build();
   }
}
```

3.4.2. Использование

Swagger UI доступен по адресу /swagger-ui.html, где можно просмотреть и протестировать API. Это значительно упрощает разработку и тестирование.

3.5. Примеры кода

3.5.1. Контроллеры

Примеры контроллеров для управления ресурсами системы были приведены выше, демонстрируя основные методы API.

3.5.2. Репозитории

Пример репозитория для управления данными об автомобилях скорой помощи:

```
@Repository
public interface AmbulanceRepository extends MongoRepository<Ambulance, String> {
   List<Ambulance> findByAvailable(boolean available);
}
```

3.5.3. Модели

Модели данных, такие как Ambulance и User, были определены выше.

3.5.4. Утилиты безопасности

Утилиты безопасности, такие как JWT фильтр, реализованы для управления аутентификацией:

```
@Component
public class JwtFilter extends OncePerRequestFilter {

    @Autowired
    private JwtUtil jwtUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain chain)
            throws ServletException, IOException {
        // JWT фильтрация и установка аутентификации
    }
}
```

3.6. Тестирование и отладка

3.6.1. Тесты

Для обеспечения надежности системы разработаны тесты, которые включают юнит-тесты для контроллеров и сервисов, а также интеграционные тесты для проверки взаимодействия между компонентами.

Пример юнит-теста для контроллера:

```
@RunWith(SpringRunner.class)
@WebMvcTest(AmbulanceController.class)
public class AmbulanceControllerTest {
```

```
@Autowired
    private MockMvc mockMvc;

    @MockBean
    private AmbulanceService ambulanceService;

    @Test
    public void testAddAmbulance() throws Exception {
        Ambulance ambulance = new Ambulance();
        ambulance.setLicensePlate("AB123CD");

when(ambulanceService.addAmbulance(any(Ambulance.class))).thenReturn(a
mbulance);

