Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

**Дисциплина «Программирование интернет-приложений»**

**Курсовая работа**

**Вариант 2017**

**Выполнили:**

Съестов Дмитрий Вячеславович

Давлетшин Рушан Равильевич

Группа P3217

**Преподаватель:**

Николаев Владимир Вячеславович

Санкт-Петербург

2017

1. **Техническое задание**

Система представляет собой простой блог на астрономическую тематику. Отличительной особенностью является наличие интерактивной карты Солнечной системы, в которой планеты и их спутники вращаются вокруг своих орбит. По клику на объект можно прочитать описание и посмотреть связанные статьи.

* Администраторы могут писать статьи и редактировать уже написанные. Объекты на карте связаны с определённой статьёй, так что при её изменении изменится описание объекта.
* Пользователи соцсетей могут комментировать материалы и добавлять их в избранное.

1. **Требования**

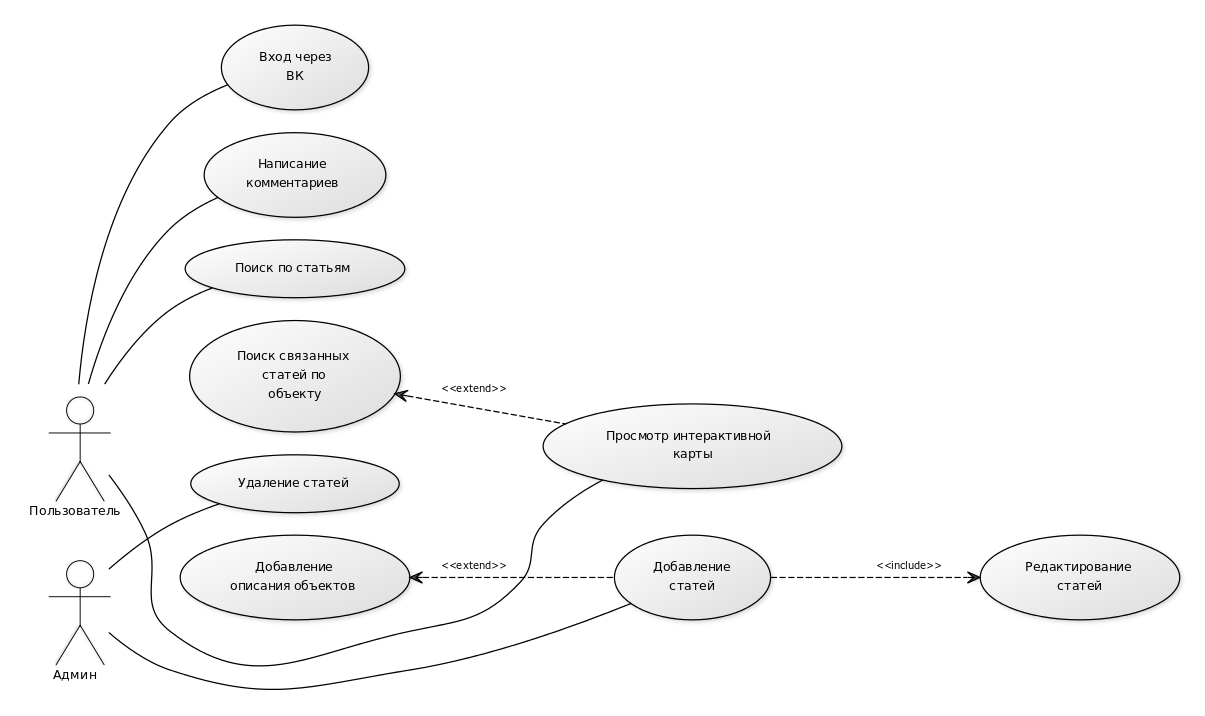
**Функциональные требования**

|  |  |  |
| --- | --- | --- |
| **ID** | **Требование** | **Описание** |
| FRU1 | Возможность администрирования | Система должна позволять администратору редактировать и удалять статьи |
| FRU2 | Добавление материалов | Система должна позволять администратору создавать статьи |
| FRU3 | Аутентификация через соцсети | Система должна поддерживать аутентификацию через аккаунты социальных сетей с использованием протокола 0Auth |
| FRU4 | Новостная лента | Система должна отображать свежие статьи на главной странице и в ленте |
| FRU5 | Комментирование | Система должна позволять вошедшим через соцсети пользователям оставлять комментарии под материалами |
| FRU6 | Поиск | Система должна позволять пользователю осуществлять поиск по тексту статей |

