**ФЕДЕРАЛЬОНЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗАВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**“НАЦИОНАЛЬНЫЙ ИСЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”**

Факультет Программной Инженерии и Компьютерной Техники

Дисциплина: «Программирование»

**ОТЧЁТ**

по лабораторной работе №8  
Вариант №591014

Выполнил:

Студент группы P3111

Дорохин Сергей Константинович

Проверил:

Бойко Владислав Алексеевич

Санкт-Петербург

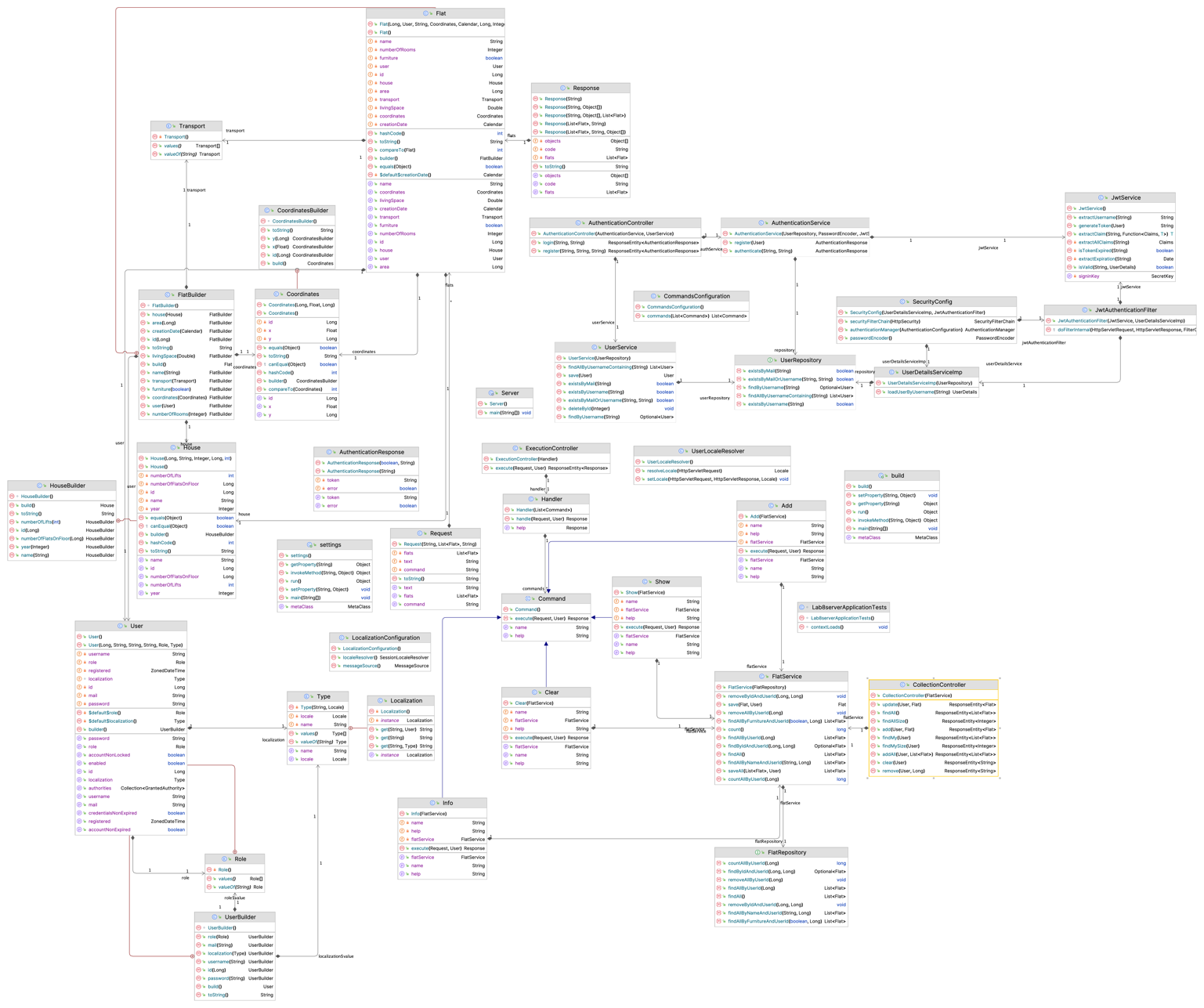
2024 г.