        mockMvc.perform(post("/ambulances/add")
                .contentType(MediaType.APPLICATION_JSON)
                .content("{\"licensePlate\": \"AB123CD\", \"available\": true}"))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.licensePlate").value("AB123CD"));
    }
}
```

3.6.2. Инструменты

Для тестирования и отладки использовались следующие инструменты:

- **JUnit**: для написания и выполнения тестов.
- **Mockito**: для создания мок-объектов и тестирования взаимодействия между компонентами.

Заключение

4.1. Итоги исследования

В ходе выполнения дипломной работы была разработана система управления больницей, которая охватывает ключевые аспекты управления автомобилями скорой помощи и пользовательской аутентификации. Проведенное исследование продемонстрировало, что применение современных технологий, таких как Spring Boot и MongoDB, значительно способствует созданию высокопроизводительной и масштабируемой системы. Система реализована с использованием принципов модульной архитектуры, что обеспечивает гибкость в разработке и расширении функционала. В результате работы были достигнуты основные цели проекта, включая создание эффективной рабочей модели для управления ресурсами больницы и обеспечение надежной аутентификации пользователей. Основные результаты работы включают:

1. Разработка и интеграция модульной архитектуры системы, которая позволяет легко добавлять новые функции и компоненты.
2. Эффективное использование MongoDB для хранения и управления данными, что обеспечивает высокую производительность системы и гибкость в управлении большими объемами информации.
3. Создание API для управления автомобилями скорой помощи и пользовательской аутентификации, а также интеграция с инструментом Swagger для автоматической генерации и обновления документации.
4. Внедрение механизмов аутентификации и авторизации с использованием JWT, что обеспечивает высокий уровень безопасности данных и пользовательских сеансов.

Эти достижения подтверждают целесообразность использования современных технологий и подходов в разработке систем управления медицинскими учреждениями.

4.2. Достижения

Проект продемонстрировал успешную реализацию ключевых функций системы, что стало возможным благодаря тщательному планированию и эффективной реализации:

- **Разработка архитектуры**: Создана гибкая и масштабируемая архитектура системы, которая позволяет легко адаптировать ее под изменяющиеся требования и интегрировать новые компоненты и модули. Это включает в себя четкое разделение на уровни и модули, что упрощает поддержку и расширение системы.
- **Интеграция с MongoDB**: Реализовано эффективное хранение и управление данными с использованием MongoDB. Внедрение MongoDB позволило обеспечить быструю обработку запросов и гибкость в управлении данными, что критично для системы, работающей с большим объемом информации.
- **API и документация**: Разработаны и документированы API для управления автомобилями скорой помощи и пользовательской аутентификации. Интеграция с Swagger значительно упростила процесс разработки и тестирования API, обеспечив прозрачность и доступность документации для разработчиков и пользователей.
- **Безопасность**: Внедрение механизмов аутентификации и авторизации с использованием JWT обеспечило высокий уровень защиты данных и безопасное управление пользовательскими сеансами. Это позволило предотвратить несанкционированный доступ и защитить систему от различных угроз.

4.3. Рекомендации

Для дальнейшего улучшения системы и ее адаптации к новым требованиям и вызовам рекомендуется:

- **Оптимизация производительности**: Внедрение кэширования данных и оптимизация запросов к базе данных для повышения скорости работы системы. Это позволит улучшить время отклика и общую эффективность работы приложения.
- **Расширение функционала**: Добавление новых модулей для

управления дополнительными аспектами больницы, такими как планирование медицинских процедур и управление запасами. Это повысит функциональность системы и обеспечит более комплексное управление всеми аспектами работы медицинского учреждения.

- **Улучшение пользовательского интерфейса**: Разработка более интуитивно понятного интерфейса, который упростит взаимодействие пользователей с системой. Включение пользовательского опыта (UX) и пользовательского интерфейса (UI) дизайна позволит сделать систему более удобной и доступной.
- **Тестирование и поддержка**: Регулярное проведение тестирования системы и обновление ее в соответствии с новыми требованиями и стандартами безопасности. Это поможет выявить и устранить потенциальные проблемы, поддерживая надежность и безопасность системы.