**Нефункциональные требования**

|  |  |  |
| --- | --- | --- |
| **ID** | **Требование** | **Описание** |
| QR1 | Back-end | Система должна использовать Enterprise JavaBeans на уровне back-end |
| QR2 | Front-end | Система должна использовать ReactJS, ECMAScript, JSX на уровне front-end |
| QR3 | Использование REST API | Система должна использовать REST API для организации взаимодействия между уровнями front-end и back-end |
| QR4 | Режимы веб-интерфейсов | Система должна иметь три режима для отображения веб-интерфейсов:   * "Десктопный" - для устройств, ширина экрана которых равна или превышает 1182 пикселей. * "Планшетный" - для устройств, ширина экрана которых равна или превышает 858, но меньше 1182 пикселей. * "Мобильный"- для устройств, ширина экрана которых меньше 858 пикселей. |
| QR5 | Использование JPA | Система должна использовать Java Persistence API для доступа к базе данных |

1. **Прецеденты использования**



|  |
| --- |
| **LoginUsingVK** |
| **ID**: 1 |
| **Краткое** **описание**: пользователь входит через ВК |
| **Актёры**: пользователь |
| **Основной поток:**   1. Пользователь нажимает кнопку «Вход через ВК»   1.1. Пользователь вводит логин и пароль  1.1. ALT VKLoginFailure |

|  |
| --- |
| **VKLoginFailure** |
| **ID**: 2 |
| **Краткое** **описание**: неудачная попытка входа через ВК |
| **Актёры**: пользователь |
| **Предусловия:** введён неверный логин и/или пароль |
| **Основной поток:**   1. Пользователь получает сообщение об ошибке   1.1. ALT LoginUsingVK |

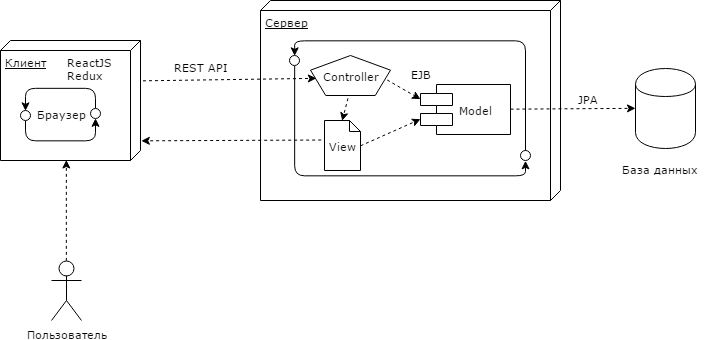
|  |
| --- |
| **Comment** |
| **ID**: 3 |
| **Краткое** **описание**: написание комментария |
| **Актёры**: пользователь |
| **Предусловия:** пользователь вошёл через аккаунт соцсети |
| **Основной поток:**   1. Пользователь набирает текст комментария и нажимает «Отправить» |

|  |
| --- |
| **Post** |
| **ID**: 4 |
| **Краткое** **описание**: добавление материала |
| **Актёры**: администратор |
| **Предусловия:** пользователь является админом |
| **Основной поток:**   1. Администратор нажимает «Новая статья»    1. Администратор вводит текст и название    2. В базе данных появляется новая статья |

|  |
| --- |
| **Edit** |
| **ID**: 5 |
| **Краткое** **описание**: редактирование материала |
| **Актёры**: администратор |
| **Предусловия:** пользователь является админом |
| **Основной поток:**   1. Администратор нажимает «Редактировать»    1. Администратор вводит новые текст и название    2. В базе данных меняются соответствующие значения |