**Задание**

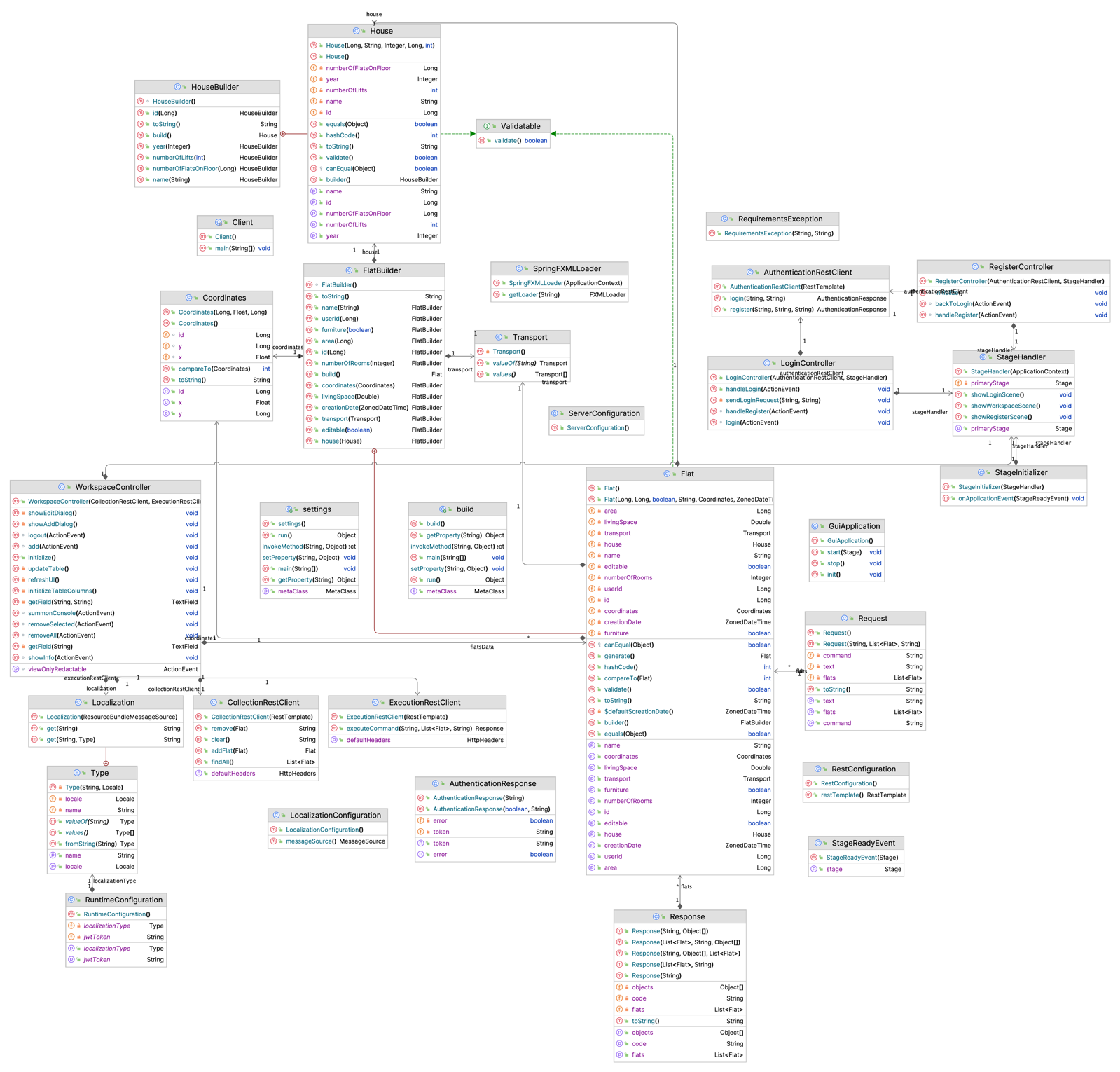
[**Вариант 591014**](https://github.com/serezk4/ITMO.labs/blob/master/programming/lab_5_6/LAB7.md)

