Geekbrains

**Создание простого сайта и его тестирование**

Студент:

Программист мобильных устройств

Хозин Е. А.

Чистополь

2024

**Содержание**

|  |  |
| --- | --- |
| [Введение](#Введение) | 3 |
| 1. [Spring](#Spring) | 6 |
| 1.1. [Цели и задачи Spring](#Цели_Spring) | 6 |
| 1.2. [Инъекция зависимостей](#Инъекция_зависимостей) | 6 |
| 1.3. [Аспектно - ориентированное программирование](#АОП) | 6 |
| 1.4. [Функции Spring](#Функции_Spring) | 7 |
| 1.5. [Архитектура Spring](#Архитектура_Spring) | 7 |
| 1.6. [Контейнер компонентов](#Контейнер_компонентов) | 8 |
| 1.7. [Контейнеры](#Контейнеры) | 8 |
| 2. [Подготовительный этап разработки](#Подготовительный_этап_разработки) | 9 |
| 2.1. [Выбор среды разработки](#Выбор_среды_разработки) | 9 |
| 2.2. [Автоматическая сборка проекта](#Автоматическая_сборка_проекта) | 10 |
| 2.3. [Описание зависимостей](#Описание_зависимостей) | 10 |
| 3. [Разработка вэб – приложения](#Разработка_вэб_приложения) | 13 |
| 3.1. [Конфигурация Spring Security](#Конфигурация_Spring_Security) | 13 |
| 3.2. [Конфигурация подключения к базе данных](#Конфигурация_подключения_БД) | 14 |
| 3.3. [Основные сущности](#Основные_сущности) | 15 |
| 3.4. [Сервисы](#Сервисы) | 24 |
| 3.5. [Контроллеры](#Контроллеры) | 28 |
| 3.6. [Представления](#Представления) | 36 |
| 4. [База данных](#База_данных) | 48 |
| [Заключение](#Заключение) | 52 |
| [Список литературы и ресурсов](#Список_литературы) | 53 |

**Введение**

В последнее время становятся актуальными вэб - приложения, приходя на смену обычным приложениям для телефонов. Причин для этого может быть много: санкции, требования и комиссии «Магазинов» (Google Play, App Store, AppGallery и т.п.). Вэб – приложение же изначально является кросплаформенным решением и не зависит от операционной системы устройства. Оно работает, как вэб – сайт, поэтому при разработке необходимо учитывать, чтобы приложение одинаково работало с разными браузерами (Google Chrome, Mozilla Firefox, Microsoft Edge, Tor Browser, Opera, Safari) и было адаптивным (могло подстраиваться к разным размерам экрана).

Цель проекта это создать вэб – приложение для записи на междугородний автобус. В качестве примера направления движения автобуса я буду использовать города Чистополь и Казань. Будут организованы 14 рейсов из Чистополя в Казань и столько же из Казани в Чистополь. Записаться можно будет на три даты: сегодня, завтра и послезавтра. За час до отправления автобуса запись будет закрываться. Помимо пользователя который будет пользоваться функционалом записи так же планирую добавить «водителей» и «диспетчеров», чтобы водитель мог видеть свои ближайшие маршруты, и список пассажиров, с указанием времени и места посадки, для каждого из них. Диспетчер же сможет контролировать выполнение водителями рейсов, посадку пассажиров , так же записывать самостоятельно пассажиров, которые по каким либо причинам не могут воспользоваться приложением и звонят записаться по телефону, и отменять записи.

Написание вэб – приложения будет происходить в среде разработки **IntelliJ IDEA** на языке программирования **Java**, используя фреймворк **Spring Framework**, базу данных **MySQL**. Фронтенд будет написан на «языке гипертекстовой разметки» **HTML**, «каскадных таблиц стилей» **CSS**, языке программирования **JavaScript**. Также в проекте будет использоваться шаблонизатор **Thymeleaf** и css фреймворк **Bootstrap**.

«**Java** — строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process; язык и основные реализующие его технологии распространяются по лицензии GPL. Права на торговую марку принадлежат корпорации Oracle.

Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре, для которой существует реализация виртуальной Java-машины. Дата официального выпуска — 23 мая 1995 года. Занимает высокие места в рейтингах популярности языков программирования (2-е место в рейтингах IEEE Spectrum (2020) и TIOBE (2021)).» © [Википедия](https://ru.wikipedia.org/wiki/Java)

«**Spring Framework** (или коротко Spring) — универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.

Spring обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java. Из-за широкой функциональности трудно определить наиболее значимые структурные элементы, из которых он состоит. Spring не всецело связан с платформой Java Enterprise, несмотря на свою масштабную интеграцию с ней, что является важной причиной его популярности.

Spring может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании.» © [Википедия](https://ru.wikipedia.org/wiki/Spring_Framework)

«**MySQL** — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.» © [Википедия](https://ru.wikipedia.org/wiki/MySQL)

В дипломной работе я буду выступать в качестве Full - stack разработчика.

1. **Spring**
   1. **Цели и задачи Spring**

* Любое нетривиальное приложение состоит из множества компонентов, каждый из которых несет ответственность за свою часть общей функциональности приложения, взаимодействуя с другими элементами.
* При запуске приложения необходимо выполнять интеграцию компонентов.
* Spring предлагает контейнер – контекст приложения Spring – который создаёт и управляет компонентами приложения.
* Компоненты (или beans) соединены вместе в контексте приложения Spring, чтобы создать полноценное приложение.
  1. **Инъекция зависимостей**
* Архитектурный шаблон, подразумевающий подход, при котором вместо того, чтобы компоненты создавали и поддерживали жизненный цикл других компонентов, от которых они зависят, приложение, зависящее от зависимостей, полагается на отдельный объект (контейнер) для создания и поддержки всех компонентов и внедрения их в нужные им компоненты.
* Реализуется за счёт: параметризированных конструкторов и публичных свойств.
  1. **Аспектно - ориентированное программирование**
* АОП – аспектно-ориентированное программирование – архитектурный принцип, способствующий разделению задач в программной системе. Системы состоят из компонентов, каждый из которых имеет высокое зацепление.
  1. **Функции Spring**
* Spring Framework предоставляет архитектурную структуру приложения.
* Бизнес – логика инкапсулируется в набор бизнес – объектов – сервисы.
* Сервисы выполняют операции, общие для всех пользовательских интерфейсов.
* Приложение содержит разный набор контроллеров и представлений для каждого пользовательского интерфейса, и они используют общие бизнес - объекты для выполнения критических операций.
* Контроллеры необходимы только для выполнения операций с конкретным интерфейсом пользователя:
* HTML Views
* JPS
* JSON
* и т.д.
  1. **Архитектура Spring**
* Spring Framework состоит из модулей, которые разделены на категории.
* Модули предоставляют все необходимое для разработки корпоративных приложений.
* Включаемые модули (Spring Initializer):
* Core
* Spring MVC
* JDBC
* и т.д.
  1. **Контейнер компонентов**
* В приложении на основе Spring Framework объекты приложения содержатся в контейнере Spring.
* Контейнер будет:
* создавать объекты
* связывать их вместе
* настраивать и управлять их жизненным циклом
  1. **Контейнеры**
* Spring Framework состоит из контейнеров, которые можно разделить на два типа:
* Фабрики компонентов (интерфейс org.springframework.beans.factory. BeanFactory) являются простейшими контейнерами, предоставляющими базовую поддержку для внедрения зависимостей.
* Контекст приложений (интерфейс org.springframework.context. ApplicationContext) основан на понятии фабрики компонентов, реализует прикладные службы фреймворка, такие как:
* возможность приема текстовых сообщений из файлов свойств
* возможность публиковать события приложений для заинтересованных слушателей

1. **Подготовительный этап разработки**
   1. **Выбор среды разработки**

«Интегрированная среда разработки, ИСP (англ. integrated development environment — IDE), также единая среда разработки, ЕСР — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).

Среда разработки включает в себя:

* текстовый редактор,
* Транслятор (компилятор и/или интерпретатор),
* средства автоматизации сборки

Иногда содержит также средства для интеграции с системами управления версиями и разнообразные инструменты для упрощения конструирования графического интерфейса пользователя. Многие современные среды разработки также включают браузер классов, инспектор объектов и диаграмму иерархии классов для использования при объектно-ориентированной разработке ПО. ИСР обычно предназначены для нескольких языков программирования — такие как IntelliJ IDEA, NetBeans, Eclipse, Qt Creator, Geany, Embarcadero RAD Studio, Code::Blocks, Xcode, Dev-C++ или Microsoft Visual Studio, но есть и IDE для одного определённого языка программирования — как, например, Visual Basic или Delphi.» © [Википедия](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)

Мой выбор пал на IntelliJ IDEA, потому что эта среда разработки мне понравилась больше, чем Visual Studio Code. А программировал на языке Java я только на этих двух IDE.

* 1. **Автоматическая сборка проекта**

В проекте будет использоваться система автоматической сборки (также известная как сборочный сервер) Gradle. И стоит начать с добавления в файл build.gradle зависимостей, благодаря которым сборщик скачает и установит в наш проект необходимые библиотеки:

dependencies {  
 implementation 'org.springframework.boot:spring-boot-starter-web'  
 developmentOnly 'org.springframework.boot:spring-boot-devtools'  
 testImplementation 'org.springframework.boot:spring-boot-starter-test'  
 implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
 runtimeOnly 'com.mysql:mysql-connector-j'  
 implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'  
   
 implementation group: 'org.webjars', name: 'bootstrap', version: '5.2.3'  
 implementation group: 'org.webjars', name: 'webjars-locator-core', version: '0.52'  
   
 implementation 'org.springframework.boot:spring-boot-starter-security'  
// Temporary explicit version to fix Thymeleaf bug  
 implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity6:3.1.1.RELEASE'  
 implementation 'org.springframework.security:spring-security-test'  
}

* 1. **Описание зависимостей**

**Spring Boot** (spring-boot-starter-web) это средство с открытым исходным кодом, которое упрощает использование платформ на основе Java для создания микрослужб и веб-приложений. Обеспечивает возможность создавать автономные приложения Spring, которые можно запускать сразу и которым не нужны заметки, XML-файл конфигурации или написание множества дополнительных строк кода. [©](https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what-is-java-spring-boot)

**Spring Security** (spring-boot-starter-security) это фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. [©](https://habr.com/ru/articles/203318/)