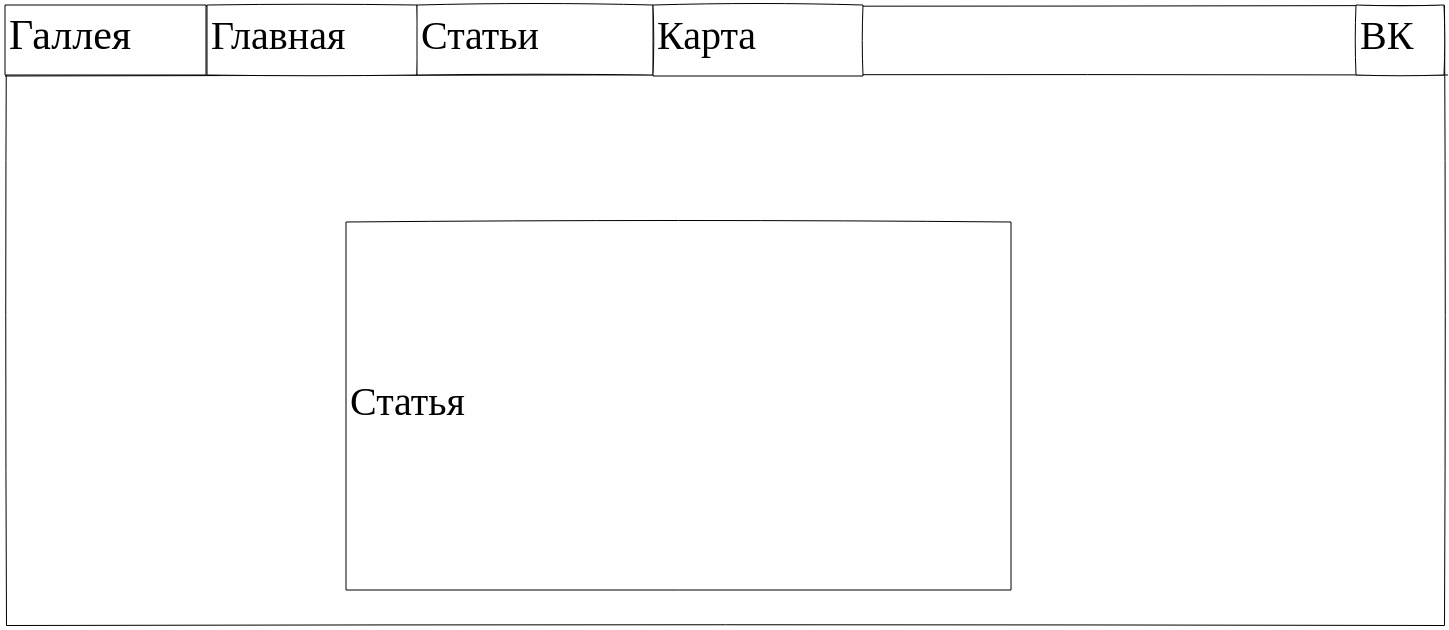
1. **Архитектура системы**

* За отображение данных отвечает ReactJS.
* Обработка данных осуществляется сервлетами с использованием Enterprise JavaBeans.
* Данные хранятся в БД, доступ к которой осуществляется через Java Persistence API.
* Обмен данными между клиентом и сервером осуществляется по REST API.



