«Журнал работы преподавателя»

Функциональные требования:

многопользовательский режим

журнал действий пользователя

журнал работы приложения

мониторинг доступности

резервное копирование БД (по расписанию, по запросу пользователя)

● реализация функций

○ создание обзор редактирование плана и фактического выполнения

○ создание обзор редактирование плана и фактического выполнения занятия преподавателем. Занятие состоит из: опроса ячеек данных, лекции, лабораторной, докладов, ответов на вопросы

○ создание обзор редактирование плана и фактического наполнения лекций, разделов материала и лабораторных работ, примеров заданий

○ создание обзор и редактирование сводки посещаемости, успеваемости ячеек данных

○ создание обзор и редактирование сводки плана, факта курса и отдельных занятий

○ создание обзор редактирование материалов контрольных мероприятий, вопросы и задания, компоновка билетов

Доп. требования:

Использование открытого репозитория.

Использование открытого таск-трекера.

Документирование проекта (как для ВКР) и документирование кода.

Использование скрама для организации работ.

Отчеты на лабораторных занятиях.

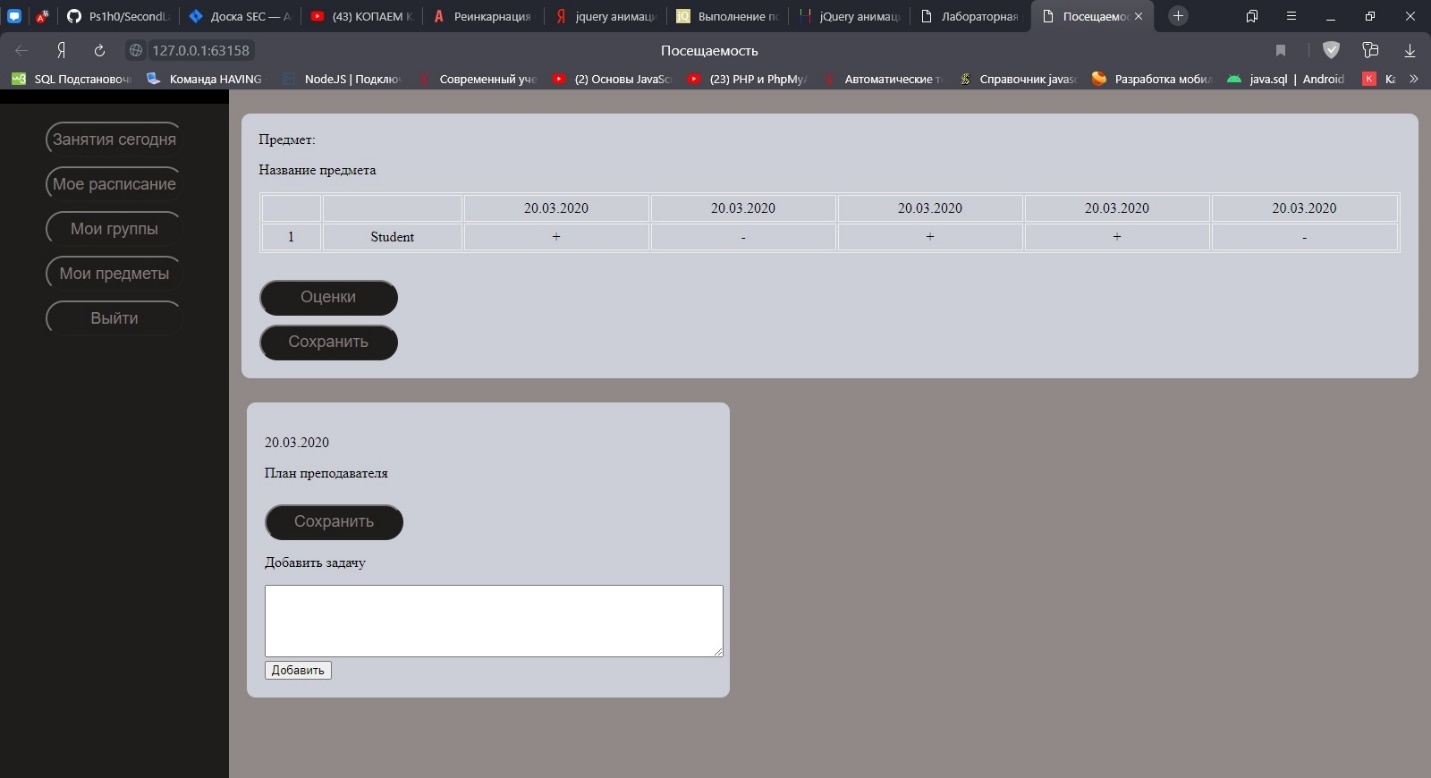
Требования безопасности:

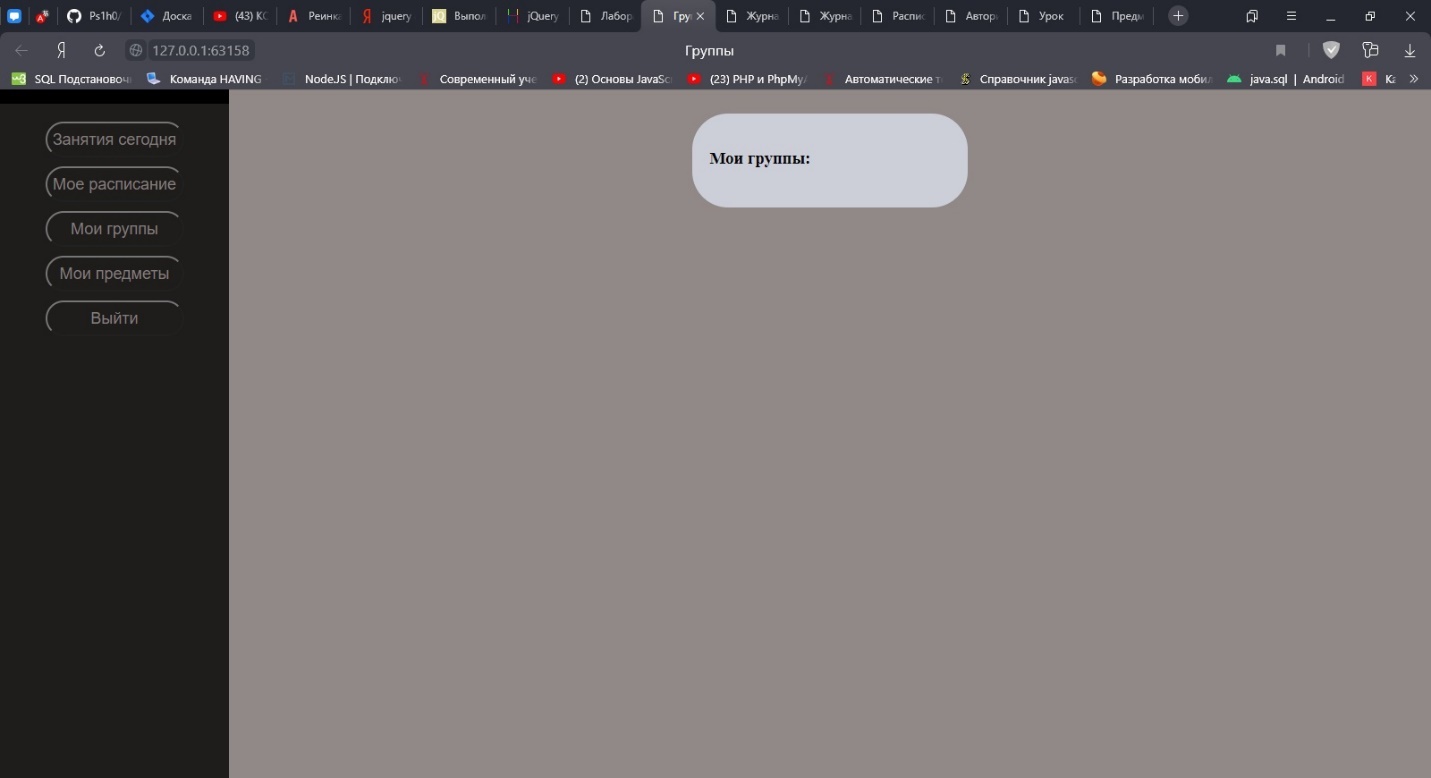
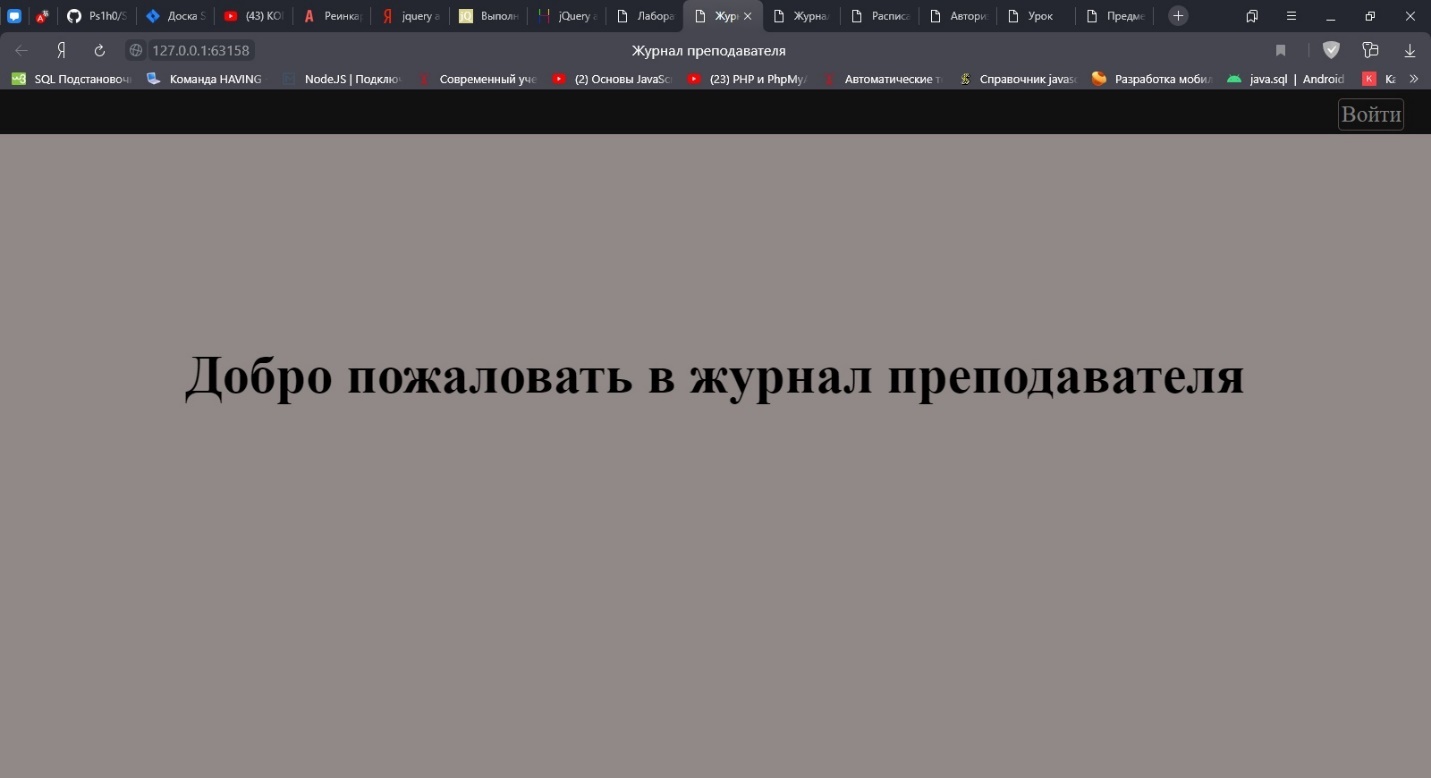