**Thymeleaf** (spring-boot-starter-thymeleaf) — современный серверный механизм Java-шаблонов для веб- и автономных сред, способный обрабатывать HTML, XML, JavaScript, CSS и простой текст. Основной целью Thymeleaf является создание элегантного и удобного способа шаблонизации. Чтобы достичь этого, Thymeleaf основывается на концепции Natural Templates, чтобы внедрить свою логику в файлы шаблонов таким образом, чтобы этот шаблон не влиял на отображение прототипа дизайна. Thymeleaf также был разработан с самого начала с учетом стандартов Web, особенно HTML5, что позволяет вам создавать полностью соответствующие стандарту шаблоны. [©](https://habr.com/ru/articles/350864/)

**Java Persistence API (**spring-boot-starter-data-jpa) – это стандартная технология, позволяющая "отображать" объекты на реляционные базы данных. POM-модель spring-boot-starter-data-jpa позволяет быстро начать работу. Она обеспечивает следующие ключевые зависимости:

* Hibernate одна из самых популярных реализаций JPA.
* Spring Data JPA: помогает реализовать репозитории на основе JPA.
* Spring ORM: основное средство поддержки ORM из Spring Framework. [©](https://javarush.com/quests/lectures/questspringboot.level03.lecture02)

**Spring DevTools** (spring-boot-devtools) автоматически запускает локальный экземпляр сервера LiveReload, который отслеживает ваши файлы. Все, что вам нужно сделать, это установить расширение для браузера, и готово. Он не только полезен для разработки внешнего интерфейса вашего приложения (в случае, если вы распространяете его как часть артефакта приложения Spring), но также может использоваться для мониторинга и перезагрузки вывода вашего REST API. [©](https://habr.com/ru/articles/479382/)

**Spring Test** (spring-boot-starter-test) — это одна из библиотек, входящих в Spring Framework, по сути все, что описано в разделе документации про интеграционное тестирование как раз о ней. Четыре главных задачи, которые решает библиотека это:

* управлять Spring IoC контейнерами и их кэшированием между тестами;
* предоставить внедрение зависимостей для тестовых классов;
* предоставить управление транзакциями, подходящее для интеграционных тестов;
* предоставить набор базовых классов чтобы помочь разработчику писать интеграционные тесты. [©](https://habr.com/ru/articles/446184/)

**MySQL** (mysql-connector-j) — свободная реляционная система управления базами данных (СУБД). Под словом «свободная» подразумевается ее бесплатность, под «реляционная» – работа с базами данных, основанных на двумерных таблицах. Система выпущена в 1995 году, её разработка активно продолжается. MySQL универсальна и применяется при разработке веб-сайтов, веб-приложений и корпоративных баз данных начального уровня. [©](https://www.nic.ru/help/chto-takoe-mysql_8510.html)

**Bootstrap** (bootstrap) (также известен как Twitter Bootstrap) — свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения. [©](https://ru.wikipedia.org/wiki/Bootstrap_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA))

**WebJars** (webjars-locator-core) также поддерживаются при помощи WebJarsResourceResolver, который автоматически регистрируется при наличии библиотеки org.webjars:webjars-locator-core в classpath. Распознаватель может переписывать URL-адреса, чтобы включить в состав версию jar, а также сопоставлять входящие URL-адреса без версий – например, из /jquery/jquery.min.js в /jquery/1.2.0/jquery.min.js. [©](https://javarush.com/quests/lectures/questspring.level04.lecture39)

1. **Разработка вэб – приложения**
   1. **Конфигурация Spring Security**

@Configuration  
@EnableWebSecurity  
public class WebSecurityConfig{  
  
 private UserJPAService userJPAService;  
 @Autowired  
 public void setUserService(UserJPAService *userJPAService*) {  
 this.userJPAService = *userJPAService*;  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity *http*) throws Exception {  
 *http* .authorizeHttpRequests((*requests*) -> *requests* .requestMatchers("/", "/home", "/login", "/students", "/signin" ,"/webjars/\*\*", "/ava\_bus.png", "/bus.svg", "signin.css", "style.css", "/registration").permitAll()  
 .requestMatchers("/routes/dispatcher/\*\*").hasRole("ADMIN")  
// .requestMatchers("/travels/\*\*").hasRole("DISPATCHER")  
 .anyRequest().authenticated()  
  
 // разрешить доступ всем  
// .anyRequest().permitAll()  
 )  
 .formLogin((*form*) -> *form* .loginPage("/login")  
 .defaultSuccessUrl("/", true)  
 .permitAll()  
 )  
 .logout(LogoutConfigurer::permitAll);  
  
 return *http*.build();  
 }  
  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public DaoAuthenticationProvider daoAuthenticationProvider() {  
 DaoAuthenticationProvider *authenticationProvider* = new DaoAuthenticationProvider();  
 *authenticationProvider*.setPasswordEncoder(passwordEncoder());  
 *authenticationProvider*.setUserDetailsService(userJPAService);  
 return *authenticationProvider*;  
 }  
}

* 1. **Конфигурация подключения к базе данных**

Для настройки подключения к базе данных необходим файл application.properties, который располагается по пути diplom\_tiket\src\main\resources.

server.port=8081  
spring.jpa.hibernate.ddl-auto=update  
spring.datasource.url=jdbc:mysql://${MYSQL\_HOST:localhost}:3306/tickets?createDatabaseIfNotExist=true&useSSL=false&useUnicode=yes&characterEncoding=utf8  
spring.datasource.username=\*  
spring.datasource.password=\*   
hibernate.dialect=org.hibernate.dialect.MySQL8Dialect   
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
server.error.whitelabel.enabled=false  
server.error.path=/error

В этом файле мы указываем порт, по которому у нас будет осуществляться доступ, в данном случае 8081, т.е. при локальном запуске приложения мы его увидим в строке браузера по адресу: <http://localhost:8081/> или <http://127.0.0.1:8081/>.

(spring.jpa.hibernate.ddl-auto=update) – эта строчка означает, что при запуске приложения мы будем обновлять нашу базу данных, если в сущностях, которые передают свои свойства в базу данных произошли изменения.

Следующая строчка отвечает за соединение с базой данных, а параметр (tickets?createDatabaseIfNotExist=true) означает, что если база данных удалена, то должна быть создана новая БД.

Следующие две строчки отвечают за доступ к базе данных и их я скрыл, в целях конфиденциальности. Последующие строчки отвечают за настройки при работе с СУБД и они стандартные. Последняя строчка будет отвечать за ошибки при обращении клиента к несуществующему адресу или адресу к которому у него нет доступа, будет показана страница с ошибкой (404 если страница не найдена).

* 1. **Основные сущности**

Теперь создадим папку **db** и папки **entity** и **repository** в ней, эта папка будет лежать по пути **src/main/java/chistTravel.tiket**. В папке entity будут содержаться основные сущности (классы) с которыми я буду работать, такие как: **Bus, Driver, LandingSite, RoleJPA, Route, Travels, UserJPA**. В папке **repository** будут содержаться основные сущности (классы) с которыми я буду работать, такие как: **BusRepo, DriverRepo, LandingSiteRepo, RoleRepository, RouteRepo, TravelsRepo, UserRepository**.

* + 1. **UserJPA**

@Entity  
@Table(name = "user\_jpa")  
public class UserJPA {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(unique = true, length = 100)  
 private String username;  
  
 @Column(length = 100)  
 private String password;  
  
 @Column(length = 50)  
 private String registered;  
  
 @ManyToMany  
 @JoinTable(name = "users\_roles\_jpa",  
 joinColumns = @JoinColumn(name = "user\_id\_jpa"),  
 inverseJoinColumns = @JoinColumn(name = "role\_id\_jpa"))  
 private Collection<RoleJPA> rolesJPA;  
  
 @Column(length = 100)  
 private String firstName; // имя  
  
 @Column(length = 100)  
 private String lastName; // фамилия  
  
 @Column(length = 100)  
 private String surName;  
  
 @Column(unique = true, length = 50)  
 private String phone;  
  
 @Column(length = 200)  
 private String comment;  
  
 @OneToMany(mappedBy = "user")  
 private List<Route> travels;

}

Геттеры (Getters), сеттеры (Setters), метод toString(), импорты (import) и путь к файлу (package) показывать не буду, чтобы сэкономить место. Полные классы будут на платформе **Git**.

**UserJPA** - это основная сущность, которая будет отвечать за пользователя, благодаря ней можно будет получить доступ к сайту (вэб – приложению). Аннотация @Entity нужна для того, чтобы Spring понимал, что это сущность. @Table – указывает, что эта сущность будет выглядеть в СУБД, как таблица, а свойства этого класса, как столбцы в этой таблице. @Column – указывает, что перед нами столбец в таблице. @ManyToMany – «многие ко многим» говорит нам, что у одного экземпляра этого класса может быть несколько ролей и для того, чтобы указывать будет создана отдельная таблица (@JoinTable) со своими столбцами (@JoinColumn). @OneToMany – «один ко многим» означает, что у одного пользователя может быть много поездок, которые будут храниться в списке. В базе данных это будет выглядеть, как таблица сущности **Route** в которой есть столбец, который ссылается на пользователей (их id).

Этот класс имеет такие свойства (поля):

* id – уникальный идентификатор,
* username – логин для доступа к сайту,
* password – пароль,
* registered – дата регистрации на сайте,
* rolesJPA – роль (уровень доступа),
* firstName – имя пользователя,
* lastName – фамилия,
* surname – отчество,
* phone – телефон,
* comment – комментарий (его смогут оставлять водители и диспетчера),
* travels – поездки, на которые записался пользователь.

**UserRepository:**

public interface UserRepository extends JpaRepository<UserJPA, Long> {  
  
 UserJPA findByUsername(String *username*);  
}

Этот интерфейс позволит нам использовать все экземпляры класса **UserJPA** как элемент репозиторий. Мы сможем при общении с базой данных использовать **CRUD** (создание (create), чтение (read), модификация (update), удаление (delete)).

* + 1. **RoleJPA**

@Entity  
@Table(name = "rolesJPA")  
public class RoleJPA {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String name;  
  
 @ManyToMany(mappedBy = "rolesJPA")  
 private Collection<UserJPA> usersJPA;  
}

**RoleJPA** - это основная сущность, которая будет отвечать за роль пользователя, благодаря ней можно будет предоставлять доступ к разным частям сайта разным группам пользователей. Существует особенность при заполнении базы данных с этой сущностью, мы должны будем записать в поле name = ROLE\_ADMIN, и тогда в файле WebSecurityConfig мы сможем написать requestMatchers("/routes/dispatcher/\*\*").hasRole("ADMIN"), что даст доступ по данному адресу только тем, у кого есть соответствующая роль.