1. **Прототипы интерфейсов**

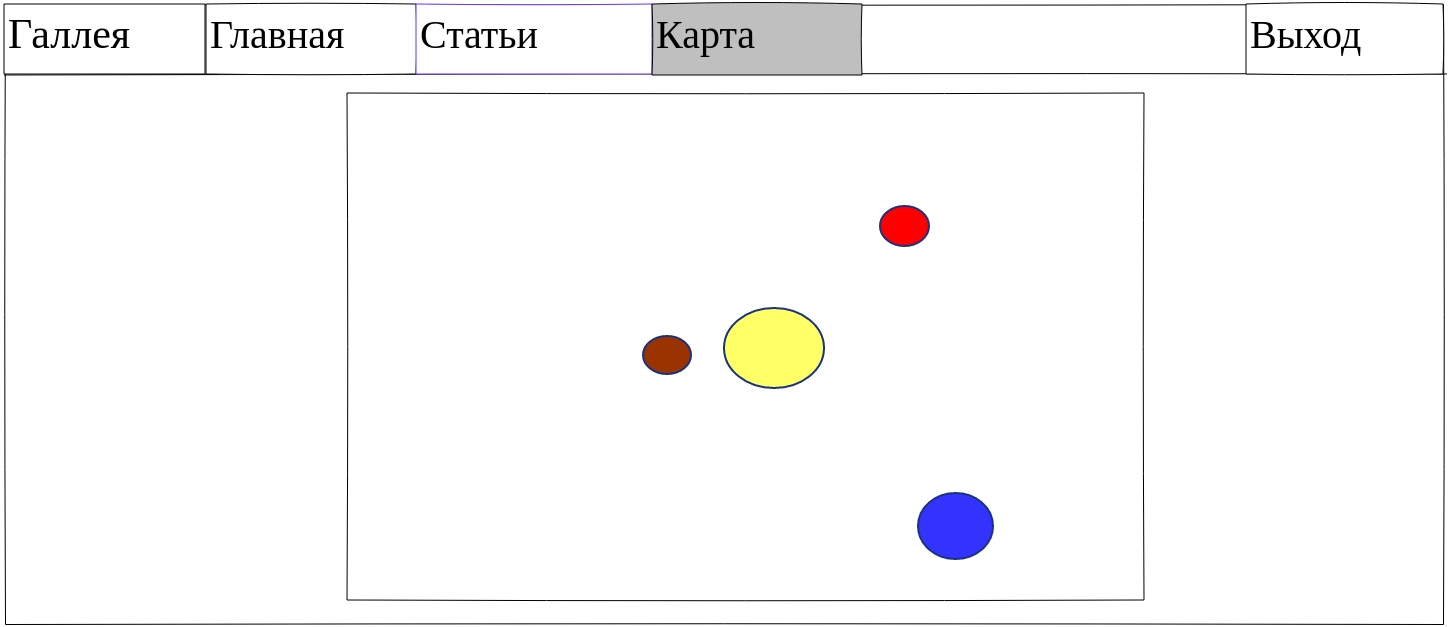
Главная страница



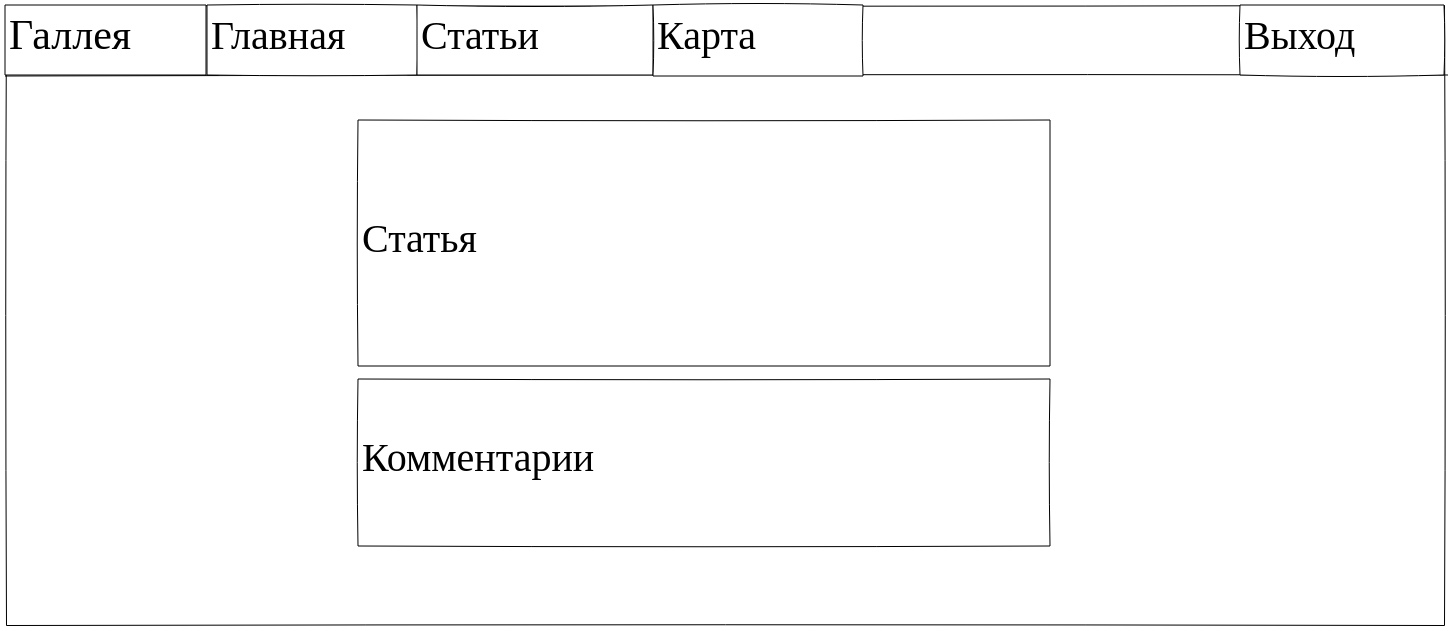
Просмотр статей



Интерактивная карта (пользователь авторизован)



Просмотр отдельной статьи



Меню для мобильных устройств



1. **Уровень Data Access**

CRUD API реализован через бины и DAO (Data Access Object), наследующиеся от класса AbstractDao:

*/\*\*  
 \* Предоставляет абстрактный интерфейс для создания, чтения и удаления данных в БД.  
 \** ***@author*** *Дмитрий  
 \** ***@since*** *2 этап  
 \*/*@Stateful  
@Produces(MediaType.*APPLICATION\_JSON*)  
public abstract class AbstractDao<T, TId> implements Serializable {  
  
 protected final EntityManagerFactory factory;  
 protected final EntityManager em;  
  
 private final Class<T> type;  
  
 protected AbstractDao(Class<T> type) {  
 try {  
 factory = Persistence.*createEntityManagerFactory*("halley");  
 } catch (Throwable ex) {  
 System.*err*.println("Failed to create EntityManagerFactory: " + ex);  
 throw new PersistenceException(ex.getMessage());  
 }

try {  
 em = factory.createEntityManager();  
 } catch (Throwable ex) {  
 System.*err*.println("Failed to create EntityManager: " + ex);  
 throw new PersistenceException(ex.getMessage());  
 }  
  
 this.type = type;  
 }  
  
 */\*\*  
 \* Сохраняет объект в базе данных.  
 \** ***@author*** *Дмитрий  
 \** ***@since*** *2 этап  
 \*  