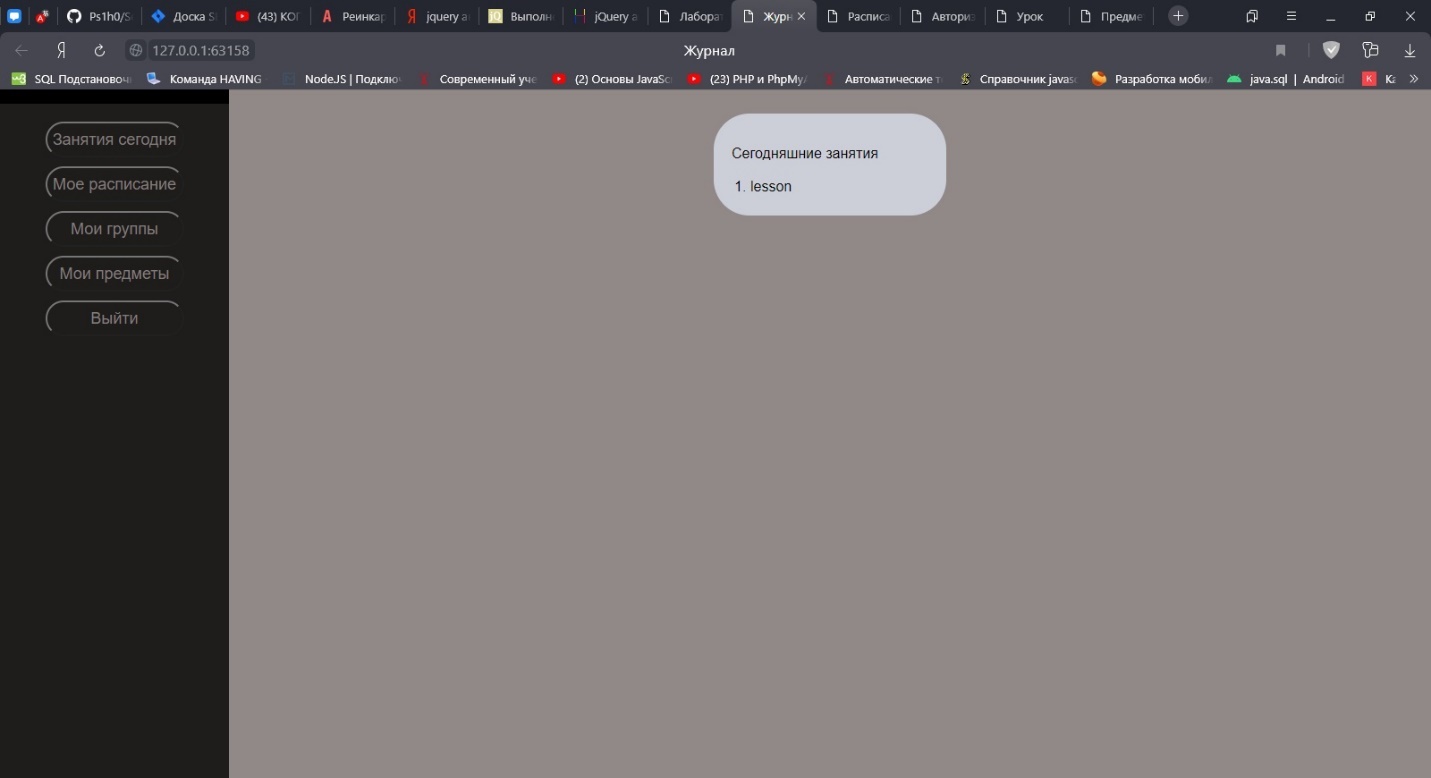
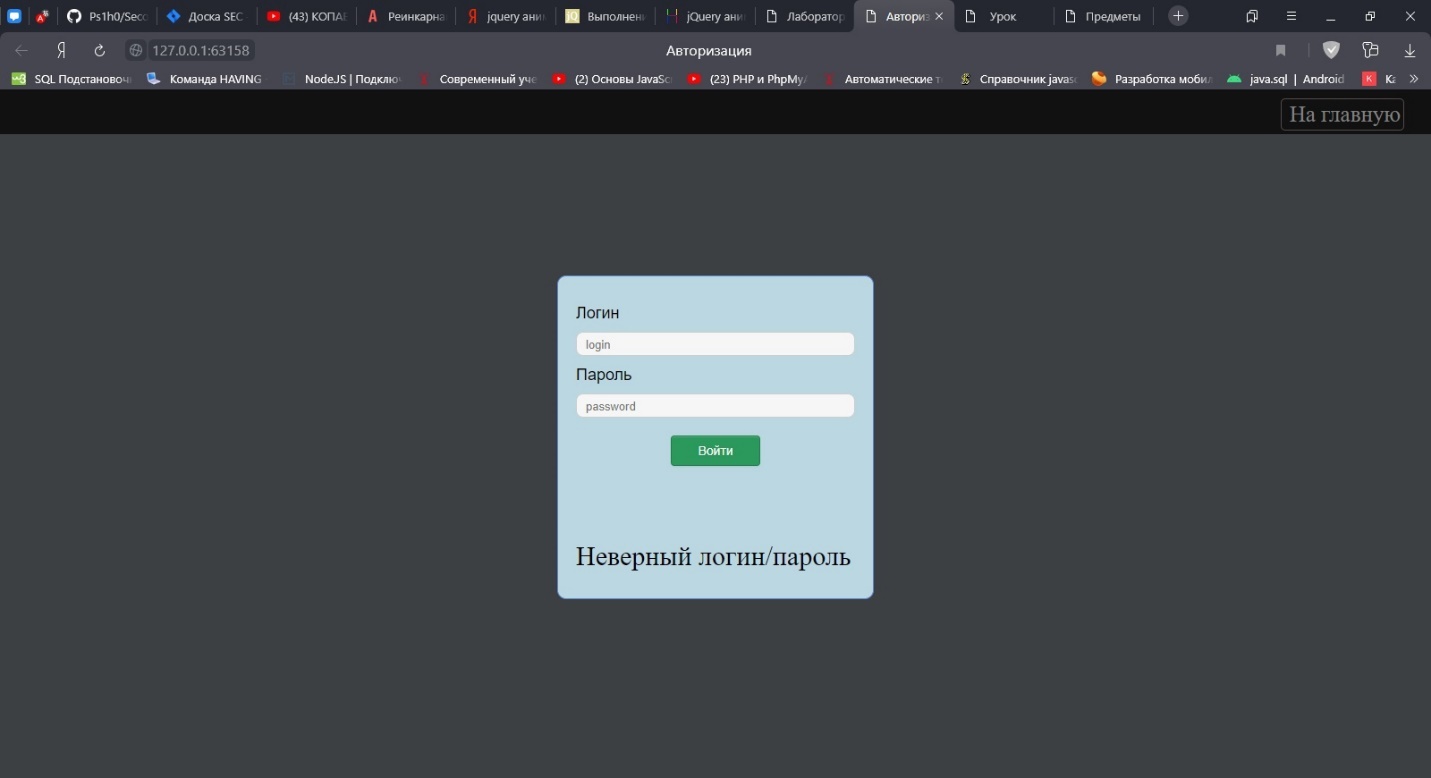
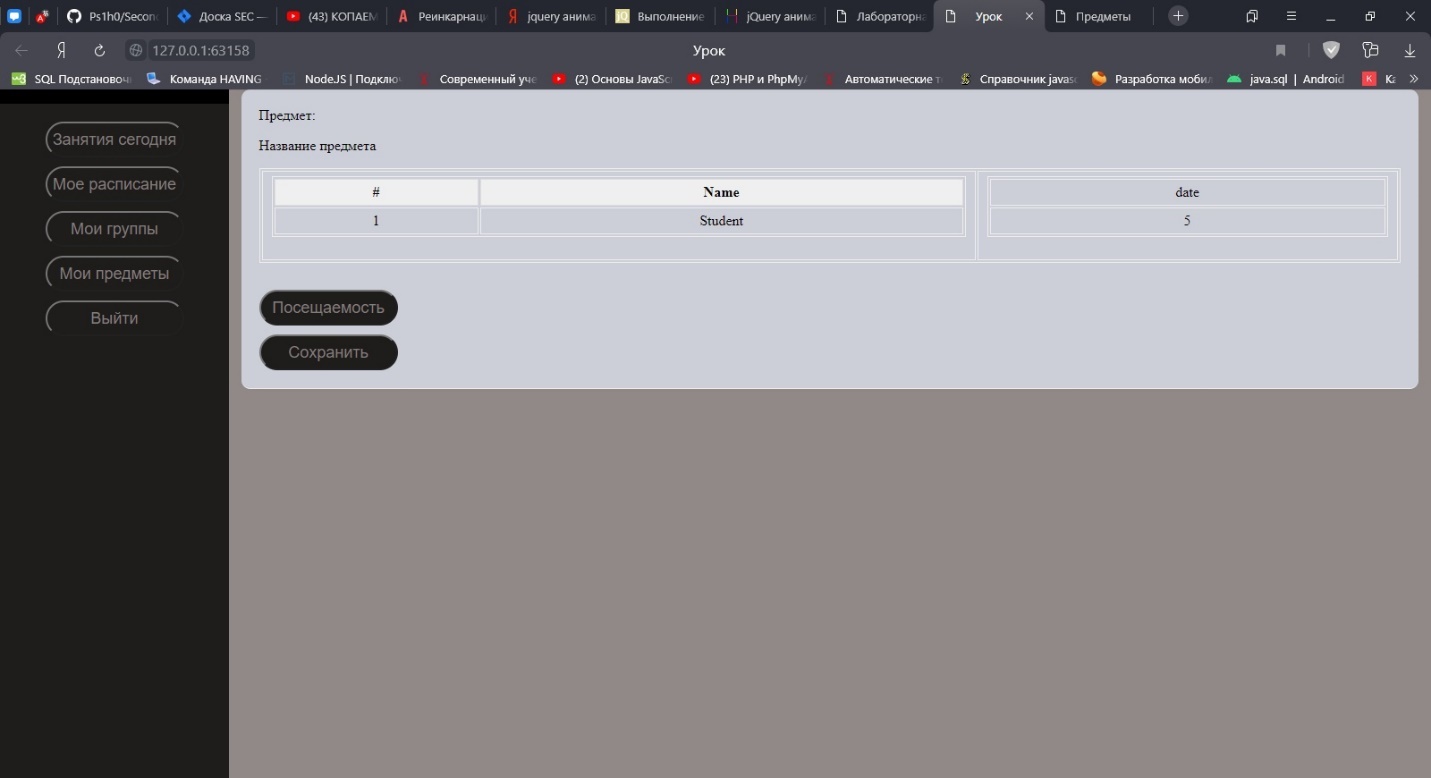