Этот класс имеет такие свойства (поля):

* id – уникальный идентификатор,
* name – название роли,
* usersJPA – коллекция пользователей у которых есть эта роль.

**RoleRepository:**

@Repository  
public interface RoleRepository extends JpaRepository<RoleJPA, Long> {  
}

Этот интерфейс позволит нам использовать все экземпляры класса **RoleJPA** так же, как и **UserJPA** и использовать **CRUD**.

* + 1. **Bus**

@Entity  
@Table(name = "bus\_t")  
public class Bus {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false, length = 20)  
 private String busName;  
  
 @Column(nullable = false, length = 20)  
 private String number;  
  
 @Column(nullable = false, length = 20)  
 private int capacity;  
  
 @OneToMany(mappedBy = "bus")  
 private List<Travels> travels;

}

**Bus** – это сущность, которая будет отвечать за автобусы.

Этот класс имеет такие свойства (поля):

* id – уникальный идентификатор,
* busName – название автобуса (Марка и модель),
* number – номер автобуса ( АС 001 16 РУС),
* capacity – вместимость автобуса (количество мест),
* travels – поездки, на которых задействован данный автобус.

**BusRepo:**

public interface BusRepo extends CrudRepository<Bus, Long> {  
}

Этот интерфейс позволит нам использовать CRUD.

* + 1. **Driver**

@Entity  
@Table(name = "driver\_t")  
public class Driver {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false, length = 100)  
 private String firstName;  
  
 @Column(nullable = false, length = 100)  
 private String lastName;  
  
 @Column(nullable = false, length = 100)  
 private String surName;  
  
 @Column(nullable = false, length = 50)  
 private String phone;  
  
 @Column(nullable = true, length = 200)  
 private String comment;  
  
 @OneToMany(mappedBy = "driver")  
 private List<Travels> driverTravels;

}

**Driver** – это сущность, которая будет отвечать за водителя,

Этот класс имеет такие свойства (поля):

* id – уникальный идентификатор,
* firstName – имя,
* lastName – фамилия,
* surname – отчество,
* phone – телефон,
* comment – комментарий (его смогут оставлять диспетчера),
* driverTravels – поездки, на которые водителю предстоит ехать.

**DriverRepo:**

public interface DriverRepo extends CrudRepository<Driver, Long> {  
}

Этот интерфейс позволит нам использовать CRUD.

* + 1. **LandingSite**

@Entity  
@Table(name = "landingSite\_t")  
public class LandingSite {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Column(nullable = false, length = 100)  
 private String placeName;  
  
 @Column(nullable = false, length = 10)  
 private String time;  
  
 @OneToMany(mappedBy = "landingSite")  
 private List<Route> routes;  
  
 @Column  
 private boolean direction;

}

**LandingSite** - это сущность, которая будет отвечать за место посадки. Так как в своей работе я использую только два направления: Чистополь – Казань и Казань – Чистополь, то поле направление (direction) я сделал булевым (1 or 0, или true or false). Поле время я собираюсь использовать, как время в которое человек должен ждать в определённом месте. Например рейс из Чистополя в Казань в 18:00, в 18:00 отправление будет от остановки «Дом медиков», значит для остановки «Дом медиков» поле time будет равно нулю. А для остановки «ПМК» время стоит 17:30, т.е. автобус забирает с этой остановки заранее, и значит поле time в этом случает будет равно 30. Отображением правильного времени для пользователя будет заниматься контроллер, это мы увидим позже.

Этот класс имеет такие свойства (поля):

* id – уникальный идентификатор,
* placeName – название остановки,
* time – разница во времени для посадки,
* routes – список поездок где фигурирует эта остановка,
* direction – направление.

**LandingSiteRepo:**

public interface LandingSiteRepo extends JpaRepository<LandingSite, Long> {  
}

Этот интерфейс позволит нам использовать CRUD.

* + 1. **Travels**

@Entity  
@Table(name = "travels\_t")  
public class Travels {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @ManyToOne  
 @JoinColumn(name = "bus\_id", nullable = true)  
 private Bus bus;  
  
 @ManyToOne  
 @JoinColumn(name = "driver\_id", nullable = true)  
 private Driver driver;  
  
 @OneToMany(mappedBy = "travel")  
 private List<Route> routes;  
  
 @Column(nullable = true, length = 200)  
 private String timeToDrive;  
  
 @Column(nullable = true, length = 200)  
 private String timeParsed;  
  
 @Column(nullable = true, length = 200)  
 private String dateParsed;  
  
 @Column(nullable = true, length = 200)  
 private String dateFull;  
  
 @Column(nullable = true, length = 8)  
 private boolean forward;

}

**Travels** - это сущность, которая будет отвечать за рейсы, т.е. Во сколько, откуда и куда едет автобус, кто водитель и кто пассажиры. Аннотация @ManyToOne «многие к одному» означает, что в базе данных в данном столбце может быть много «ссылок» на одного водителя / автобус.

Этот класс имеет такие свойства (поля):

* id – уникальный идентификатор,
* bus – автобус,
* driver – водитель,
* routes – список пассажиров,
* timeToDrive – время и дата выезда,
* timeParsed – время выезда, для отображения на страницах сайта,
* dateParsed – дата выезда, для отображения на страницах сайта,
* dateFull – полное дата и время в формате LocalDateTime,
* forward – направление.

**TravelsRepo:**

public interface TravelsRepo extends CrudRepository<Travels, Long> {  
}

Этот интерфейс позволит нам использовать CRUD.

* + 1. **Route**

@Entity  
@Table(name = "route\_t")  
public class Route {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @ManyToOne  
 @JoinColumn(name = "user\_id")  
 private UserJPA user;  
  
 @ManyToOne  
 @JoinColumn(name = "travel\_id")  
 private Travels travel;  
  
 @ManyToOne  
 @JoinColumn(name = "landingSite\_id")  
 private LandingSite landingSite;  
  
 @Column(length = 30)  
 private String landingTime;  
  
 @Column(length = 200)  
 private String registered;  
  
 @Column(name = "status")  
 private Boolean status;

}

**Route** - это сущность, которая будет отвечать за поездки, включает в себя информацию о том какой пассажир записался на какой рейс и откуда и во сколько поедет.

Этот класс имеет такие свойства (поля):

* id – уникальный идентификатор,
* user – пассажир ,
* travel – рейс, на который записался пассажир,
* landingSite – место откуда он поедет,
* landingTime – время посадки,
* registered – когда была произведена запись на рейс,
* status – статус (в силе поездка, либо отменилась).

**RouteRepo:**

public interface RouteRepo extends CrudRepository<Route, Long> {  
}

Этот интерфейс позволит нам использовать CRUD.

* 1. **Сервисы**
     1. **MainController**

@Controller  
@RequestMapping("")  
public class MainController {  
  
 @GetMapping("")  
 public String showHomePage(Model *model*){  
 LocalDateTime *today* = LocalDateTime.*now*();  
 LocalDateTime *tomorrow* = LocalDateTime.*now*().plusDays(1);  
 LocalDateTime *dayAfterTomorrow* = LocalDateTime.*now*().plusDays(2);  
 DateTimeFormatter *formatter* = DateTimeFormatter.*ofPattern*("dd.MM.yyyy");  
 *model*.addAttribute("today",*formatter*.format(*today*));  
 *model*.addAttribute("tomorrow",*formatter*.format(*tomorrow*));  
 *model*.addAttribute("dayAfterTomorrow",*formatter*.format(*dayAfterTomorrow*));  
 return "index"; // index.html - шаблон  
 }  
}

* + 1. **AppErrorController**

// собственный контроллер обработки ошибок  
@Controller  
@RequestMapping()  
public class AppErrorController implements ErrorController {  
 // метод обработки запрос с ошибкой  
 @RequestMapping("/error")  
 public String handleError(HttpServletRequest *request*, Model *model*){  
 // ...  
 *model*.addAttribute("message", "Error: " +  
 *request*.getAttribute(RequestDispatcher.*ERROR\_STATUS\_CODE*));  
 return "layout/error";  
 }  
}

* + 1. **UserJPAService**

@Service  
public class UserJPAService implements UserDetailsService {  
 @Autowired  
 private UserRepository userRepository;  
 @Autowired  
 public void setUserRepository(UserRepository *userRepository*) {  
 this.userRepository = *userRepository*;  
 }  
 public UserJPA findByUsername(String *username*) {  
 return userRepository.findByUsername(*username*);  
 }  
  
 @Override  
 @Transactional  
 public UserDetails loadUserByUsername(String *username*) throws UsernameNotFoundException {  
 UserJPA *user* = findByUsername(*username*);  
 if(*user* == null) {  
 throw new UsernameNotFoundException(String.*format*("User '%s' not found", *username*));  
 }  
 return new org.springframework.security.core.userdetails.User(*user*.getUsername(), *user*.getPassword(),  
 mapRolesToAuthorities(*user*.getRolesJPA()));  
 }  
 private Collection<? extends GrantedAuthority> mapRolesToAuthorities(Collection<RoleJPA> *roles*) {  
 return *roles*.stream().map(*r* -> new SimpleGrantedAuthority(*r*.getName())).collect(Collectors.*toList*());  
 }  
 public UserJPA saveUserJPA(UserJPA *user*){  
 return userRepository.save(*user*);  
 }  
 public List<UserJPA> listAllUsers(){  
 return (List<UserJPA>) userRepository.findAll();  
 }  
}

* + 1. **RoleJPAService**

@Service  
public class RoleJPAService {  
 @Autowired  
 private RoleRepository repository;  
 public List<RoleJPA> listAllRoles(){  
 return repository.findAll();  
 }  
}

* + 1. **BusService**

@Service  
public class BusService {  
  
 @Autowired  
 private BusRepo repository;  
  
 public List<Bus> listAllBus(){  
 return (List<Bus>) repository.findAll();  
 }  
  
 public Bus saveBus(Bus *bus*){  
 return repository.save(*bus*);  
 }  
}

* + 1. **DriverService**

@Service  
public class DriverService {  
  
 @Autowired  
 private DriverRepo repository;  
  
 public List<Driver> listAllDrivers(){  
 return (List<Driver>) repository.findAll();  
 }  
  
 public Driver saveDriver(Driver *driver*){  
 return repository.save(*driver*);  
 }  
}