 \** ***@param*** *item Объект для записи в БД.  
 \*/* public void persist(T item) {  
 EntityTransaction tx = null;  
 try {  
 tx = em.getTransaction();  
 tx.begin();  
 em.persist(item);  
 tx.commit();  
 } catch (Throwable e) {  
 if (tx != null) tx.rollback();  
 e.printStackTrace();  
 throw e;  
 }  
 }  
  
 */\*\*  
 \* Получает запись из базы данных по первичному ключу.  
 \** ***@author*** *Дмитрий  
 \** ***@since*** *2 этап  
 \*  
 \** ***@param*** *key Первичный ключ записи  
 \*  
 \** ***@return*** *Полученный из базы данных объект  
 \** ***@throws*** *NoResultException Если запись не найдена  
 \*/* public T read(TId key) throws NoResultException {  
 return em.find(type, key);  
 }  
  
 public boolean exists(TId key) {  
 try {  
 read(key);  
 return true;  
 } catch (NoResultException ex) {  
 return false;  
 }  
 }  
  
 */\*\*  
 \* Удаляет запись из базы данных. Запись должна управляться JPA (т.е. получена методом read() или JPQL-запросом)  
 \** ***@author*** *Дмитрий  
 \** ***@since*** *2 этап  
 \*  
 \** ***@param*** *item Запись, которую нужно удалить  
 \*  
 \** ***@throws*** *NoResultException Если запись не найдена  
 \*/* public void delete(T item) {  
 EntityTransaction tx = null;  
 try {  
 tx = em.getTransaction();  
 tx.begin();  
 em.remove(item);  
 tx.commit();  
 }  
 catch (NoResultException e) {  
 if (tx != null) tx.rollback();  
 e.printStackTrace();  
 throw e;  
 }  
 }