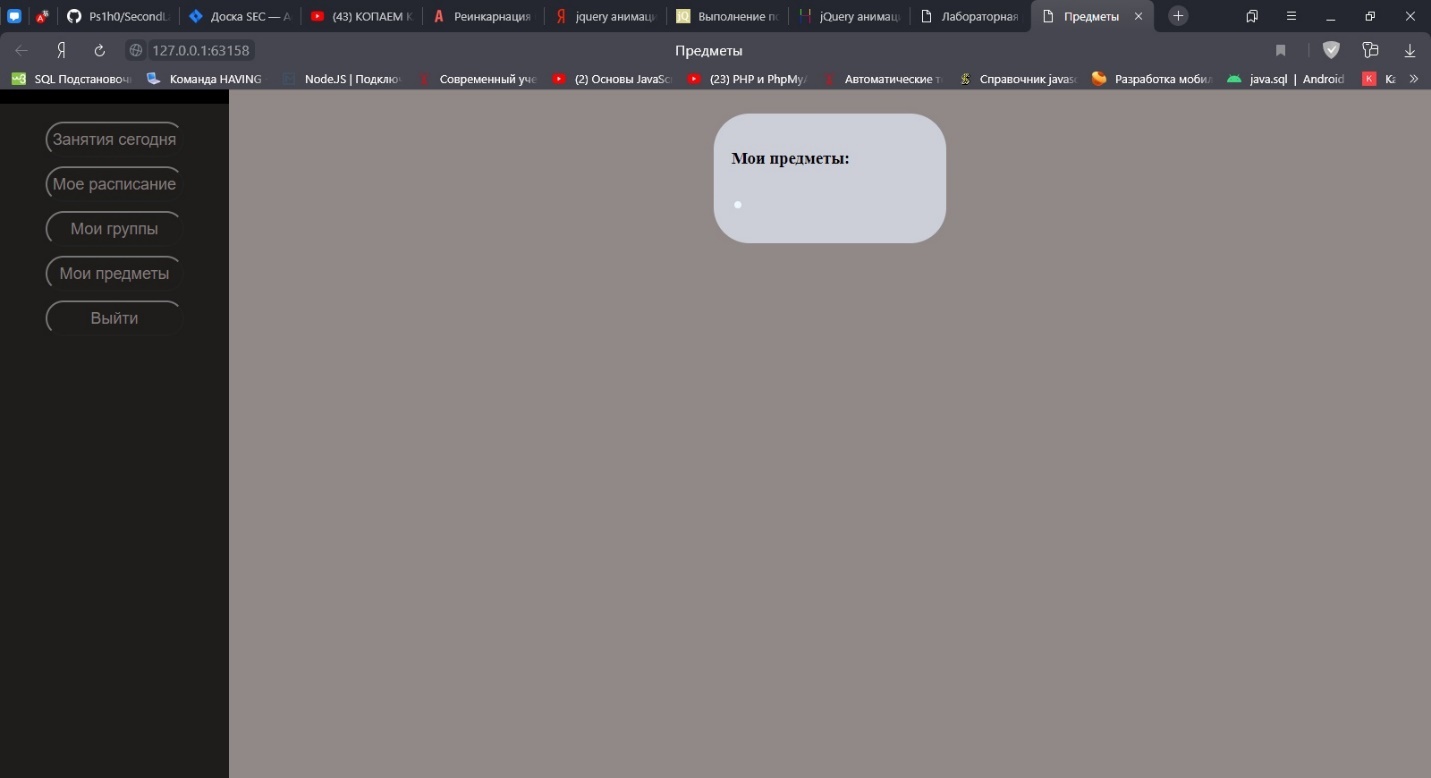
Программа должна стабильно работать, не предоставлять данные посторонним лицам, иметь систему авторизации и разграничение уровней доступа .