* + 1. **LandingSiteService**

@Service  
public class LandingSiteService {  
 @Autowired  
 private LandingSiteRepo repository;  
  
 public List<LandingSite> listAllLandingSites(){  
 return (List<LandingSite>) repository.findAll();  
 }  
  
 public LandingSite saveLandingSite(LandingSite *landingSite*){  
 return repository.save(*landingSite*);  
 }  
  
 public LandingSite findLandingSiteById(Long *id*){  
 Optional<LandingSite> *landingSite* = repository.findById(*id*);  
 return *landingSite*.orElse(null);  
 }  
}

* + 1. **TravelService**

@Service  
public class TravelService {  
 @Autowired  
 private TravelsRepo repository;  
  
 public List<Travels> listAllTravels(){  
 return (List<Travels>) repository.findAll();  
 }  
  
 public Travels saveTravel(Travels *travels*){  
 return repository.save(*travels*);  
 }  
  
 public Travels findTravelsById(Long *id*){  
 Optional<Travels> *travels* = repository.findById(*id*);  
 return *travels*.orElse(null);  
 }  
  
  
 public List<Travels> findTravelsByDateForward(String *date*){  
 List<Travels> *travels* = (List<Travels>) repository.findAll();  
 List<Travels> *travelsForDate* = new ArrayList<>();  
 for (Travels *travel*: *travels*) {  
 if (*travel*.getDateParsed().equals(*date*) && *travel*.isForward()) {  
 *travelsForDate*.add(*travel*);  
 }  
 }  
 *travelsForDate*.sort(Comparator.*comparing*(Travels::getTimeParsed));  
 return *travelsForDate*;  
 }  
  
 public List<Travels> findTravelsByDateBack(String *date*){  
 List<Travels> *travels* = (List<Travels>) repository.findAll();  
 List<Travels> *travelsForDate* = new ArrayList<>();  
 for (Travels *travel*: *travels*) {  
 if (*travel*.getDateParsed().equals(*date*) && !*travel*.isForward()) {  
 *travelsForDate*.add(*travel*);  
 }  
 }  
 *travelsForDate*.sort(Comparator.*comparing*(Travels::getTimeParsed));  
 return *travelsForDate*;  
 }  
  
 public List<Travels> findTravelsByDateDisabled(String *dateParsed*){  
 List<Travels> *travels* = (List<Travels>) repository.findAll();  
 List<Travels> *travelsForDateDisabled* = new ArrayList<>();  
 LocalTime *timeNow* = LocalTime.*now*();  
 LocalDate *dateNow* = LocalDate.*now*();  
 System.*out*.println("date = " + *dateParsed*);  
 System.*out*.println("dateNow = " + *dateNow*);  
 System.*out*.println("timeNow = " + *timeNow*);  
 LocalTime *timePlusHour* = *timeNow*.plusHours(1);  
 System.*out*.println(*timePlusHour*);  
  
 return *travelsForDateDisabled*;  
 }  
}

* + 1. **RouteService**

@Service  
public class RouteService {  
 @Autowired  
 private RouteRepo repository;  
  
 public List<Route> listAllRoute(){  
 return (List<Route>) repository.findAll();  
 }  
  
 public Route saveRoute(Route *route*){  
 return repository.save(*route*);  
 }  
  
 public void deleteRouteById(Long *id*){  
 Optional<Route> *deleted* = repository.findById(*id*);  
 *deleted*.ifPresent(*route* -> repository.delete(*route*));  
 }  
  
 public void statusToFalseRouteById(Long *id*){  
 Optional<Route> *deleted* = repository.findById(*id*);  
 *deleted*.ifPresent(*route* -> *route*.setStatus(false));  
 *deleted*.ifPresent(*route* -> saveRoute(*deleted*.get()));  
 }  
  
 public void statusToTrueRouteById(Long *id*){  
 Optional<Route> *deleted* = repository.findById(*id*);  
 *deleted*.ifPresent(*route* -> *route*.setStatus(true));  
 *deleted*.ifPresent(*route* -> saveRoute(*deleted*.get()));  
 }  
}

* 1. **Контроллеры**
     1. **UserJPAController**

@Controller  
@RequestMapping("/registration")  
public class UserJPAController {  
 @Autowired  
 private UserJPAService userJPAService;  
 @Autowired  
 private RoleJPAService roleJPAService;  
  
 private LocalDateTime time;  
  
 @GetMapping("")  
 public String showNewRegistrationForm(Model *model*) {  
 *model*.addAttribute("userJPA", new UserJPA());  
 return "registration";  
 }  
  
 @PostMapping("")  
 private String saveNewUserJPA(UserJPA *user*, RedirectAttributes *ra*) {  
 BCryptPasswordEncoder *passwordEncoder* = new BCryptPasswordEncoder(8);  
 String *hashedPassword* = *passwordEncoder*.encode(*user*.getPassword());  
 *user*.setPassword(*hashedPassword*);  
 ArrayList<RoleJPA> *roles* = new ArrayList<>();  
 *roles*.add(roleJPAService.listAllRoles().get(0));  
 *roles*.add(roleJPAService.listAllRoles().get(1));  
 *roles*.add(roleJPAService.listAllRoles().get(2));  
 *user*.setRolesJPA(*roles*);  
  
 time = LocalDateTime.*from*(ZonedDateTime.*now*(ZoneId.*of*("Africa/Addis\_Ababa")));  
 DateTimeFormatter *formatter* = DateTimeFormatter.*ofPattern*("dd.MM.yyyy HH:mm");  
 *user*.setRegistered(*formatter*.format(time));  
  
 UserJPA *saved* = userJPAService.saveUserJPA(*user*);  
 *ra*.addFlashAttribute("message", "User " + *saved* + " saved successfully");  
 return "redirect:/";  
 }  
  
 @GetMapping("/show")  
 private String showAllUsers(Model *model*) {  
 List<UserJPA> *usersList* = userJPAService.listAllUsers();  
  
 for (UserJPA *user*: *usersList*) {  
  
 List<Route> *travels* = *user*.getTravels();  
 *travels*.sort(Comparator.*comparing*(Route::getLandingTime));  
 }  
 *model*.addAttribute("usersList", *usersList*);  
 return "users-list";  
 }  
}

* + 1. **BusController**

@Controller  
@RequestMapping("/bus")  
public class BusController {  
 @Autowired  
 private BusService busService;  
  
 @GetMapping("")  
 private String showAllBus(Model *model*) {  
 List<Bus> *busList* = busService.listAllBus();  
 *model*.addAttribute("busList", *busList*);  
 return "bus-list";  
 }  
  
 @GetMapping("/new")  
 public String showNewBus(Model *model*) {  
 *model*.addAttribute("bus", new Bus());  
 return "bus-form";  
 }  
  
 @PostMapping("/new")  
 private String saveNewBus(Bus *bus*, RedirectAttributes *ra*) {  
 Bus *saved* = busService.saveBus(*bus*);  
 *ra*.addFlashAttribute("message", "Bus " + *saved* + " saved successfully");  
 return "redirect:/bus";  
 }  
}

* + 1. **DriverController**

@Controller  
@RequestMapping("/drivers")  
public class DriverController {  
 @Autowired  
 private DriverService driverService;  
  
 @GetMapping("")  
 private String showAllUsers(Model *model*) {  
 List<Driver> *driverList* = driverService.listAllDrivers();  
 *model*.addAttribute("driverList", *driverList*);  
 return "drivers-list";  
 }  
  
 @GetMapping("/new")  
 public String showNewDriverForm(Model *model*) {  
 *model*.addAttribute("driver", new Driver());  
 return "driver-form";  
 }  
  
 @PostMapping("/new")  
 private String saveNewDriver(Driver *driver*, RedirectAttributes *ra*) {  
   
 Driver *saved* = driverService.saveDriver(*driver*);  
   
 *ra*.addFlashAttribute("message", "Driver " + *saved* + " saved successfully");  
 return "redirect:/drivers";  
 }  
}

* + 1. **OrderController**

@Controller  
@RequestMapping("/order")  
public class OrderController {  
  
 @Autowired  
 private TravelService travelService;  
  
 private LocalDateTime timeNow;  
  
 @Autowired  
 private UserJPAService userJPAService;  
  
 @Autowired  
 private RouteService routeService;  
 @Autowired  
 private LandingSiteService landingSiteService;  
  
 @GetMapping("")  
 private String showOrder(Model *model*) {  
 List<Travels> *travelsLists* = travelService.listAllTravels();  
 *model*.addAttribute("travelsLists", *travelsLists*);  
 return "travels-list";  
 }  
  
 @GetMapping("/{date}") // chistopol-kazan  
 public String detailsTravelsForward(@PathVariable("date") String *date*, Model *model*) {  
 String [] *dateSplit* = *date*.split(":");  
 String *dateParsed* = *dateSplit*[1];  
 String *direction* = *dateSplit*[0];  
 List<Travels> *travelsDateParsed* = new ArrayList<>();  
 if (*direction*.equals("chistopol-kazan")) {  
 *travelsDateParsed* = travelService.findTravelsByDateForward(*dateParsed*);  
 *model*.addAttribute("directionName", "Чистополь - Казань");  
 } else if (*direction*.equals("kazan-chistopol")) {  
 *travelsDateParsed* = travelService.findTravelsByDateBack(*dateParsed*);  
 *model*.addAttribute("directionName", "Казань - Чистополь");  
 } else {  
 *model*.addAttribute("directionName", "Ошибка направления");  
 }  
 timeNow = LocalDateTime.*from*(ZonedDateTime.*now*(ZoneId.*of*("Africa/Addis\_Ababa")));  
 LocalTime *timeHourPlus* = LocalTime.*from*(timeNow).plusHours(1);  
 int *dateDay* = LocalDate.*from*(timeNow).getDayOfYear();  
 List<Travels> *travelsValid* = new ArrayList<>();  
 List<Travels> *travelsInvalid* = new ArrayList<>();  
 for (Travels *travel* : *travelsDateParsed*) {  
 LocalTime *time* = LocalTime.*parse*(*travel*.getTimeParsed());  
 int *localDate* = LocalDateTime.*parse*(*travel*.getDateFull()).getDayOfYear();  
 // рейсы через час  
 if (*timeHourPlus*.isBefore(*time*) && *localDate* == *dateDay* && *timeHourPlus*.getHour() >= 1 || *localDate* > *dateDay* && *localDate* < *dateDay* + 3) {  
 *travelsValid*.add(*travel*);  
 } else {  
 *travelsInvalid*.add(*travel*);  
 }  
 }  
 *model*.addAttribute("travelsDateParsed", *travelsDateParsed*);  
 *model*.addAttribute("travelsValid", *travelsValid*);  
 *model*.addAttribute("travelsInvalid", *travelsInvalid*);  
 *model*.addAttribute("dateParsed", *dateParsed*);  
 *model*.addAttribute("direction", *direction*);   
 return "order-list";  
 }  
  