*/\*\*  
 \* Удаляет объект из базы данных по первичному ключу.  
 \** ***@author*** *Дмитрий  
 \** ***@since*** *2 этап  
 \*  
 \** ***@param*** *key Первичный ключ записи, которую нужно удалить  
 \*  
 \** ***@throws*** *NoResultException Если запись не найдена  
 \*/* public void deleteByKey(TId key) throws NoResultException {  
 T item = read(key);  
 delete(item);  
 };  
}

Пример entity-бина:

@Entity  
@Table(name = "articles", schema = "s225116")  
public class Article  
{  
 private Integer id;  
 private String name;  
 private String date;  
 private String text;  
  
 public Article() {  
 DateFormat df = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss:SSSS");  
 Date today = Calendar.*getInstance*().getTime();  
 date = df.format(today);  
 }  
  
 public Article(String name, String text) {  
 this();  
 this.name = name;  
 this.text = text;  
 }

@Id  
 @Column(name = "id", nullable = false, columnDefinition = "serial")  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 public Integer getId() { return id; }  
 public void setId(Integer id) { this.id = id; }  
  
 @Basic  
 @Column(name = "name", nullable = false)  
 public String getName() { return name; }  
 public void setName(String name) { this.name = name; }  
  
 @Basic  
 @Column(name = "date", nullable = false)  
 public String getDate() { return date; }  
 public void setDate(String date) { this.date = date; }

@Basic  
 @Column(name = "text", nullable = false, columnDefinition = "text")  
 public String getText() { return text; }  
 public void setText(String text) { this.text = text; }  
}

Пример использования:

Article article = new Article("Lorem ipsum", "Lorem ipsum dolor sit amet...");

ArticleDao articleDao = new ArticleDao();

articleDao.persist(article);

1. **Бэкэнд**

Взаимодействие с сервером осуществляется через HTTP-запросы к бинам, обрабатываемые библиотекой Jersey. Данные OAuth-авторизации хранятся в данных сессии и запрашиваются клиентом при необходимости.

@Singleton  
@Path(value = "/user")  
public class OAuthBean implements AuthorizationInterface, AuthorizationFromSocialNetworksInterface {  
 @EJB  
 private OAuthService OAuthService;  
 @EJB  
 private ValidatorAuthInterface validatorService;  
  
 public OAuthBean() {  
 final String clientId = "CLIENT\_ID";  
 final String clientSecret = "CLIENT\_SECRET";  
 OAuth20Service service = new ServiceBuilder(clientId)  
 .apiSecret(clientSecret)  
 .scope("email")  
 .callback("http://localhost:49680/rest/user/vk/callback")  
 .build(VkontakteApi.*instance*());  
  
 this.service = service;  
 }  
  
 private static final String *PROTECTED\_RESOURCE\_URL* = "https://api.vk.com/method/users.get?v="  
 + VkontakteApi.*VERSION*;  
  
 private final OAuth20Service service;  
  
 @GET  
 @Path("/vk")  
 public String loginUsingVK(@Context HttpServletResponse response,  
 @Context HttpServletRequest request  
 ) {  
 final String authorizationUrl;  
 try {  
 authorizationUrl = service.getAuthorizationUrl();  
 } catch (Exception e) {  
 return e.toString();  
 }  
 return authorizationUrl;  
 }  
 @GET  
 @Path("/vk/callback")  
 public String authUsingVK(  
 @QueryParam("code") String code,  
 @Context HttpServletRequest req,  
 @Context HttpServletResponse resp  
 ) {  
 String firstName = null,  
 lastName = null;  
 int id = 0;  
  
 OAuth2AccessToken accessToken = null;  
 Response response = null;  
 try {  
 accessToken = service.getAccessToken(code);  
  
 OAuthRequest request = new OAuthRequest(Verb.*GET*,*PROTECTED\_RESOURCE\_URL*);  
 service.signRequest(accessToken,request);  
 response = service.execute(request);  
  