Требования к интерфейсу:

Не предъявлялись.

Иллюстрация интерфейса:

 Рисунок 1 – Иллюстрация интерфейса

Рисунок 2 – Иллюстрация интерфейса  Рисунок 3 – Иллюстрация интерфейса  Рисунок 4 – Иллюстрация интерфейса  Рисунок 5 – Иллюстрация интерфейса  Рисунок 6 – Иллюстрация интерфейса  Рисунок 7 – Иллюстрация интерфейса  Рисунок 8 – Иллюстрация интерфейса

Требования к производительности:

Программа должна стабильно работать с большим количеством записей (до 10 000 записей примерно за 30 сек).

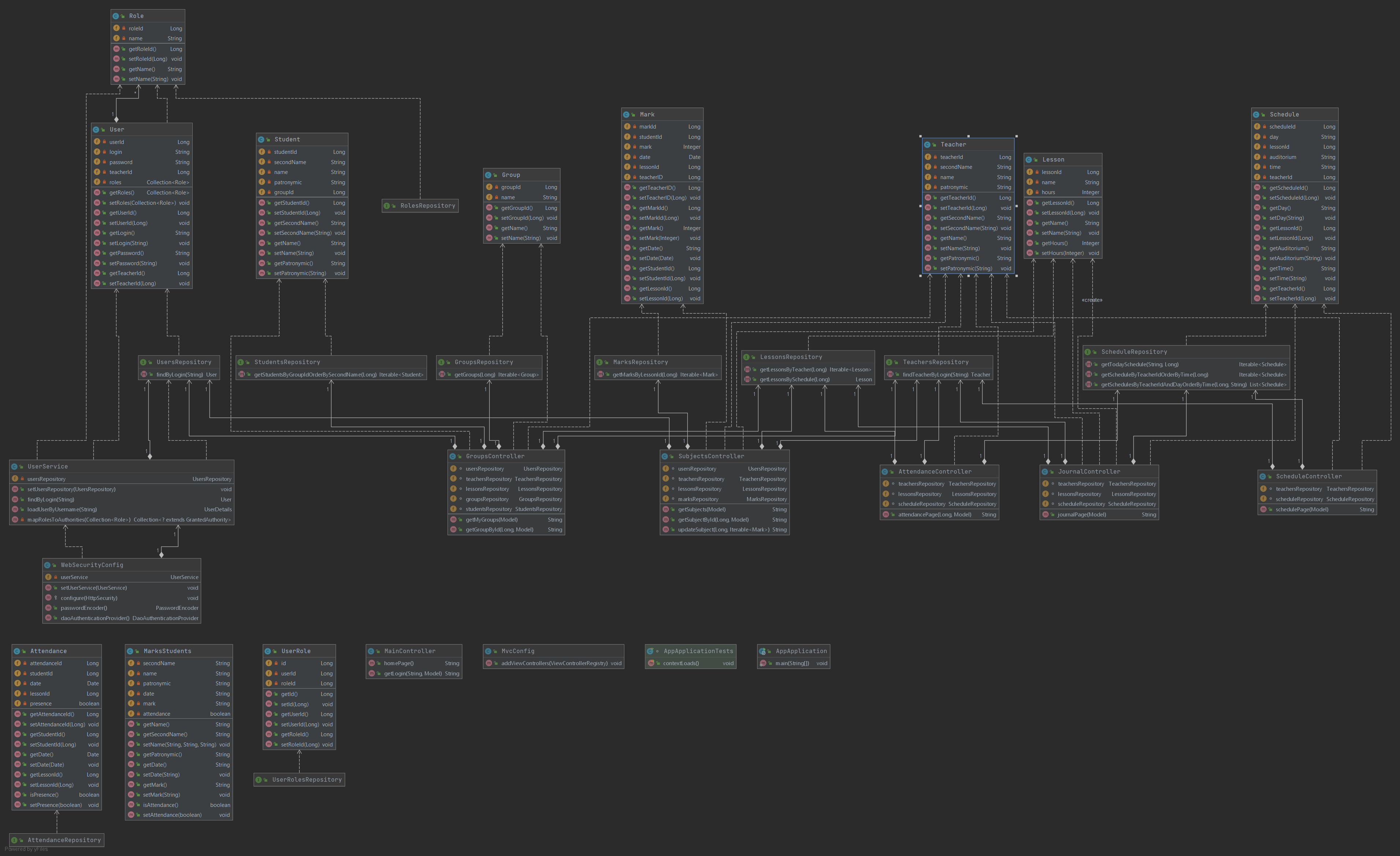


Рисунок 9 – Диаграмма классов

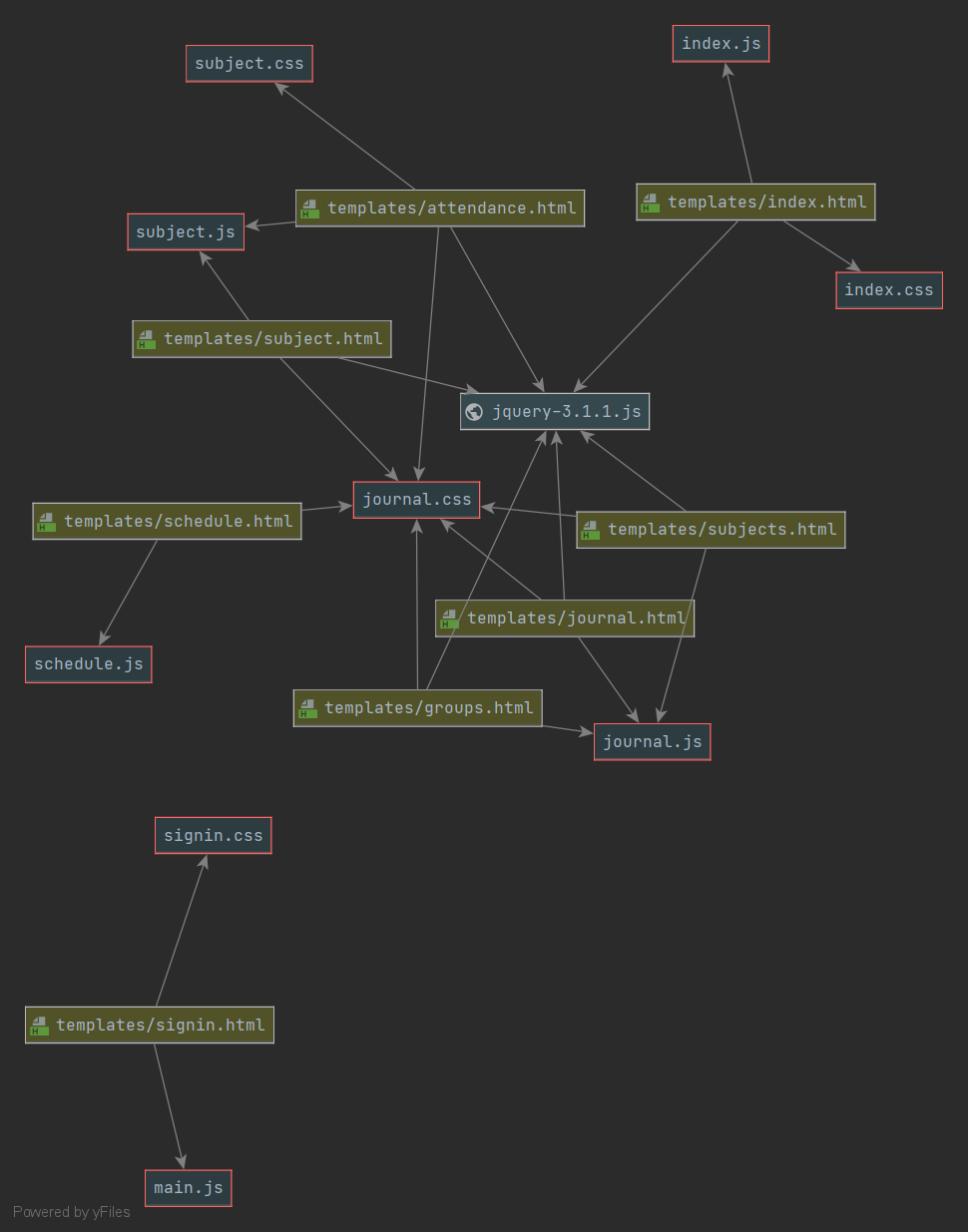


Рисунок 10 – UML Диаграмма

Конструирование:

Использованный Фреймворк и библиотеки:

* import org.springframework.boot.SpringApplication
  + public class SpringApplication
  + extends Object
  + Classes that can be used to bootstrap and launch a Spring application from a Java main method. By default class will perform the following steps to bootstrap your application:
  + Create an appropriate ApplicationContext instance (depending on your classpath)
  + Register a CommandLinePropertySource to expose command line arguments as Spring properties
  + Refresh the application context, loading all singleton beans
  + Trigger any CommandLineRunner beans
  + In most circumstances the static run(Object, String[]) method can be called directly from your main method to bootstrap your application:
* import org.springframework.boot.autoconfigure.SpringBootApplication
  + public interface SpringBootApplication
  + extends annotation.Annotation
* import org.springframework.security.core.GrantedAuthority;
  + public interface GrantedAuthority
  + extends Serializable
  + Represents an authority granted to an Authentication object.
  + A GrantedAuthority must either represent itself as a String or be specifically supported by an AccessDecisionManager.
* import org.springframework.security.core.authority.SimpleGrantedAuthority;
  + public final class SimpleGrantedAuthority
  + extends Object
  + implements GrantedAuthority
  + Basic concrete implementation of a GrantedAuthority.
  + Stores a String representation of an authority granted to the Authentication object.
* import org.springframework.security.core.userdetails.UserDetails;
  + public interface UserDetails
  + extends Serializable
  + Provides core user information.
  + Implementations are not used directly by Spring Security for security purposes. They simply store user information which is later encapsulated into Authentication objects. This allows non-security related user information (such as email addresses, telephone numbers etc) to be stored in a convenient location.
* import org.springframework.security.core.userdetails.UserDetailsService;
  + public interface UserDetailsService
  + Core interface which loads user-specific data.
  + It is used throughout the framework as a user DAO and is the strategy used by the DaoAuthenticationProvider.
  + The interface requires only one read-only method, which simplifies support for new data-access strategies.
* import org.springframework.security.core.userdetails.UsernameNotFoundException;
  + public class UsernameNotFoundException
  + extends AuthenticationException
  + Thrown if an UserDetailsService implementation cannot locate a User by its username.
* import org.springframework.stereotype.Service;
  + Annotations denoting the roles of types or methods in the overall architecture (at a conceptual, rather than implementation, level).
  + Intended for use by tools and aspects (making an ideal target for pointcuts).
* import javax.transaction.Transactional;
  + public @interface Transactional
  + The javax.transaction.Transactional annotation provides the application the ability to declaratively control transaction boundaries on CDI managed beans, as well as classes defined as managed beans by the Java EE specification, at both the class and method level where method level annotations override those at the class level.
* import java.util.Collection;
  + public interface Collection<E>
  + extends Iterable<E>
  + The root interface in the collection hierarchy. A collection represents a group of objects, known as its elements. Some collections allow duplicate elements and others do not. Some are ordered and others unordered. The JDK does not provide any direct implementations of this interface: it provides implementations of more specific subinterfaces like Set and List. This interface is typically used to pass collections around and manipulate them where maximum generality is desired.
* import java.util.stream.Collectors;
  + public final class Collectors
  + extends Object
  + Implementations of Collector that implement various useful reduction operations, such as accumulating elements into collections, summarizing elements according to various criteria, etc.
* import org.springframework.data.jpa.repository.JpaRepository;
  + public interface JpaRepository<T,ID>
  + extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>
  + JPA specific extension of Repository.
* import org.springframework.stereotype.Repository;
  + public @interface Repository
  + Indicates that an annotated class is a "Repository", originally defined by Domain-Driven Design (Evans, 2003) as "a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects".
* import javax.persistence.\*;
  + Java Persistence is the API for the management for persistence and object/relational mapping.
* import java.sql.Time;
  + public class Time
  + extends Date
  + A thin wrapper around the java.util.Date class that allows the JDBC API to identify this as an SQL TIME value. The Time class adds formatting and parsing operations to support the JDBC escape syntax for time values.
* import java.util.Date;
  + public class Date
  + extends Object
  + implements Serializable, Cloneable, Comparable<Date>
  + The class Date represents a specific instant in time, with millisecond precision.
* org.springframework.security.core.context.SecurityContextHolder;
  + ublic class SecurityContextHolder
  + extends Object
  + Associates a given SecurityContext with the current execution thread.
* import org.springframework.stereotype.Controller;
  + public @interface
  + Indicates that an annotated class is a "Controller" (e.g. a web controller).
  + This annotation serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning. It is typically used in combination with annotated handler methods based on the RequestMapping annotation.
* import org.springframework.ui.Model;
  + public interface Model
  + Java-5-specific interface that defines a holder for model attributes. Primarily designed for adding attributes to the model. Allows for accessing the overall model as a java.util.Map.
* import org.springframework.web.bind.annotation.PathVariable;
  + public @interface PathVariable
  + Annotation which indicates that a method parameter should be bound to a URI template variable. Supported for RequestMapping annotated handler methods.
  + If the method parameter is Map<String, String> then the map is populated with all path variable names and values.
* import org.springframework.web.bind.annotation.RequestMapping;
  + public @interface RequestMapping
  + Annotation for mapping web requests onto methods in request-handling classes with flexible method signatures.
* import org.springframework.web.bind.annotation.RequestParam;
  + public @interface RequestParam
  + Annotation which indicates that a method parameter should be bound to a web request parameter.
* import org.springframework.beans.factory.annotation.Autowired;
  + public @interface Autowired
  + Marks a constructor, field, setter method, or config method as to be autowired by Spring's dependency injection facilities. This is an alternative to the JSR-330 Inject annotation, adding required-vs-optional semantics.
* import org.springframework.security.core.Authentication;
  + public interface Authentication
  + extends Principal, Serializable
  + Represents the token for an authentication request or for an authenticated principal once the request has been processed by the AuthenticationManager.authenticate(Authentication) method.
* import org.springframework.context.annotation.Bean;
  + public @interface Bean
  + Indicates that a method produces a bean to be managed by the Spring container.
* import org.springframework.context.annotation.Configuration;
  + public @interface Configuration
  + Indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime, for example
* import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
  + public class DaoAuthenticationProvider
  + extends AbstractUserDetailsAuthenticationProvider
  + AuthenticationProviderAn implementation that retrieves user details from a UserDetailsService.
* import org.springframework.security.config.annotation.web.builders.HttpSecurity;
  + public final class HttpSecurity
  + extends AbstractConfiguredSecurityBuilder<DefaultSecurityFilterChain,HttpSecurity>
  + implements SecurityBuilder<DefaultSecurityFilterChain>, HttpSecurityBuilder<HttpSecurity>
  + A HttpSecurity is similar to Spring Security's XML <http> element in the namespace configuration. It allows configuring web based security for specific http requests. By default it will be applied to all requests, but can be restricted using requestMatcher(RequestMatcher) or other similar methods.
* import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
  + public @interface EnableWebSecurity
  + Add this annotation to an @Configuration class to have the Spring Security configuration defined in any WebSecurityConfigurer or more likely by extending the WebSecurityConfigurerAdapter base class and overriding individual methods
* import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
* import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
* import org.springframework.security.crypto.password.PasswordEncoder;
* import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
* import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