 @GetMapping("/{date}/{id}")  
 public String detailsOrder(@PathVariable("id") Long *id*, @PathVariable("date") String *date*, Model *model*) {  
 String [] *dateSplit* = *date*.split(":");  
 String *dateParsed* = *dateSplit*[1];  
 String *direction* = *dateSplit*[0];  
 *model*.addAttribute("dateParsed", *dateParsed*);  
 *model*.addAttribute("direction", *direction*);  
 if (*direction*.equals("chistopol-kazan")) {  
 *model*.addAttribute("directionName", "Чистополь - Казань");  
 } else if (*direction*.equals("kazan-chistopol")) {  
 *model*.addAttribute("directionName", "Казань - Чистополь");  
 } else {  
 *model*.addAttribute("directionName", "Ошибка направления");  
 }  
 Travels *travel* = travelService.findTravelsById(*id*);  
 *model*.addAttribute("travels", *travel*);  
 List<Route> *usersInTravel* = *travel*.getRoutes();  
 *usersInTravel*.sort(Comparator.*comparing*(Route::getLandingTime));  
 *model*.addAttribute("usersInTravel", *usersInTravel*);   
 *model*.addAttribute("route", new Route());  
 List<LandingSite> *landingSites* = landingSiteService.listAllLandingSites();  
 LocalTime *time* = LocalTime.*parse*(*travel*.getTimeParsed());  
 for (LandingSite *landingSite* : *landingSites*) {  
 LocalTime *time1* = *time*.minusMinutes(Long.*parseLong*(*landingSite*.getTime()));  
 *landingSite*.setTime(*time1*.toString());  
 }  
 *landingSites*.sort(Comparator.*comparing*(LandingSite::getTime));  
 *model*.addAttribute("landingSites", *landingSites*);  
 return "order-details";  
 }  
  
 @PostMapping ("/{date}/{id}")  
 public String detailsOrderPost(@PathVariable("id") Long *id*, @PathVariable("date") String *date*, Route *route*, Principal *principal*, RedirectAttributes *ra*) {  
 *route*.setStatus(true);  
 timeNow = LocalDateTime.*from*(ZonedDateTime.*now*(ZoneId.*of*("Africa/Addis\_Ababa")));  
 DateTimeFormatter *formatter* = DateTimeFormatter.*ofPattern*("dd.MM.yyyy HH:mm");  
 *route*.setRegistered(*formatter*.format(timeNow));  
 *route*.setUser(userJPAService.findByUsername(*principal*.getName()));  
 Travels *travel* = travelService.findTravelsById(*id*);  
 *route*.setTravel(*travel*);  
 LocalTime *time* = LocalTime.*parse*(*travel*.getTimeParsed());  
 LocalTime *time1* = *time*.minusMinutes(Long.*parseLong*(*route*.getLandingSite().getTime()));  
 *route*.setLandingTime(*time1*.toString());  
 Route *saved* = routeService.saveRoute(*route*);   
 *ra*.addFlashAttribute("message", "Вы успешно записались на рейс");  
 return "redirect:/order/{date}";  
 }  
}

* + 1. **TravelController**

@Controller  
@RequestMapping("/travels")  
public class TravelController {  
  
 @Autowired  
 private TravelService travelService;  
  
 @Autowired  
 private BusService busService;  
  
 @Autowired  
 private DriverService driverService;  
  
 private LocalDateTime timeNow;  
  
 @GetMapping("")  
 private String showAllTravels(Model *model*) {  
 List<Travels> *travelsLists* = travelService.listAllTravels();  
 *travelsLists*.sort(Comparator.*comparing*(Travels::getDateFull));  
 *model*.addAttribute("travelsLists", *travelsLists*);  
 return "travels-list";  
 }  
}

* + 1. **RouteController**

@Controller  
@RequestMapping("/routes")  
public class RouteController {  
 @Autowired  
 private RouteService routeService;  
 @Autowired  
 private UserService userService;  
 @Autowired  
 private UserJPAService userJPAService;  
 @Autowired  
 private TravelService travelService;  
  
 private LocalDateTime time;  
  
 @GetMapping("")  
 private String showAllTravelLists(Model *model*) {  
 List<Route> *routeList* = routeService.listAllRoute();  
 *model*.addAttribute("routeList", *routeList*);  
 return "routes-list";  
 }  
  
 @GetMapping("/new")  
 public String showNewUserForm(Model *model*) {  
 *model*.addAttribute("route", new Route());  
 List<UserJPA> *userList* = userService.listAllUsers();  
 *model*.addAttribute("userList", *userList*);  
 List<Travels> *travelsLists* = travelService.listAllTravels();  
 *model*.addAttribute("travelsLists", *travelsLists*);  
 return "route-form";  
 }  
  
 @PostMapping("/new")  
 private String saveNewUser(Route *route*, RedirectAttributes *ra*) {  
 *route*.setStatus(true);  
 time = LocalDateTime.*now*();  
 DateTimeFormatter *formatter* = DateTimeFormatter.*ofPattern*("dd.MM.yyyy HH:mm");  
 *route*.setRegistered(*formatter*.format(time));  
 Route *saved* = routeService.saveRoute(*route*);  
 *ra*.addFlashAttribute("message", "Route " + *saved* + " saved successfully");  
 return "redirect:/routes";  
 }  
  
 @GetMapping("/deleteFull/{id}")  
 public String deleteRoute(@PathVariable("id") Long *id*, RedirectAttributes *ra*) {  
 routeService.deleteRouteById(*id*);  
 *ra*.addFlashAttribute("message", "Route deleted");  
 return "redirect:/routes";  
 }  
  
 @GetMapping("/delete/{id}")  
 public String statusToFalseRoute(@PathVariable("id") Long *id*, RedirectAttributes *ra*) {  
 routeService.statusToFalseRouteById(*id*);  
 *ra*.addFlashAttribute("message", "Route deleted by status");  
 return "redirect:/routes";  
 }  
  
 @GetMapping("/repair/{id}")  
 public String statusToTrueRoute(@PathVariable("id") Long *id*, RedirectAttributes *ra*) {  
 routeService.statusToTrueRouteById(*id*);  
 *ra*.addFlashAttribute("message", "Route repair by status");  
 return "redirect:/routes";  
 }  
  
 @GetMapping("/dispatcher1")  
 private String showAllTravelListsToDispatcher(Model *model*, Principal *principal*) {  
 List<Route> *routeList* = routeService.listAllRoute();  
 List<Route> *routeList2* = new ArrayList<>();  
 for (int *i* = 0; *i* < *routeList*.size(); *i*++) {  
 if (*routeList*.get(*i*).getStatus().equals(true)){  
 *routeList2*.add(*routeList*.get(*i*));  
 }  
 }  
 *model*.addAttribute("routeList", *routeList2*); *model*.addAttribute("userJPA",userJPAService.findByUsername(*principal*.getName()));  
 return "routes-list-dispatcher";  
 }  
  
 @GetMapping("/dispatcher/delete/{id}")  
 public String statusToFalseRouteDispatcher(@PathVariable("id") Long *id*, RedirectAttributes *ra*) {  
 routeService.statusToFalseRouteById(*id*);  
 *ra*.addFlashAttribute("message", "Route deleted by status");  
 return "redirect:/routes/dispatcher";  
 }  
}

* + 1. **DispatcherTravelController**

@Controller  
@RequestMapping("/dispatcher/travel")  
public class DispatcherTravelController {  
  
 @Autowired  
 private TravelService travelService;  
  
 @Autowired  
 private BusService busService;  
  
 @Autowired  
 private LandingSiteService landingSiteService;  
  
 @Autowired  
 private DriverService driverService;  
  
 private LocalDateTime time;  
  
 @GetMapping("")  
 private String showAllTravels(Model *model*) {  
 List<Travels> *travelsLists* = travelService.listAllTravels();  
 *model*.addAttribute("travelsLists", *travelsLists*);  
 return "dispatcher-travels-list";  
 }  
  
 @GetMapping("/new")  
 public String showNewTravelsForm(Model *model*) {  
 *model*.addAttribute("travels", new Travels());  
 List<Bus> *busList* = busService.listAllBus();  
 *model*.addAttribute("busList", *busList*);  
 List<Driver> *driverList* = driverService.listAllDrivers();  
 *model*.addAttribute("driverList", *driverList*);  
 return "travel-form";  
 }  
  
 @PostMapping("/new")  
 private String saveNewTravels(Travels *travels*, RedirectAttributes *ra*) {  
  
 time = LocalDateTime.*parse*(*travels*.getTimeToDrive());  
 DateTimeFormatter *formatter* = DateTimeFormatter.*ofPattern*("dd.MM.yyyy HH:mm");  
 *travels*.setDateFull(time.toString());  
 *travels*.setTimeToDrive(*formatter*.format(time));  
 *travels*.setDateParsed(*formatter*.format(time).split(" ")[0]);  
 *travels*.setTimeParsed(*formatter*.format(time).split(" ")[1]);  
 Travels *saved* = travelService.saveTravel(*travels*);   
 *ra*.addFlashAttribute("message", "Travels " + *saved* + " saved successfully");  
 return "redirect:/dispatcher/travels";  
 }  
  
 @GetMapping("/{date}/{id}")  
 public String detailsOrder(@PathVariable("id") Long *id*, @PathVariable("date") String *date*, Model *model*) {  
 String [] *dateSplit* = *date*.split(":");  
 String *dateParsed* = *dateSplit*[1];  
 String *direction* = *dateSplit*[0];  
 *model*.addAttribute("dateParsed", *dateParsed*);  
 *model*.addAttribute("direction", *direction*);  
 if (*direction*.equals("chistopol-kazan")) {  
 *model*.addAttribute("directionName", "Чистополь - Казань");  
 } else if (*direction*.equals("kazan-chistopol")) {  
 *model*.addAttribute("directionName", "Казань - Чистополь");  
 } else {  
 *model*.addAttribute("directionName", "Ошибка направления");  
 }  
 Travels *travel* = travelService.findTravelsById(*id*);  
 *model*.addAttribute("travels", *travel*);  
 List<Route> *usersInTravel* = *travel*.getRoutes();  
 *usersInTravel*.sort(Comparator.*comparing*(Route::getLandingTime));  
 *model*.addAttribute("usersInTravel", *usersInTravel*);   
 *model*.addAttribute("route", new Route());  
 List<LandingSite> *landingSites* = landingSiteService.listAllLandingSites();  
 LocalTime *time* = LocalTime.*parse*(*travel*.getTimeParsed());  
 for (LandingSite *landingSite* : *landingSites*) {  
 LocalTime *time1* = *time*.minusMinutes(Long.*parseLong*(*landingSite*.getTime()));  
 *landingSite*.setTime(*time1*.toString());  
 }  
 *landingSites*.sort(Comparator.*comparing*(LandingSite::getTime));  
 *model*.addAttribute("landingSites", *landingSites*);  
 return "dispatcher-order-details";  
 }  
  