**Код Программы**

**Диаграмма (сервер)**

****

**Диаграмма (клиент)**

****

**Localization.java**package com.serezka.localization;  
  
import com.serezka.configuration.RuntimeConfiguration;  
import lombok.AccessLevel;  
import lombok.Getter;  
import lombok.RequiredArgsConstructor;  
import lombok.experimental.FieldDefaults;  
import lombok.extern.log4j.Log4j2;  
import org.springframework.context.NoSuchMessageException;  
import org.springframework.context.annotation.Scope;  
import org.springframework.context.support.ResourceBundleMessageSource;  
import org.springframework.stereotype.Component;  
  
import java.util.Locale;  
  
*/\*\*  
 \* Class for localization  
 \* Allows to get localized messages  
 \* @version 1.0  
 \*/*@Component @Scope("singleton")  
@FieldDefaults(level = AccessLevel.*PRIVATE*, makeFinal = true)  
@RequiredArgsConstructor  
@Log4j2  
public class Localization {  
 @FieldDefaults(level = AccessLevel.*PRIVATE*, makeFinal = true)  
 @RequiredArgsConstructor @Getter  
 public enum Type {  
 *RU*("RU", Locale.*of*("ru")),  
 *US*("EN", Locale.*of*("us")),  
 *FR*("FR", Locale.*of*("fr")),  
 *NO*("NO", Locale.*of*("no"));  
  
 String name;  
 Locale locale;  
  
 public static final Type *DEFAULT* = Type.*RU*;  
  
 public static Type fromString(String name) {  
 for (Type type : Type.*values*())  
 if (type.getName().equalsIgnoreCase(name)) return type;  
 return *DEFAULT*;  
 }  
 }  
  
 ResourceBundleMessageSource messageSource;  
  
 */\*\*  
 \* Get localized message  
 \* @param code - message code  
 \* @param localization - localization type  
 \* @return localized message  
 \*/* public String get(String code, Type localization) {  
 if (localization == null) return get(code);  
  
 try {  
 return messageSource.getMessage(code, null, localization.getLocale());  
 } catch (NoSuchMessageException e) {  
 *log*.warn(e.getMessage());  
 return code;  
 }  
 }  
  
 */\*\*  
 \* Get localized message  
 \* @param code - message code  
 \* @return localized message with default locale  
 \*/* public String get(String code) {  
 try {  
 return messageSource.getMessage(code, new Object[0], RuntimeConfiguration.*getLocalizationType*().getLocale());  
 } catch (NoSuchMessageException e) {  
 *log*.warn(e.getMessage());  
 return code;  
 }  
 }  
}

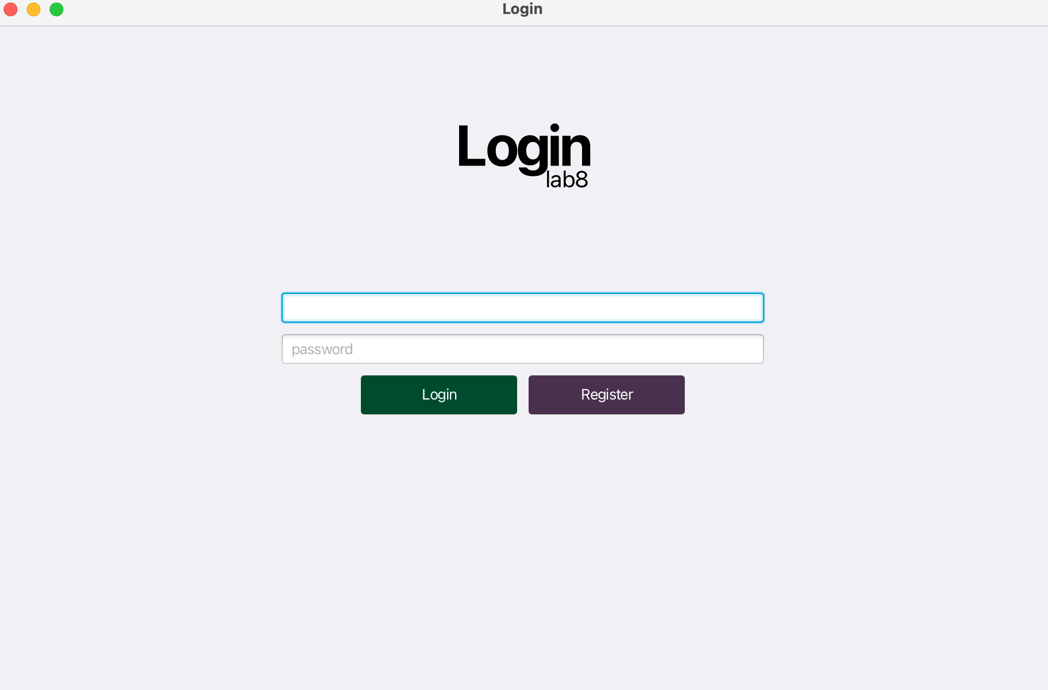
**LoginController.java**import com.serezka.configuration.RuntimeConfiguration;  
import com.serezka.gui.stage.StageHandler;  
import com.serezka.net.authorization.AuthenticationResponse;  
import com.serezka.net.authorization.AuthenticationRestClient;  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.PasswordField;  
import javafx.scene.control.TextField;  
import javafx.scene.text.Text;  
import lombok.AccessLevel;  
import lombok.RequiredArgsConstructor;  
import lombok.SneakyThrows;  
import lombok.experimental.FieldDefaults;  
import org.springframework.stereotype.Component;  
  