Описание методов, классов и интерфейсов:

* public class AppApplication - Точка выхода приложения
* public class UserService - Класс сервис пользователя приложения
  + public void setUsersRepository(UsersRepository usersRepository) - Метод поиска пользователя в базе данных
  + public User findByLogin(String login) - Метод загружает пользователя из базе данных по логину
  + public UserDetails loadUserByUsername(String login) - Метод для присвоения роли пользователю
* public interface AttendanceRepository extends JpaRepository<Attendance, Long> - Интерфейс для запросов в таблицу "посещаемость" базы данных
* public interface GroupsRepository extends JpaRepository<Group, Long> - Интерфейс для запросов в таблицу "группы" базы данных
* public interface LessonsRepository extends JpaRepository<Lesson, Long> - Интерфейс для запросов в таблицу "уроки" базы данных
* public interface MarksRepository extends JpaRepository<Mark, Long> - Интерфейс для запросов в таблицу "оценки" базы данных
* public interface RolesRepository extends JpaRepository<Role, Long> - Интерфейс для запросов в таблицу "роли" базы данных
* public interface ScheduleRepository extends JpaRepository<Schedule, Long> - Интерфейс для запросов в таблицу "расписание" базы данных
* public interface StudentsRepository extends JpaRepository<Student, Long> - Интерфейс для запросов в таблицу "студенты" базы данных
* public interface TeachersRepository extends JpaRepository<Teacher, Long> - Интерфейс для запросов в таблицу "учителя" базы данных
* public interface UserRolesRepository extends JpaRepository<UserRole, Long> - Интерфейс для запросов в таблицу "роли пользователей" базы данных
* public interface UsersRepository extends JpaRepository<User, Long> - Интерфейс для запросов в таблицу "пользователи" базы данных
* public class Attendance - Класс, соответствующий таблице "посещаемость" в базе данных, предназначен для хранения данных
* public class Group - Класс, соответствующий таблице "группы" в базе данных, предназначен для хранения данных
* public class Lesson - Класс, соответствующий таблице "уроки" в базе данных, предназначен для хранения данных
* public class Mark - Класс, соответствующий таблице "оценки" в базе данных, предназначен для хранения данных
* public class Role - Класс, соответствующий таблице "роли" в базе данных, предназначен для хранения данных
* public class Schedule - Класс, соответствующий таблице "расписание" в базе данных, предназначен для хранения данных
* public class Student - Класс, соответствующий таблице "студенты" в базе данных, предназначен для хранения данных
* public class Teacher - Класс, соответствующий таблице "учителя" в базе данных, предназначен для хранения данных
* public class User - Класс, соответствующий таблице "пользователи" в базе данных, предназначен для хранения данных
* public class UserRole – Класс, соответствующий таблице "роли пользователей" в базе данных, предназначен для хранения данных
* public class GroupsController – Класс, контроллер групп. Определяет переходы между страницами групп и дочерними страницами
* public class JournalController - Класс контроллер журнала преподавателя. Определены переходы на страницу журнала и дочерние страницы
  + public String journalPage(Model model) - Метод перехода на страницу журнала преподавателя после аутентификации пользователя
* public class MainController - Класс контроллер главной страницы приложения. Определен переход к аутентификации
  + public String homePage() - Метод перехода на главную страницу приложения
  + public String getLogin(@RequestParam(value = "error", required = false) String error, Model model) - Метод перехода на страницу аутентификации пользователя.
* public class SubjectsController
  + public String getSubjects(Model model) - Метод перехода на страницу предметов преподавателя
  + public String getSubjectById(@PathVariable("id") Long id, Model model) – Метод перехода к предмету преподавателя из списка.
* public class MvcConfig - Класс конфигурация приложения
  + public void addViewControllers(ViewControllerRegistry registry) - Метод, определяющий переход к страницам по адресу
* public class WebSecurityConfig - Класс конфигурация безопасности приложения
  + public void setUserService(UserService userService)
  + protected void configure (HttpSecurity http) - Метод, определяющий, по каким адресам могут переходить авторизованные и неавторизованные пользователи, переадресации страницы
  + public PasswordEncoder passwordEncoder() - Метод, возвращающий декодировщик пароля пользователя
  + public DaoAuthenticationProvider daoAuthenticationProvider() - Метод, позволяющий проводить аутентификацию пользователей из базы данных с помощью декодировщика пароля, установленного методом {@link #passwordEncoder()} и параметров пользователя, установленных методом {@link #setUserService(UserService)}

Нагрузочное тестирование для всей системы:

Файл, содержащий 1000 ячеек данных, обрабатывается 300 – 400 миллисекунд



Рисунок 2 – Работа с файлом в 1000 ячеек данных

Файл, содержащий 10000 ячеек данных, обрабатывается в среднем 3500 миллисекунд



Рисунок 3 – Работа с файлом в 10000 ячеек данных

Файл, содержащий 20000 ячеек данных, обрабатывается в среднем 11000 - 13000 миллисекунд



Рисунок 4 – Работа с файлом в 20000 ячеек данных

Для работы системы с приемлемым временем рекомендуется работать с файлами в 10000 – 20000 ячеек данных.

Предельная нагрузка – 10 т. ячеек данных. Ожидается, что система будет работать с 2-5 т. ячеек данных, не вызывая чрезмерных нагрузок на сервер.

При разработке данного продукта была использована СКВ github. Ссылка на github: <https://github.com/Ps1h0/SecondLab.git>