}

* + 1. **DispatcherController**

@Controller  
@RequestMapping("/dispatcher")  
public class DispatcherController {  
 @Autowired  
 private TravelService travelService;  
  
 @GetMapping("")  
 private String showAllTravels(Model *model*) {  
 List<Travels> *travelsLists* = travelService.listAllTravels();  
 *model*.addAttribute("travelsLists", *travelsLists*);  
 return "dispatcher-list";  
 }  
  
 @GetMapping("/new/{id}")  
 public String detailsTravels(@PathVariable("id") Long *id*, Model *model*) {  
 Travels *travels* = travelService.findTravelsById(*id*);  
 *model*.addAttribute("travels", *travels*);  
 List<Route> *usersInTravel* = *travels*.getRoutes();  
 *model*.addAttribute("usersInTravel", *usersInTravel*);  
  
 List<Integer> *listNumbers* = new ArrayList<>();  
 for (int *i* =1; *i* <=*usersInTravel*.size(); *i* ++){  
 *listNumbers*.add(*i*);  
 }  
 *model*.addAttribute("listNumbers", *listNumbers*);  
 *model*.addAttribute("listNumbersSize", *listNumbers*.size());  
 return "dispatcher-form";  
 }  
  
}

* 1. **Представления**
     1. **index.html**

<!DOCTYPE html>  
<html lang="ru" class="h-100" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
  
 <title>Index</title>  
  
 <link rel="shortcut icon" th:href="@{bus.svg}" type="image/svg">  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
 <link href="/style.css" rel="stylesheet">  
</head>  
  
<body>  
<nav th:insert="~{layout/layouts.html :: navs}"></nav>  
  
<div class="container-fluid text-center m-auto">  
 <main>  
 <div th:if="${message != null}" class="alert alert-success">  
 [[${message}]]  
 </div>  
  
 <h1>Пассажирские перевозки</h1>  
 <img src="/ava\_bus.png" alt="" width="200" height="200">  
  
 <div class="container-fluid text-center mt-5">  
 <h3>Чистополь - Казань</h3>  
 <div class="me-auto mb-2 mb-sm-0 row-cols-auto">  
 <a th:href="@{'/order/chistopol-kazan:' + ${today}}" class="btn btn-lg btn-warning index\_button">Сегодня ([[${today}]])</a>  
 <a th:href="@{'/order/chistopol-kazan:' + ${tomorrow}}" class="btn btn-lg btn-warning index\_button">Завтра ([[${tomorrow}]])</a>  
 <a th:href="@{'/order/chistopol-kazan:' + ${dayAfterTomorrow}}" class="btn btn-lg btn-warning index\_button">Послезавтра ([[${dayAfterTomorrow}]])</a>  
 </div>  
  
 <h3 class="mt-5">Казань - Чистополь</h3>  
 <div class="me-auto mb-2 mb-sm-0 row-cols-auto">  
 <a th:href="@{'/order/kazan-chistopol:' + ${today}}" class="btn btn-lg btn-warning index\_button">Сегодня ([[${today}]])</a>  
 <a th:href="@{'/order/kazan-chistopol:' + ${tomorrow}}" class="btn btn-lg btn-warning index\_button">Завтра ([[${tomorrow}]])</a>  
 <a th:href="@{'/order/kazan-chistopol:' + ${dayAfterTomorrow}}" class="btn btn-lg btn-warning index\_button">Послезавтра ([[${dayAfterTomorrow}]])</a>  
 </div>  
 </div>  
 </main>  
</div>  
  
<footer class="footer mt-auto py-3" th:insert="~{layout/layouts.html :: footer}"></footer>  
</body>  
</html>

* + 1. **layouts.html**

<!DOCTYPE html>  
<html lang="ru" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <meta name="description" content="">  
  
 <title>Layouts</title>  
  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
</head>  
<body>  
<nav class="navbar navbar-expand-sm navbar-dark bg-dark" aria-label="Third navbar example" th:fragment="navs">  
 <div class="container-fluid">  
 <a class="navbar-brand" th:href="@{/}">На главную</a>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarsExample03"  
 aria-controls="navbarsExample03" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
  
 <div class="collapse navbar-collapse" id="navbarsExample03">  
 <ul class="navbar-nav me-auto mb-2 mb-sm-0">  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/users}">Users</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/drivers}">Drivers</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/bus}">Bus</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/travels}">Travels</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/routes}">Routes</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/routes/dispatcher1}">Dispatcher</a>  
 </li>  
 <li class="nav-item">  
 <a class="nav-link active" aria-current="page" th:href="@{/dispatcher/travel}">DisTravels</a>  
 </li>  
 </ul>  
 </div>  
 <form th:action="@{/logout}" method="post">  
 <button class="btn btn-danger">Выйти</button>  
 </form>  
 </div>  
 <script src="/js/bootstrap.bundle.min.js"></script>  
</nav>  
  
  
<!--фрагмент хедер login-->  
<nav class="navbar navbar-expand-sm navbar-dark bg-dark" aria-label="Third navbar example" th:fragment="navs\_login">  
 <div class="container-fluid justify-content-center">  
 <h1 class="text-light">Пассажирские перевозки</h1>  
 </div>  
</nav>  
  
<!--фрагмент хедер index-->  
<nav class="navbar navbar-expand-sm navbar-dark bg-dark" aria-label="Third navbar example" th:fragment="navs\_index">  
 <div class="container-fluid justify-content-end">  
 <a th:href="@{/login}" class="btn btn-lg btn-primary">Вход</a>  
 <a th:href="@{/registration}" class="btn btn-lg btn-success mx-sm-3 ms-1">Регистрация</a>  
 </div>  
</nav>  
  
<!--фрагмент footer-->  
<footer class="text-muted text-center text-small" th:fragment="footer">  
 <div class="container">  
 <p class="mb-1 unselectable">&copy; 2024 Евгений Хозин</p>  
 </div>  
</footer>  
  
</body>  
</html>

* + 1. **error.html**

<!DOCTYPE html>  
<html lang="ru" class="h-100" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <meta name="description" content="">  
 <meta name="generator" content="Hugo 0.84.0">  
 <title>Ошибка</title>  
 <link rel="shortcut icon" th:href="@{/bus.svg}" type="image/svg">  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
 <link rel="stylesheet" href="/style.css"/>  
</head>  
<body class="d-flex flex-column h-100 bg-light">  
<nav th:insert="~{layout/layouts.html :: navs}"></nav>  
<div class="container-fluid text-center error\_margin-auto">  
 <main class="flex-shrink-0">  
 <div class="container">  
 <h1>Что-то пошло нет так! </h1>  
 <h2>[[${message}]]</h2>  
 <a th:href="@{/}">На главную</a>  
 </div>  
 </main>  
</div>  
<footer class="footer mt-auto py-3" th:insert="~{layout/layouts.html :: footer}"></footer>  
</body>  
</html>

* + 1. **login.html**

<!DOCTYPE html>  
<html lang="ru" class="h-100" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <meta name="description" content="">  
  
 <title>Войти</title>  
  
 <link rel="shortcut icon" th:href="@{bus.svg}" type="image/svg">  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
 <link href="/signin.css" rel="stylesheet">  
 <link href="/style.css" rel="stylesheet">  
</head>  
  
<body class="d-flex flex-column h-100 bg-light text-center">  
<nav th:insert="~{layout/layouts.html :: navs\_login}"></nav>  
  
 <main class="form-signin">  
  
 <div th:if="${param.error}">  
 Неправильный логин или пароль  
 </div>  
  
 <form th:action="@{/login}" method="post">  
 <img class="mb-4" src="/ava\_bus.png" alt="" width="200" height="200">  
 <h1 class="h3 mb-3 fw-normal">Вход</h1>  
  
 <div class="form-floating">  
 <input type="text" class="form-control" name="username" placeholder="89999999999">  
 <label>Логин</label>  
 </div>  
 <div class="form-floating">  
 <input type="password" class="form-control" name="password" placeholder="Пароль">  
 <label>Пароль</label>  
 </div>  
  
 <div class="checkbox mb-3">  
 <label>  
 <input type="checkbox" value="remember-me"> Запомнить меня  
 </label>  
 </div>  
 <button class="w-100 btn btn-lg btn-primary" type="submit">Войти</button>  
  
 </form>  
 <div class="d-flex justify-content-end mt-2">  
 <a th:href="@{/registration}" class="header">Регистрация</a>  
 </div>  
  
 </main>  
<footer class="footer mt-auto py-3" th:insert="~{layout/layouts.html :: footer}"></footer>  
</body>  
</html>

* + 1. **registration.html**

<!DOCTYPE html>  
<html lang="ru" class="h-100" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <meta name="description" content="">  
  
 <title>Регистрация</title>  
  
 <link rel="shortcut icon" th:href="@{/bus.svg}" type="image/svg">  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
 <link href="/signin.css" rel="stylesheet">  
 <link href="/style.css" rel="stylesheet">  
</head>  
  
<body class="text-center">  
<nav th:insert="~{layout/layouts.html :: navs\_login}"></nav>  
  
<div class="container text-center m-auto">  
  
 <main>  
 <div class="py-5 text-center">  
 <img class="d-block mx-auto mb-4" src="/bus.svg" alt="" width="72" height="57">  
 <h2>Регистрация</h2>  
 <i class="bi bi-bus-front-fill"></i>  
 </div>  
  
 <div class="row justify-content-center">  
 <div class="col-md-7 col-lg-8">  
 <form th:action="@{/registration}" th:object="${userJPA}" method="post" class="needs-validation"  
 novalidate>  
 <div class="row g-3">  
 <div class="col-sm-6">  
 <label class="form-label">Фамилия</label>  
 <input th:field="\*{lastName}" minlength="1" maxlength="50" type="text" class="form-control"  
 placeholder="Иванов" value="" required>  
 <div class="invalid-feedback">  
 Требуется фамилия  
 </div>  
 </div>  
  