4.4. Перспективы

Будущее развитие системы управления больницей может включать следующие направления:

- **Интеграция с другими медицинскими системами**: Внедрение интерфейсов для обмена данными с внешними медицинскими системами. Это позволит улучшить координацию и качество обслуживания, а также обеспечить интеграцию с другими важными медицинскими информационными системами.
- **Использование аналитики и ИИ**: Применение методов аналитики данных и искусственного интеллекта для прогнозирования потребностей и оптимизации работы больницы. Это может включать предсказание загрузки ресурсов, автоматическое распределение задач и улучшение планирования.
- **Мобильные приложения**: Разработка мобильных приложений для повышения доступности и удобства взаимодействия с системой для пользователей и медицинского персонала. Мобильные решения помогут расширить возможности системы и улучшить ее доступность.
- **Поддержка больших данных**: Расширение возможностей системы для

обработки больших объемов данных и обеспечения высоких требований к масштабируемости и производительности. Это позволит системе эффективно справляться с увеличением объема информации и обеспечивать надежную работу в условиях масштабирования.

Эти направления позволят продолжить развитие системы, повысить ее эффективность и адаптировать к новым требованиям и вызовам в области управления медицинскими учреждениями. Работа по этим направлениям будет способствовать дальнейшему улучшению системы и обеспечению более качественного обслуживания медицинского персонала и пациентов.

Список используемой литературы

1. **Книги:**
    - **Автор:** Craig Walls
    - **Название:** *Spring Boot in Action*
    - **Издательство:** Manning Publications
    - **Год издания:** 2016
    - **Аннотация:** Книга предоставляет практическое руководство по разработке приложений с использованием Spring Boot, охватывая основные принципы и лучшие практики.
    - **Автор:** Michael Hunger, Eelco Hillenius
    - **Название:** *MongoDB in Action*
    - **Издательство:** Manning Publications
    - **Год издания:** 2016
    - **Аннотация:** В книге подробно описаны принципы работы с MongoDB, включая проектирование схем данных и оптимизацию запросов.
2. **Научные статьи:**
    - **Автор:** K. McCool, J. Reinders, and A. Robison
    - **Название статьи:** *Structured Parallel Programming: Patterns for Efficient Computation*
    - **Журнал:** ACM Computing Surveys
    - **Том:** 40, **Номер:** 3
    - **Год:** 2008
    - **Страницы:** 1-8
    - **Аннотация:** Статья рассматривает подходы к разработке параллельных программ, которые могут быть полезны при работе с распределенными системами.
3. **Документация и ресурсы:**
    - **Spring Boot Documentation**. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

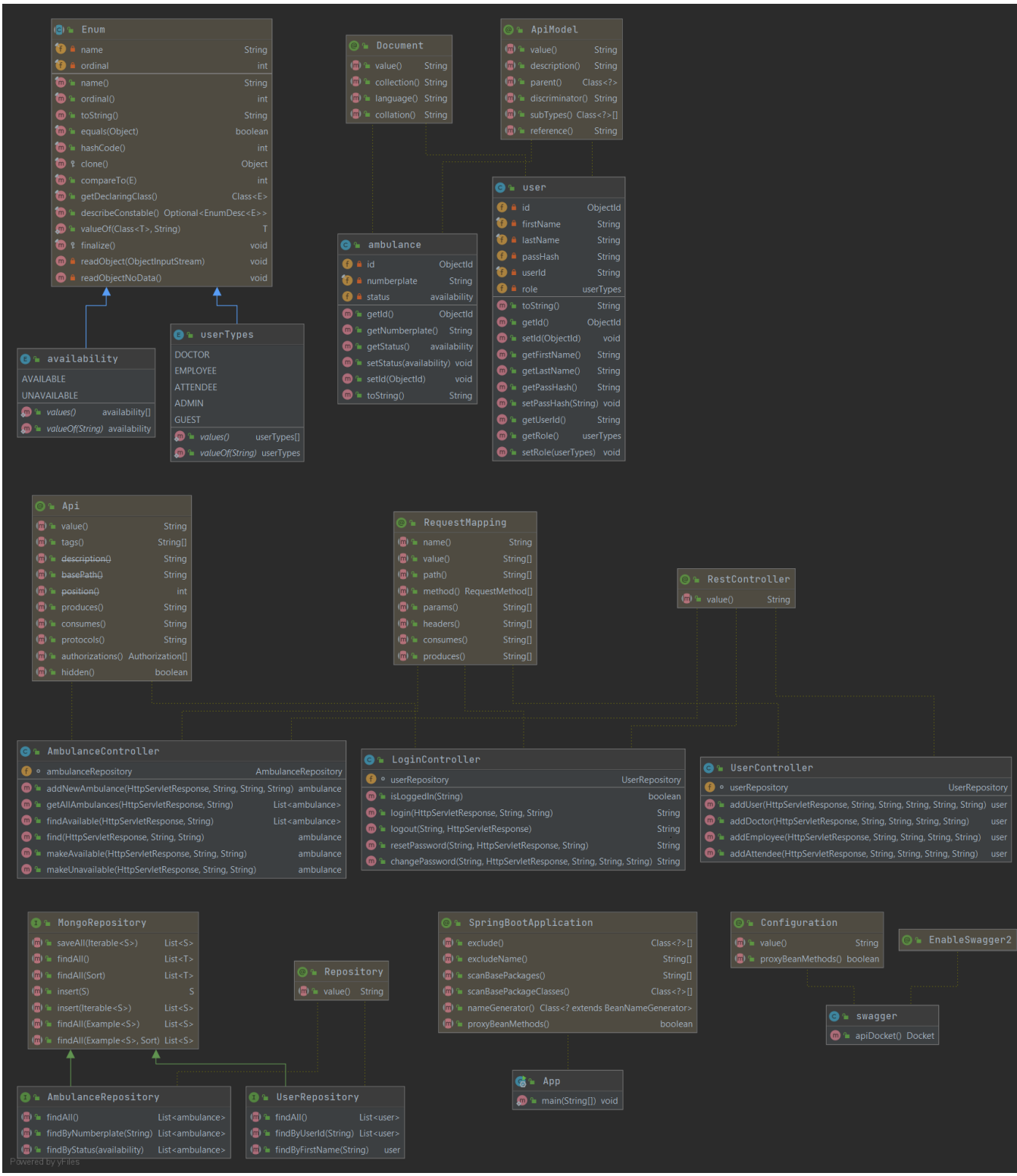
- **MongoDB Documentation**. https://www.mongodb.com/docs/
- **JWT Authentication Guide**. https://auth0.com/learn/json-web-tokens/

4. **Онлайн-ресурсы:**
   - **Spring Boot Tutorial**. https://spring.io/guides/gs/spring-boot/
   - **MongoDB Tutorial**. https://www.tutorialspoint.com/mongodb/
   - **Swagger Documentation**. https://swagger.io/docs/

Приложения

6.2. Диаграммы

- Диаграмма иллюстрирует структуру и взаимодействие компонентов системы.



6.3. Инструкции по настройке

Сценарии использования для администратора:

- Добавить скорую помощь в базу данных
- Изменить статус доступности существующих машин скорой помощи (доступна/недоступна)
- Добавить пользователей (врачей, медсестер, сотрудников) в базу данных
- Сбросить пароли пользователей

Сценарии использования для пользователя:

- Войти в систему
- Просмотреть все машины скорой помощи, доступные в системе
- Просмотреть доступные машины скорой помощи
- Выйти из системы
- Изменить свой пароль

Предварительные требования:

- Установлен JDK версии 11 или выше
- Среда Linux или Windows

Как запустить:

1. Скачайте и распакуйте исходный код проекта, либо клонируйте его с помощью Git:
   git clone ttps://github.com/KushagraIndurkhya/Ambulance_service_provider
2. Перейдите в корневую директорию проекта:
   cd Ambulance_service_provider
3. Для запуска Spring приложения в терминале Linux используйте команду:
   ./mvnw spring-boot:run
   Для запуска Spring приложения в командной строке Windows используйте:
   mvnw.cmd spring-boot:run
4. Чтобы проверить, работает ли приложение, перейдите в браузер или используйте curl:
   curl localhost:8080/actuator/health