@Component  
@RequiredArgsConstructor  
@FieldDefaults(level = AccessLevel.*PRIVATE*)  
public class LoginController {  
 private final AuthenticationRestClient authenticationRestClient;  
 private final StageHandler stageHandler;  
  
 @FXML Button authorize;  
 @FXML Text errorText;  
 @FXML TextField login;  
 @FXML PasswordField password;  
 @FXML Button register;  
  
 @FXML  
 void login(ActionEvent event) {  
 sendLoginRequest(login.getText(), password.getText());  
 }  
  
 @SneakyThrows  
 private void sendLoginRequest(String username, String password) {  
 try {  
 AuthenticationResponse authenticationResponse = authenticationRestClient.login(username, password);  
  
 if (authenticationResponse.isError()) {  
 errorText.setText(authenticationResponse.getToken());  
 return;  
 }  
  
 RuntimeConfiguration.*setJwtToken*(authenticationResponse.getToken());  
 errorText.setText("");  
  
 stageHandler.showWorkspaceScene();  
 } catch (Exception e) {  
 errorText.setText("Server is not available");  
 }  
 }  
  
 public void handleLogin(ActionEvent actionEvent) {  
   
 }  
  
 @FXML  
 void handleRegister(ActionEvent event) {  
 try {  
 stageHandler.showRegisterScene();  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