 <div class="col-sm-6">  
 <label class="form-label">Имя</label>  
 <input th:field="\*{firstName}" minlength="1" maxlength="50" type="text" class="form-control"  
 placeholder="Иван" value="" required>  
 <div class="invalid-feedback">  
 Требуется имя  
 </div>  
 </div>  
  
 <div class="col-sm-6">  
 <label class="form-label">Отчество</label>  
 <input th:field="\*{surName}" maxlength="50" type="text" class="form-control"  
 placeholder="Иванович" value="">  
 <!-- <div class="invalid-feedback">-->  
 <!-- Valid first name is required.-->  
 <!-- </div>-->  
 </div>  
  
 <div class="col-sm-6">  
 <label class="form-label">Телефон</label>  
 <input th:field="\*{phone}" minlength="1" maxlength="50" type="tel" class="form-control"  
 placeholder="" value="" required>  
 <div class="invalid-feedback">  
 Требуется телефон  
 </div>  
 </div>  
  
 <div class="col-12">  
 <label class="form-label">Логин</label>  
 <div class="input-group has-validation">  
 <span class="input-group-text">@</span>  
 <input th:field="\*{username}" minlength="1" maxlength="50" type="text"  
 class="form-control" placeholder="" required>  
 <div class="invalid-feedback">  
 Требуется логин  
 </div>  
 </div>  
 </div>  
  
 <div class="col-12">  
 <label class="form-label">Пароль</label>  
 <input th:field="\*{password}" minlength="1" maxlength="50" type="password"  
 class="form-control" placeholder="" required>  
 <div class="invalid-feedback">  
 Требуется пароль  
 </div>  
 </div>  
 </div>  
  
 <hr class="my-4">  
 <button class="w-100 btn btn-primary btn-lg" type="submit">Зарегистрироваться</button>  
 </form>  
 </div>  
 </div>  
 </main>  
  
  
</div>  
<footer class="footer mt-auto py-3" th:insert="~{layout/layouts.html :: footer}"></footer>  
<script src="/form-validation.js"></script>  
</body>  
</html>

* + 1. **order-list.html**

<!DOCTYPE html>  
<html lang="ru" class="h-100" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
  
 <title>Order list</title>  
  
 <link rel="shortcut icon" href="/bus.svg" type="image/svg">  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
 <link href="/style.css" rel="stylesheet">  
</head>  
<body>  
<nav th:insert="~{layout/layouts.html :: navs}"></nav>  
<div class="container-fluid text-center m-auto">  
 <main>  
 <h1>Рейсы [[${directionName}]] [[${dateParsed}]]</h1>  
 <div th:if="${message != null}" class="alert alert-success">  
 [[${message}]]  
 </div>  
 <div class="table-responsive container mt-5">  
 <table class="table table-borderless">  
 <thead>  
 <th:block th:each="travels: ${travelsInvalid}">  
 <tr>  
 <a th:href="@{'/order/' + ${direction} + ':' + ${dateParsed} + '/' + ${travels.id}}"  
 class="btn btn-lg btn-secondary travel\_button disabled px-1">  
 <p>[[${travels.timeParsed}]]</p>  
 <p>Запись закрыта</p>  
 </a>  
 </tr>  
 </th:block>  
 <th:block th:each="travels: ${travelsValid}">  
 <tr>  
 <a th:if="${travels.getRoutes.size < travels.bus.capacity}" th:href="@{'/order/' + ${direction} + ':' + ${dateParsed} + '/' + ${travels.id}}"  
 class="btn btn-lg btn-warning travel\_button px-1">  
 <p>[[${travels.timeParsed}]]</p>  
 <p>Мест осталось: [[${travels.bus.capacity - travels.getRoutes.size}]]</p>  
 </a>  
 <a th:if="${travels.getRoutes.size == travels.bus.capacity}" th:href="@{'/order/' + ${direction} + ':' + ${dateParsed} + '/' + ${travels.id}}"  
 class="btn btn-lg btn-secondary travel\_button disabled">  
 <p>[[${travels.timeParsed}]]</p>  
 <p>Мест нет</p>  
 </a>  
 </tr>  
 </th:block>  
 </thead>  
 </table>  
 </div>  
 </main>  
</div>  
<footer class="footer py-3" th:insert="~{layout/layouts.html :: footer}"></footer>  
</body>  
</html>

* + 1. **order-details.html**

<!DOCTYPE html>  
<!DOCTYPE html>  
<html lang="ru" class="h-100" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
  
 <title>Order details</title>  
  
 <link rel="shortcut icon" href="/bus.svg" type="image/svg">  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
 <link href="/style.css" rel="stylesheet">  
</head>  
  
<body>  
<nav th:insert="~{layout/layouts.html :: navs}"></nav>  
  
<div class="container-fluid text-center m-auto">  
  
 <main>  
 <!-- <h4>[[${travels.driver.firstName}]] [[${travels.driver.lastName}]]</h4>-->  
 <h2>Рейс [[${directionName}]]</h2>  
 <h4>Время рейса: [[${travels.timeToDrive.toString()}]]</h4>  
 <h4 class="mb-5">Номер автобуса: [[${travels.bus.number}]]</h4>  
  
  
 <form th:action="@{'/order/' + ${direction} + ':' + ${dateParsed} + '/' + ${travels.id}}" th:object="${route}"  
 method="post" class="needs-validation m-auto mt-4 col-sm-8 col-lg-5" novalidate>  
 <div class="row g-3 container-fluid">  
 <label class="form-label">Место посадки</label>  
 <select th:field="\*{landingSite}" class="form-select" required>  
 <option value="">Выбор...</option>  
 <th:block th:each="landingSitee: ${landingSites}">  
 <option th:value="${landingSitee.id}">[[${landingSitee.time}]] [[${landingSitee.placeName}]]  
 </option>  
 </th:block>  
 </select>  
 <div class="invalid-feedback">  
 Пожалуйста выберете место посадки.  
 </div>  
  
 <button type="submit" class="btn btn-success">Выбрать</button>  
 </div>  
 </form>  
  
  
 </main>  
</div>  
  
  
<footer class="footer py-3" th:insert="~{layout/layouts.html :: footer}"></footer>  
<script src="/form-validation.js"></script>  
</body>  
</html>

* + 1. **bus-list.html**

<!DOCTYPE html>  
<html lang="ru" class="h-100" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <meta name="description" content="">  
  
 <title>Регистрация</title>  
  
 <link rel="shortcut icon" th:href="@{/bus.svg}" type="image/svg">  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
 <link href="/signin.css" rel="stylesheet">  
 <link href="/style.css" rel="stylesheet">  
</head>  
  
<body class="text-center">  
<nav th:insert="~{layout/layouts.html :: navs\_login}"></nav>  
  
<div class="container text-center m-auto">  
  
 <main>  
 <div class="py-5 text-center">  
 <img class="d-block mx-auto mb-4" src="/bus.svg" alt="" width="72" height="57">  
 <h2>Регистрация</h2>  
 <i class="bi bi-bus-front-fill"></i>  
 </div>  
  
 <div class="row justify-content-center">  
 <div class="col-md-7 col-lg-8">  
 <form th:action="@{/registration}" th:object="${userJPA}" method="post" class="needs-validation"  
 novalidate>  
 <div class="row g-3">  
 <div class="col-sm-6">  
 <label class="form-label">Фамилия</label>  
 <input th:field="\*{lastName}" minlength="1" maxlength="50" type="text" class="form-control"  
 placeholder="Иванов" value="" required>  
 <div class="invalid-feedback">  
 Требуется фамилия  
 </div>  
 </div>  
  
 <div class="col-sm-6">  
 <label class="form-label">Имя</label>  
 <input th:field="\*{firstName}" minlength="1" maxlength="50" type="text" class="form-control"  
 placeholder="Иван" value="" required>  
 <div class="invalid-feedback">  
 Требуется имя  
 </div>  
 </div>  
  
 <div class="col-sm-6">  
 <label class="form-label">Отчество</label>  
 <input th:field="\*{surName}" maxlength="50" type="text" class="form-control"  
 placeholder="Иванович" value="">  
 <!-- <div class="invalid-feedback">-->  
 <!-- Valid first name is required.-->  
 <!-- </div>-->  
 </div>  
  
 <div class="col-sm-6">  
 <label class="form-label">Телефон</label>  
 <input th:field="\*{phone}" minlength="1" maxlength="50" type="tel" class="form-control"  
 placeholder="" value="" required>  
 <div class="invalid-feedback">  
 Требуется телефон  
 </div>  
 </div>  
  
 <div class="col-12">  
 <label class="form-label">Логин</label>  
 <div class="input-group has-validation">  
 <span class="input-group-text">@</span>  
 <input th:field="\*{username}" minlength="1" maxlength="50" type="text"  
 class="form-control" placeholder="" required>  
 <div class="invalid-feedback">  
 Требуется логин  
 </div>  
 </div>  
 </div>  
  
 <div class="col-12">  
 <label class="form-label">Пароль</label>  
 <input th:field="\*{password}" minlength="1" maxlength="50" type="password"  
 class="form-control" placeholder="" required>  
 <div class="invalid-feedback">  
 Требуется пароль  
 </div>  
 </div>  
 </div>  
  
 <hr class="my-4">  
 <button class="w-100 btn btn-primary btn-lg" type="submit">Зарегистрироваться</button>  
 </form>  
 </div>  
 </div>  
 </main>  
  
  
</div>  
<footer class="footer mt-auto py-3" th:insert="~{layout/layouts.html :: footer}"></footer>  
<script src="/form-validation.js"></script>  
</body>  
</html>

* + 1. **drivers-list.html**

<!DOCTYPE html>  
<html lang="ru" class="h-100" xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <meta name="description" content="">  
  
 <title>Drivers List</title>  
  
 <link rel="shortcut icon" th:href="@{bus.svg}" type="image/svg">  
 <link rel="stylesheet" type="text/css" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>  
</head>  
<body class="d-flex flex-column h-100 bg-light">  
<nav th:insert="~{layout/layouts.html :: navs}"></nav>  
  
<div class="container text-center m-auto">  
 <main class="flex-shrink-0">  
  
 <h1>Welcome to Drivers list</h1>  
  
 <div th:if="${message != null}" class="alert alert-success">  
 [[${message}]]  
 </div>  
  
