**Отчет о системе продажи билетов на спортивные мероприятия (TicketBookingSystem)**

**Архитектура приложения**

Приложение разработано с использованием трехслойной архитектуры, которая включает следующие слои:

1. **Слой представления данных:**
   * Этот слой отвечает за взаимодействие с пользователем и отображение данных. Используются JSP страницы для рендеринга пользовательского интерфейса.
   * Пользовательские запросы поступают через контроллеры Spring MVC, которые обрабатывают запросы и вызывают соответствующие методы сервисного слоя.
2. **Service-слой:**
   * Этот слой содержит бизнес-логику приложения. Он взаимодействует с репозиториями (ранее DAO) для выполнения операций с базой данных.
   * Сервисные классы отвечают за выполнение операций, таких как создание, обновление и удаление сущностей, а также за проведение валидации данных.
3. **Слой доступа к данным (Repository):**
   * Этот слой отвечает за взаимодействие с базой данных. Используется Spring Data JPA.
   * Репозитории (ранее DAO) выполняют CRUD (Create, Read, Update, Delete) операции и выполняют запросы к базе данных.

**Использованные технологии**

1. **Spring Framework:**
   * Основной платформой для разработки приложения является Spring Framework, который обеспечивает мощные возможности для создания корпоративных приложений, включая поддержку MVC, безопасность и работу с базой данных через JPA.
2. **Spring MVC:**

* Spring MVC (Model-View-Controller) используется для создания веб-приложений, поддерживая четкое разделение между представлением, бизнес-логикой и контроллерами. Это позволяет легко управлять входящими HTTP-запросами, обрабатывать их и возвращать представления пользователю.
* Настройка маршрутизации осуществляется с использованием аннотации @RequestMapping, что обеспечивает гибкость и удобство в разработке веб-приложений.

Ниже приведен пример соответствующей Java based конфигурации для модуля:

@Configuration  
@EnableWebMvc  
public class WebMvcConfig implements WebMvcConfigurer {  
  
 @Bean  
 public LocaleResolver localeResolver() {  
 return new SessionLocaleResolver();  
 }  
  
 @Bean  
 public LocaleChangeInterceptor localeChangeInterceptor() {  
 LocaleChangeInterceptor lci = new LocaleChangeInterceptor();  
 lci.setParamName("lang");  
 return lci;  
 }  
  
 @Bean  
 public ResourceBundleMessageSource messageSource() {  
 ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();  
 messageSource.setBasename("messages");  
 messageSource.setDefaultEncoding("UTF-8");  
 return messageSource;  
 }  
  
 @Bean  
 public HandlerMappingIntrospector mvcHandlerMappingIntrospector() {  
 return new HandlerMappingIntrospector();  
 }  
  
 @Bean  
 public InternalResourceViewResolver viewResolver() {  
 InternalResourceViewResolver resolver = new InternalResourceViewResolver();  
 resolver.setPrefix("/WEB-INF/jsp/");  
 resolver.setSuffix(".jsp");  
 return resolver;  
 }  
  
 @Override  
 public void addResourceHandlers(ResourceHandlerRegistry registry) {  
 registry.addResourceHandler("/css/\*\*").addResourceLocations("/css/");  
 registry.addResourceHandler("/js/\*\*").addResourceLocations("/js/");  
 }  
  
 @Override  
 public void addInterceptors(InterceptorRegistry registry) {  
 registry.addInterceptor(localeChangeInterceptor());  
 }  
  
 @Override  
 public void addArgumentResolvers(List<HandlerMethodArgumentResolver> resolvers) {  
 PageableHandlerMethodArgumentResolver pageableResolver = new PageableHandlerMethodArgumentResolver();  
 pageableResolver.setPageParameterName("page");  
 pageableResolver.setSizeParameterName("size");  
 pageableResolver.setFallbackPageable(PageRequest.*of*(0, 10));  
 resolvers.add(pageableResolver);  
 }  
}

public class WebAppInitializer implements WebApplicationInitializer {  
  
 @Override  
 public void onStartup(ServletContext servletContext) {  
  
 AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext();  
 context.register(ApplicationConfiguration.class);  
 context.register(SecurityConfiguration.class);  
 context.register(WebMvcConfig.class);  
  
 DispatcherServlet servlet = new DispatcherServlet(context);  
 ServletRegistration.Dynamic registration = servletContext.addServlet("dispatcher", servlet);  
 registration.setLoadOnStartup(1);  
 registration.addMapping("/");  
 }  
}

1. **Spring Data JPA:**
   * Заменяет Hibernate SessionFactory на EntityManagerFactory и предоставляет удобные репозитории для управления объектно-реляционными отображениями и взаимодействия с базой данных PostgreSQL.