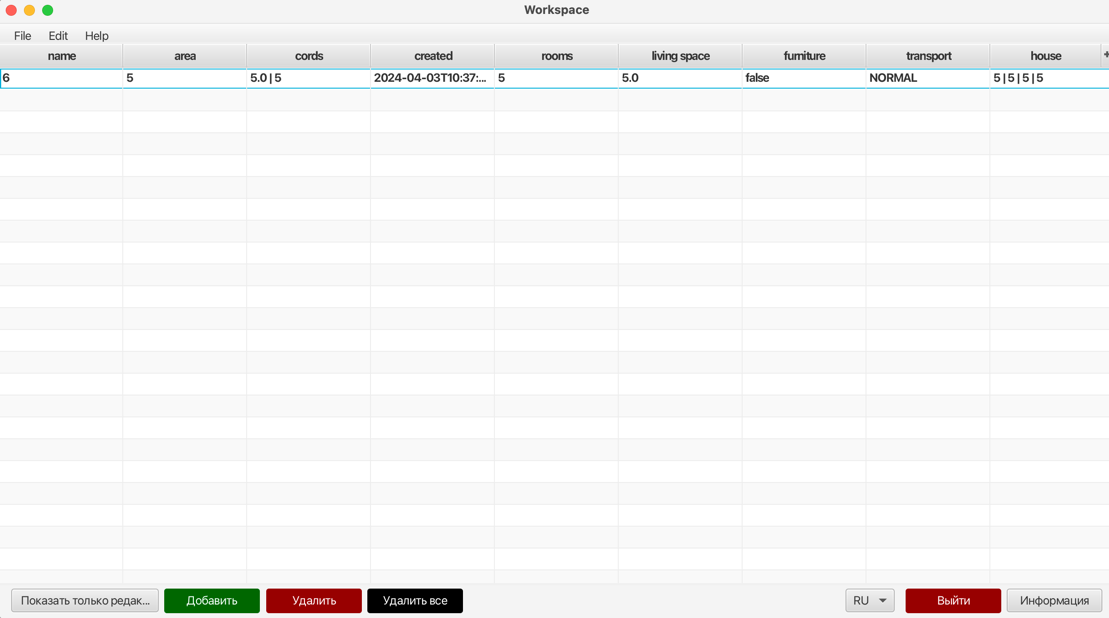
**CollectionController.java**

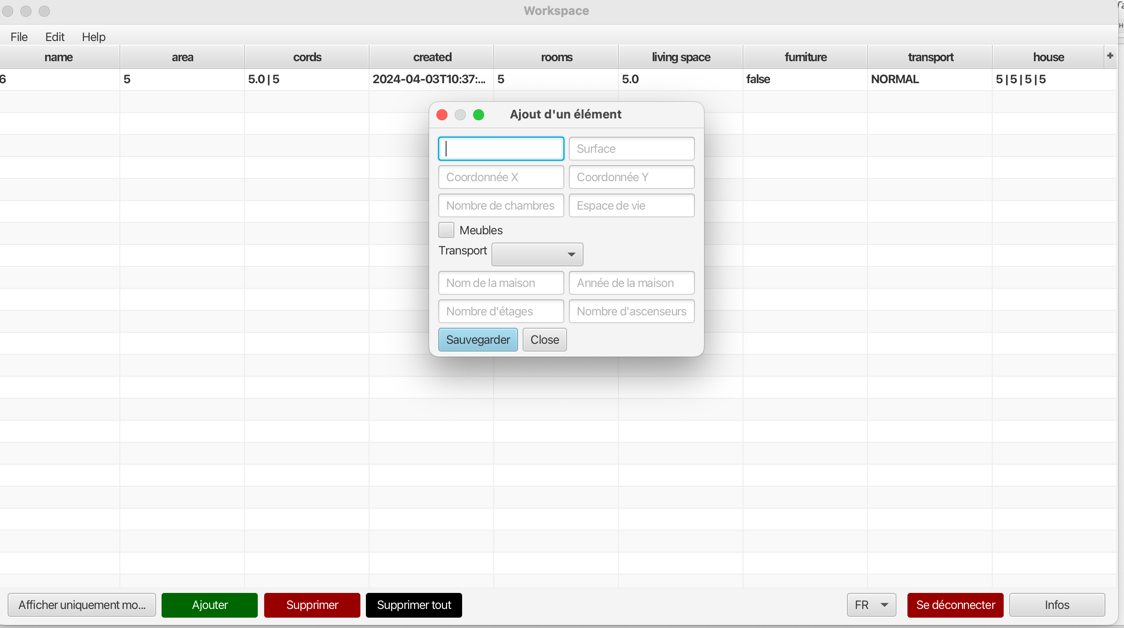
package com.serezka.server.collection.controller;  
  
import com.serezka.server.authorization.database.model.User;  
import com.serezka.server.collection.database.model.Flat;  
import com.serezka.server.collection.database.service.FlatService;  
import lombok.AccessLevel;  
import lombok.RequiredArgsConstructor;  
import lombok.experimental.FieldDefaults;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.core.annotation.AuthenticationPrincipal;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
*// filter chain /collection/\*\* and /execute/\*\**@RequestMapping("/collection")  
@RestController  
@FieldDefaults(level = AccessLevel.*PRIVATE*, makeFinal = true)  
@RequiredArgsConstructor  
public class CollectionController {  
 FlatService flatService;  
  
 @GetMapping("/all")  
 public ResponseEntity<List<Flat>> findAll(@AuthenticationPrincipal User user) {  
 return ResponseEntity.*ok*(flatService.findAll().stream()  
 .peek(flat -> flat.setEditable(flat.getUser().equals(user))).toList());  
 }  
  
 @GetMapping("/size")  
 public ResponseEntity<Integer> findAllSize() {  
 return ResponseEntity.*ok*(flatService.findAll().size());  
 }  
  
 @PostMapping("/clear")  
 public ResponseEntity<String> clear(@AuthenticationPrincipal User user) {  
 flatService.removeAllByUserId(user.getId());  
 return ResponseEntity.*ok*("Collection is cleared");  
 }  
  