 <div class="table-responsive">  
 <table class="table table-bordered">  
 <thead class="table-dark">  
 <tr>  
 <th>ID</th>  
 <th>Фамилия</th>  
 <th>Имя</th>  
 <th>Поездки</th>  
 <th>All</th>  
 <!-- <th>Actions</th>-->  
 </tr>  
 </thead>  
 <!-- тело таблицы -->  
 <tbody>  
 <!-- цикл -->  
 <th:block th:each="driver: ${driverList}">  
 <tr>  
 <td>[[${driver.id}]]</td>  
 <td>[[${driver.firstName}]]</td>  
 <td>[[${driver.lastName}]]</td>  
 <td>[[${driver.driverTravels}]]</td>  
 <td>[[${driver}]]</td>  
 <!-- <td>-->  
 <!-- &lt;!&ndash; детали &ndash;&gt;-->  
 <!-- <a th:href="@{'/users/details/' + ${user.id}}" class="btn btn-success">-->  
 <!-- Details-->  
 <!-- </a>-->  
 <!-- &lt;!&ndash; редактирование &ndash;&gt;-->  
 <!-- <a th:href="@{'/users/update/' + ${user.id}}" class="btn btn-warning">-->  
 <!-- Update-->  
 <!-- </a>-->  
 <!-- &lt;!&ndash; кнопка удаления &ndash;&gt;-->  
 <!-- <a th:href="@{'/users/delete/' + ${user.id}}" class="btn btn-danger">-->  
 <!-- Delete-->  
 <!-- </a>-->  
 <!-- </td>-->  
 </tr>  
 </th:block>  
 </tbody>  
 </table>  
 </div>  
 <a th:href="@{/drivers/new}" class="header">add new driver</a>  
 </main>  
</div>  
  
<!--<img th:src="@{favicon.svg}" alt="Картинка"/>-->  
  
<footer class="footer mt-auto py-3" th:insert="~{layout/layouts.html :: footer}"></footer>  
</body>  
</html>

1. **База данных**

**Заполнение базы данных**

Подключаемся к базе данных при помощи MySQL Workbench, начнём с заполнения основными данными. Нам необходимо заполнить таблицу ролей, чтобы Spring Security мог с помощью этих данных организовать пользователям с разными ролями доступ в разные места сайта. Заполняем таблицу ролей:

INSERT INTO `tickets`.`rolesjpa` (`name`) VALUES ('ROLE\_ADMIN');  
INSERT INTO `tickets`.`rolesjpa` (`name`) VALUES ('ROLE\_USER');  
INSERT INTO `tickets`.`rolesjpa` (`name`) VALUES ('ROLE\_DISPATCHER');  
INSERT INTO `tickets`.`rolesjpa` (`name`) VALUES ('ROLE\_DRIVER');

Теперь мы можем в файле WebSecurityConfig настройки Spring Security указать кому можно попасть в ту или иную область сайта:

.requestMatchers("/routes/dispatcher/\*\*").hasRole("DISPATCHER ")

Заполним теперь таблицу с автобусами:

INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Пежо Боксер', 'АС 001 16 РУС', '17');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Пежо Боксер', 'АС 002 16 РУС', '17');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Пежо Боксер', 'АС 003 16 РУС', '17');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Пежо Боксер', 'АС 004 16 РУС', '17');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Ситроен Джампер', 'АС 005 16 РУС', '17');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Ситроен Джампер', 'АС 006 16 РУС', '17');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Ситроен Джампер', 'АС 006 16 РУС', '17');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Ситроен Джампер', 'АС 007 16 РУС', '17');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Ютонг', 'АС 008 16 РУС', '39');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Ютонг', 'АС 009 16 РУС', '39');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Ютонг', 'АС 010 16 РУС', '39');  
INSERT INTO `tickets`.`bus\_t` (`bus\_name`, `number`, `capacity`) VALUES ('Ютонг', 'АС 011 16 РУС', '39');

Заполним таблицу с водителями:

INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Иванов', 'Иван', '123', '891791700', 'Иванович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Петров', 'Пётр', '123', '89179170001', 'Петрович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Сидоров', 'Сидр', '123', '89179170002', 'Сидорович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Антонов', 'Антон', '123', '89179170003', 'Антонович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Артёмов', 'Артём', '123', '89179170004', 'Артёмович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Никитин', 'Никита', '123', '89179170005', 'Никитович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Александров', 'Александр', '123', '89179170006', 'Александрович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Владиморов', 'Владимир', '123', '89179170007', 'Владимирович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Русланов', 'Руслан', '123', '89179170008', 'Русланович');  
INSERT INTO `tickets`.`driver\_t` (`first\_name`, `last\_name`, `password`, `phone`, `sur\_name`) VALUES ('Ибрагимов', 'Ибрагим', '123', '89179170009', 'Ибрагимович');

Заполним таблицу остановок:

INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('Автовокзал', '25');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('Кинотеатр \"Мир\"', '10');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('Дом медиков', '-5');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('ПМК', '30');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('ГАИ', '35');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('46 магазин', '30');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('Мебельная фабрика', '20');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('Батыр', '15');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('Дом учителя', '10');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('МЦ', '5');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('Мини маркет', '5');  
INSERT INTO `tickets`.`landing\_site\_t` (`place\_name`, `time`) VALUES ('Стадион', '0');

Заполним таблицу рейсов:

* Направление Чистополь – Казань на 13.01.2024:

INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 05:00', '1', '1', '13.01.2024', '05:00', b'1', '2024-01-13T05:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 06:00', '2', '2', '13.01.2024', '06:00', b'1', '2024-01-13T06:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 07:00', '3', '3', '13.01.2024', '07:00', b'1', '2024-01-13T07:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 08:00', '4', '4', '13.01.2024', '08:00', b'1', '2024-01-13T08:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 09:00', '5', '5', '13.01.2024', '09:00', b'1', '2024-01-13T09:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 10:00', '6', '6', '13.01.2024', '10:00', b'1', '2024-01-13T10:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 11:00', '9', '7', '13.01.2024', '11:00', b'1', '2024-01-13T11:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 12:00', '1', '1', '13.01.2024', '12:00', b'1', '2024-01-13T12:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 13:00', '2', '2', '13.01.2024', '13:00', b'1', '2024-01-13T13:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 14:00', '3', '3', '13.01.2024', '14:00', b'1', '2024-01-13T14:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 15:00', '4', '4', '13.01.2024', '15:00', b'1', '2024-01-13T15:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 16:00', '5', '5', '13.01.2024', '16:00', b'1', '2024-01-13T16:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 17:00', '6', '6', '13.01.2024', '17:00', b'1', '2024-01-13T17:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 18:00', '9', '7', '13.01.2024', '18:00', b'1', '2024-01-13T18:00');

* Направление Казань – Чистополь на 13.01.2024:

INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 07:30', '1', '1', '13.01.2024', '07:30', b'0', '2024-01-13T07:30');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 08:30', '2', '2', '13.01.2024', '08:30', b'0', '2024-01-13T08:30');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 10:00', '3', '3', '13.01.2024', '10:00', b'0', '2024-01-13T10:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 11:00', '4', '4', '13.01.2024', '11:00', b'0', '2024-01-13T11:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 12:00', '5', '5', '13.01.2024', '12:00', b'0', '2024-01-13T12:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 13:00', '6', '6', '13.01.2024', '13:00', b'0', '2024-01-13T13:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 14:00', '9', '7', '13.01.2024', '14:00', b'0', '2024-01-13T14:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 15:00', '1', '1', '13.01.2024', '15:00', b'0', '2024-01-13T15:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 16:00', '2', '2', '13.01.2024', '16:00', b'0', '2024-01-13T16:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 17:00', '3', '3', '13.01.2024', '17:00', b'0', '2024-01-13T17:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 18:00', '4', '4', '13.01.2024', '18:00', b'0', '2024-01-13T18:00');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 18:30', '5', '5', '13.01.2024', '18:30', b'0', '2024-01-13T18:30');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 19:30', '6', '6', '13.01.2024', '19:30', b'0', '2024-01-13T19:30');  
INSERT INTO `tickets`.`travels\_t` (`time\_to\_drive`, `bus\_id`, `driver\_id`, `date\_parsed`, `time\_parsed`, `forward`, `date\_full`) VALUES ('13.01.2024 20:30', '9', '7', '13.01.2024', '20:30', b'0', '2024-01-13T20:30');

**Заключение**

В результате проектирования вэб – приложения был создан сайт по записи пассажиров на автобус. Сайт адаптирован для мобильных устройств и сделан в первую очередь, чтобы им было удобно пользоваться с мобильных устройств. Этим сайтом одновременно могут пользоваться пассажир (записывается на рейсы, просматривает свои поездки, отменяет поездки), диспетчер (записывает пассажиров на рейсы, отменяет записи), водитель (видит свои рейсы, своих пассажиров и откуда их забирать).

Чтобы собрать проект и запустить его как сайт, был использован сборщик Gradle. Собранный проект представляет из себя приложение с расширением .jar. У меня есть виртуальный сервер на <https://cp.beget.com/> и там стоит операционная система Ubuntu 18.04. Там же куплен домен chistaitravel.ru. И чтобы сайт начал работать нужно загрузить на виртуальный сервер это приложение (допустим назовём его tiket.jar) и подключившись к внешнему терминалу запустить это приложение командой: screen java -jar tiket.jar.

В дальнейшем планирую улучшать это вэб – приложение.

Ссылка на сайт: <https://chistaitravel.ru/>

Ссылка на github: <https://github.com/JohnHozin/diplom_tiket>

**Список литературы и ресурсов**

1. <https://www.youtube.com/watch?v=szBEzz9x2cU&list=PL5uW85PisjPOnQuAe0yxw6ooZUpXgN94-&index=80>
2. <https://www.youtube.com/watch?v=aHDWra0X7vI&list=PLoq3Accf02PVdUqjqPdWMG4HbEZXlhICW&index=8>
3. <https://tproger.ru/articles/pishem-java-veb-prilozhenie-na-sovremennom-steke-s-nulja-do-mikroservisnoj-arhitektury-chast-1>
4. <https://stackoverflow.com/>
5. <https://ru.wikipedia.org/>
6. <https://mvnrepository.com/>
7. <https://habr.com/>
8. <https://azure.microsoft.com/ru-ru/>
9. <https://www.nic.ru/>
10. <https://javarush.com/>
11. <https://github.com/>
12. <https://cp.beget.com/>