Ниже приведен пример соответствующей Java based конфигурации для модуля:

@Configuration  
@EnableTransactionManagement  
@EnableJpaRepositories(basePackages = "com.example.ticketbookingsystem.repository")  
@ComponentScan(basePackages = "com.example.ticketbookingsystem")  
public class JpaConfig {  
  
 @Bean  
 public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource dataSource) {  
 LocalContainerEntityManagerFactoryBean em = new LocalContainerEntityManagerFactoryBean();  
 em.setDataSource(dataSource);  
 em.setPackagesToScan("com.example.ticketbookingsystem.entity");  
 JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();  
 em.setJpaVendorAdapter(vendorAdapter);  
 em.setJpaProperties(additionalProperties());  
  
 return em;  
 }  
  
 @Bean  
 public DataSource dataSource() {  
 HikariConfig config = new HikariConfig();  
 config.setDriverClassName("org.postgresql.Driver");  
 config.setJdbcUrl(System.*getenv*("DB\_URL"));  
 config.setUsername(System.*getenv*("DB\_USERNAME"));  
 config.setPassword(System.*getenv*("DB\_PASSWORD"));  
  
 return new HikariDataSource(config);  
 }  
  
 @Bean  
 public PlatformTransactionManager transactionManager(EntityManagerFactory emf) {  
 JpaTransactionManager transactionManager = new JpaTransactionManager();  
 transactionManager.setEntityManagerFactory(emf);  
  
 return transactionManager;  
 }  
  
 Properties additionalProperties() {  
 Properties properties = new Properties();  
 properties.setProperty("hibernate.dialect", "org.hibernate.dialect.PostgreSQLDialect");  
 properties.setProperty("hibernate.show\_sql", "true");  
 properties.setProperty("hibernate.format\_sql", "true");  
  
 return properties;  
 }  
}

1. **PostgreSQL:**
   * В качестве системы управления базами данных используется PostgreSQL.
2. **JSP:**
   * JavaServer Pages (JSP) используются для создания динамических веб-страниц, которые взаимодействуют с пользователем.
3. **Spring Security:**
   * Используется для настройки базовой аутентификации и управления ролями, что обеспечивает безопасность приложения.

Ниже приведен пример соответствующей Java based конфигурации для модуля:

@Configuration  
@EnableWebSecurity  
@RequiredArgsConstructor  
public class SecurityConfiguration {  
  
 private final CustomUserDetailsService customUserDetailsService;  
  
 @Autowired  
 public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(customUserDetailsService).passwordEncoder(passwordEncoder());  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/registration", "/css/\*\*", "/js/\*\*", "/WEB-INF/jsp/\*\*").permitAll()  
 .requestMatchers(request -> request.getParameter("lang") != null).permitAll()  
 .requestMatchers("/admin/\*\*").hasAuthority(*ADMIN*.getAuthority())  
 .requestMatchers("/view\_available\_events",  
 "/view\_available\_tickets",  
 "/user\_cart",  
 "/purchase",  
 "/purchasedTickets").hasAuthority(*USER*.getAuthority())  
 .anyRequest().authenticated())  
 .formLogin(login -> login  
 .loginPage("/login")  
 .successHandler(myAuthenticationSuccessHandler())  
 .permitAll())  
 .logout(logout -> logout  
 .logoutUrl("/logout")  
 .logoutSuccessUrl("/login")  
 .deleteCookies("JSESSIONID"));  
  
 return http.build();  
 }  
  
 @Bean  
 public AuthenticationSuccessHandler myAuthenticationSuccessHandler() {  
 return new MyAuthenticationSuccessHandler();  
 }  
  
 public static class MyAuthenticationSuccessHandler implements AuthenticationSuccessHandler {  
 @Override  
 public void onAuthenticationSuccess(HttpServletRequest request,  
 HttpServletResponse response,  
 Authentication authentication) throws IOException {  
 String redirectUrl = "/view\_available\_events";  
  
 for (var authority : authentication.getAuthorities()) {  
 if (authority.getAuthority().equals("ADMIN")) {  
 redirectUrl = "/admin";  
 break;  
 }  
 }  
  
 response.sendRedirect(redirectUrl);  
 }  
 }  
  
}

**Особенности реализации**

1. **Фильтры:**
   * LocaleFilter обрабатывает изменение языка интерфейса на основе параметров URL и cookie.
2. **Валидация данных:**
   * Валидация данных осуществляется на уровне сервисного слоя с использованием валидаторов и аннотаций javax.validation (@NotNull, @Size и т.д.).
3. **DTO и мапперы:**
   * Используются DTO (Data Transfer Objects) для передачи данных между слоями приложения.
   * Мапперы преобразуют сущности в DTO и наоборот, обеспечивая изоляцию слоев и упрощение передачи данных. Используется MapStruct для автоматического маппинга.
4. **Управление транзакциями:**
   * Управление транзакциями осуществляется с использованием аннотации @Transactional (при необходимости), что обеспечивает целостность данных и корректность выполнения операций с базой данных.
5. **Юнит-тестирование:**
   * Написаны юнит-тесты для контроллеров и сервисов с использованием JUnit 5 и Mockito. Также написаны интеграционные тесты для проверки взаимодействия нескольких компонентов (соответствующих контроллеров, сервисов и репозиториев между собой). Для интеграционного тестирования используется H2 база данных.