 JSONObject body = new JSONObject(response.getBody());  
 JSONArray arr = body.getJSONArray("response");  
  
 for (int i = 0; i < arr.length(); i++)  
 {  
 id = arr.getJSONObject(i).getInt("id");  
 firstName = arr.getJSONObject(i).getString("first\_name");  
 lastName = arr.getJSONObject(i).getString("last\_name");  
 }  
  
 req.getSession().setAttribute("id",id);  
 req.getSession().setAttribute("first\_name",firstName);  
 req.getSession().setAttribute("last\_name",lastName);  
  
 resp.sendRedirect("/");  
 return id + " " + firstName + " " + lastName;  
 } catch (Exception e) {  
 return e.toString();  
 }  
 }  
  
 @POST  
 @Path("/logout")  
 public void logout(@QueryParam("id") String id,  
 @Context HttpServletResponse response,  
 @Context HttpServletRequest request  
 ) {  
 try {  
 request.getSession().invalidate();  
 response.sendRedirect("/");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }

@GET  
 @Path("/param")  
 public String getParam(@Context HttpServletRequest request)  
 {  
 int id = (int) request.getSession().getAttribute("id");  
 String firstName = (String) request.getSession().getAttribute("first\_name"),  
 lastName = (String) request.getSession().getAttribute("last\_name");  
  
 return id + ";" + firstName + ";" + lastName + ";";  
 }  
}

1. **Фронтэнд**

Фронтэнд построен на фреймворке ReactJS с использованием Typescript, SCSS и

CSS-фреймворка mini.css, который, в частности, используется для адаптивной вёрстки.

Адаптивная вёрстка проявляется следующим образом:

* На средних экранах скрывается футер
* На маленьких экранах:
  + исчезают отступы по бокам основной части приложения
  + меню в шапке сворачивается в гамбургер
  + строка поиска расширяется во весь экран

Данные авторизации хранятся в состоянии корневого элемента и передаются через контекст. Вход и выход осуществляются через коллбэки, передаваемые вместе с ними.

Пример компонента с использованием свойств, состояния, хуков ЖЦ и REST API:

**export interface** IPlanetProps {  
 id: **string**;  
 name: **string**;  
 articleId: **number**;  
 orbit?: {  
 radius: **number**; //в а.е.  
 period: **number**; //в земных годах  
 };  
 sunId?: **string**;  
}  
  
**export interface** IPlanetState {  
 description?: **string**;  
}  
  
**export class** Planet **extends** React.Component<IPlanetProps, IPlanetState> {  
 **constructor**(props: IPlanetProps) {  
 **super**(props);  
  
 **this**.state = {};  
 }

componentDidMount() {  
 **const** {id, sunId, articleId} = **this**.props;  
 **if** (**this**.props.orbit) {  
 **const** {radius, period} = **this**.props.orbit;  
  
 **const** centerStyle = window.getComputedStyle(document.getElementById(sunId));  
 **const** pxRadius = radius \* PIXELS\_PER\_AU + parseInt(centerStyle.width)/2;  
 **const** msPeriod = period \* MSECONDS\_PER\_YEAR;  
 orbit(id, sunId, pxRadius, msPeriod);  
 }  
  
 **this**.setState({  
 description: parseJson("/rest/articles/" + articleId, **true**)  
 });  
 }  
  
 **public** render(): JSX.Element[] {  
 **const** toggle = (id: **string**) => {  
 **const** checkbox = document.getElementById(id) **as** HTMLInputElement;  
 checkbox.checked = **true**;  
 };  
  
 **const** dialogId = uniqueId("dialog\_");  
  
 **const** {id, name, articleId} = **this**.props;  
 **const** {description} = **this**.state;  
 **return** ([  
 <div id={id} key={uniqueId("planet\_")} onClick={() => toggle(dialogId)} className="planet"/>,  
 <div key={dialogId}>  
 <input type="checkbox" id={dialogId} className="modal"/>  
 <div>  
 <div className="card">  
 <label htmlFor={dialogId} className="modal-close"/>  
 <h3 className="section">{name}</h3>  
 <p className="section">{description}</p>  
 <Link to="" className="section">Связанные статьи</Link>  
 </div>  
 </div>  
 </div>  
 ]);  
 }  
}

**Вывод**

В ходе выполнения данной курсовой работы мы научились организовывать сборку веб-приложения с помощью Gradle, проверять права пользователя, реализовывать авторизацию через соцсети, а также верстать адаптивные интерфейсы, построенные из многократно используемых компонентов.