 @PostMapping("/add")  
 public ResponseEntity<Flat> add(@AuthenticationPrincipal User user,  
 @RequestBody Flat flat) {  
 return ResponseEntity.*ok*(flatService.save(flat, user));  
 }  
  
 @PostMapping("/addAll")  
 public ResponseEntity<List<Flat>> addAll(@AuthenticationPrincipal User user,  
 @RequestBody List<Flat> flats) {  
 return ResponseEntity.*ok*(flatService.saveAll(flats, user));  
 }  
  
 @PostMapping("/update")  
 public ResponseEntity<Flat> update(@AuthenticationPrincipal User user,  
 @RequestBody Flat flat) {  
 return ResponseEntity.*ok*(flatService.save(flat, user));  
 }  
  
 @PostMapping("/remove")  
 public ResponseEntity<String> remove(@AuthenticationPrincipal User user,  
 @RequestParam Long id) {  
 flatService.removeByIdAndUserId(id, user.getId());  
 return ResponseEntity.*ok*("Flat is removed");  
 }  
}

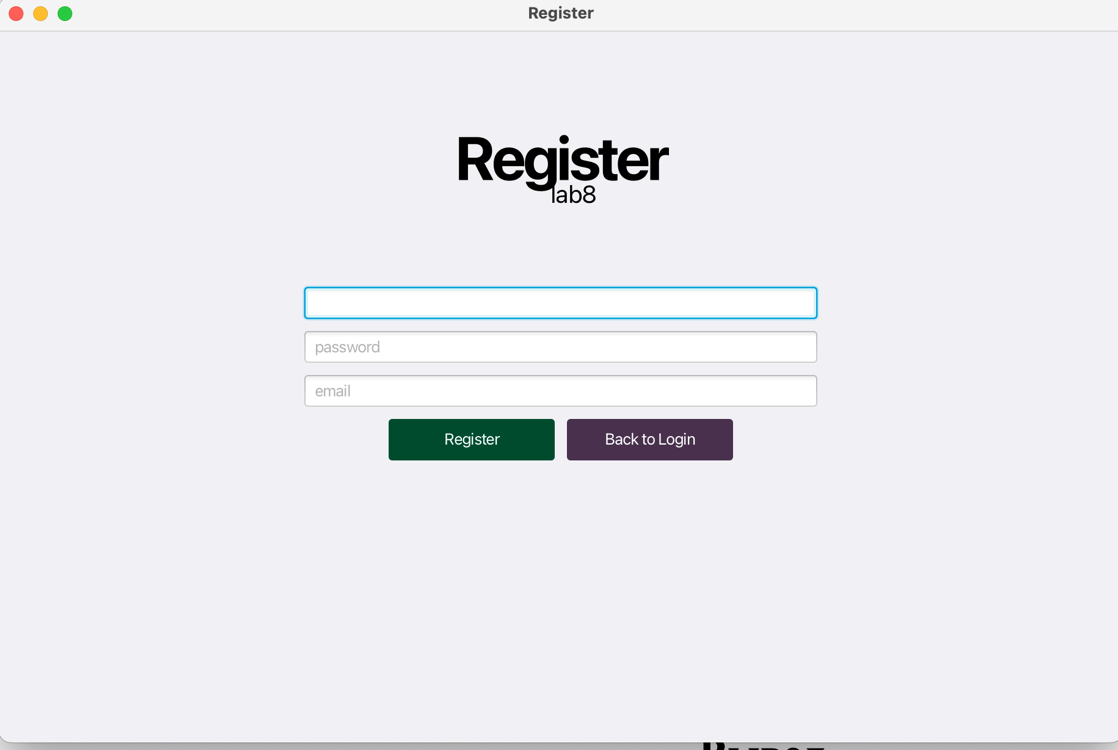
**Весь остальной код: [лабораторная работа №8 (клиент)](https://github.com/serezk4/ITMO.labs/tree/master/programming/lab8client)** [**лабораторная работа №8 (сервер)**](https://github.com/serezk4/ITMO.labs/tree/master/programming/lab8servre)

**Результат работы**

****

****



****

**Вывод**

Я получил интересный опыт в разработке клиентского приложения с GUI на JavaFX, поработал над созданием API коллекции и применил Spring Security для авторизации пользователей.