

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 Р3217

**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

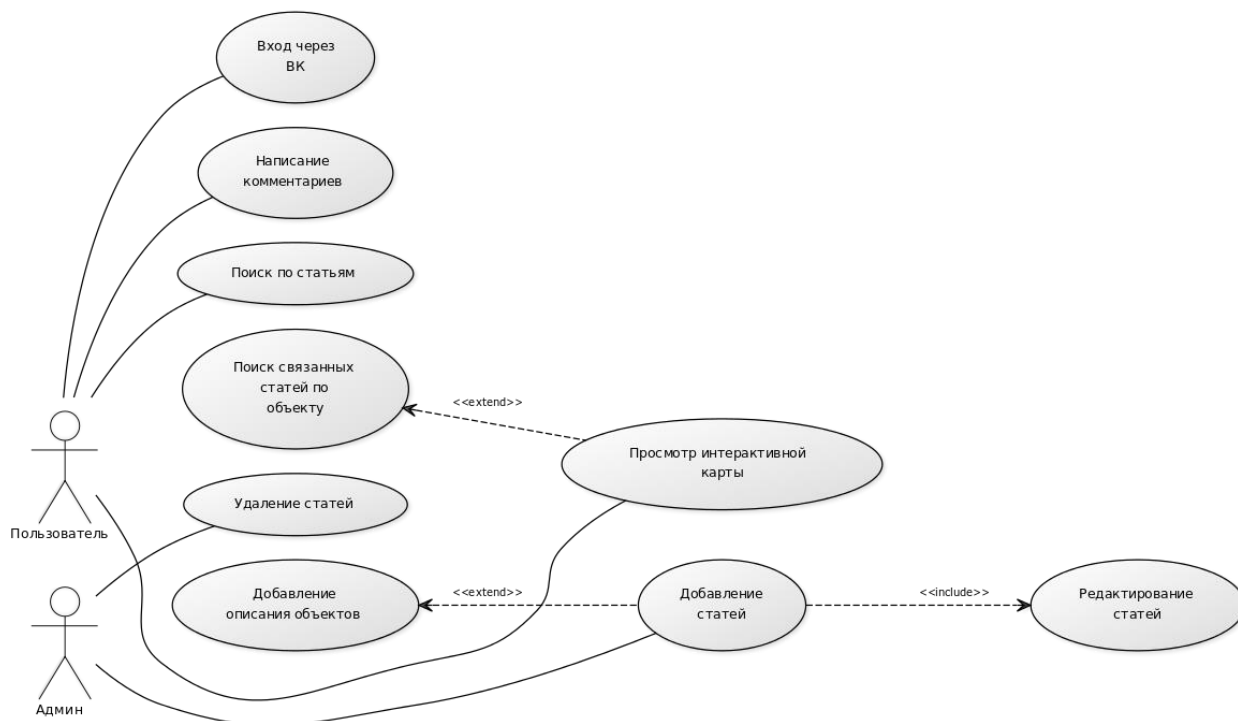
ID	Требование	Описание
FRU1	Возможность администрирования	Система должна позволять администратору редактировать и удалять статьи
FRU2	Добавление материалов	Система должна позволять администратору создавать статьи
FRU3	Аутентификация через соцсети	Система должна поддерживать аутентификацию через аккаунты социальных сетей с использованием протокола OAuth
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	<p>Система должна иметь три режима для отображения веб-интерфейсов:</p> <ul style="list-style-type: none"> <li>"Десктопный" - для устройств, ширина экрана которых равна или превышает 1182 пикселей.</li> <li>"Планшетный" - для устройств, ширина экрана которых равна или превышает 858, но меньше 1182 пикселей.</li> <li>"Мобильный" - для устройств, ширина экрана которых меньше 858 пикселей.</li> </ul>
QR5	Использование JPA	Система должна использовать Java Persistence API для доступа к базе данных

## 2. Прецеденты использования



LoginUsingVK	
<b>ID:</b> 1	
<b>Краткое описание:</b>	пользователь входит через ВК
<b>Актёры:</b>	пользователь
<b>Основной поток:</b>	<ol style="list-style-type: none"> <li>Пользователь нажимает кнопку «Вход через ВК» <ol style="list-style-type: none"> <li>Пользователь вводит логин и пароль <ol style="list-style-type: none"> <li>ALT VKLoginFailure</li> </ol> </li> </ol> </li> </ol>

<b>VKLoginFailure</b>
<b>ID:</b> 2
<b>Краткое описание:</b> неудачная попытка входа через ВК
<b>Актёры:</b> пользователь
<b>Предусловия:</b> введён неверный логин и/или пароль
<b>Основной поток:</b> <ol style="list-style-type: none"> <li>Пользователь получает сообщение об ошибке <ol style="list-style-type: none"> <li>ALT LoginUsingVK</li> </ol> </li> </ol>

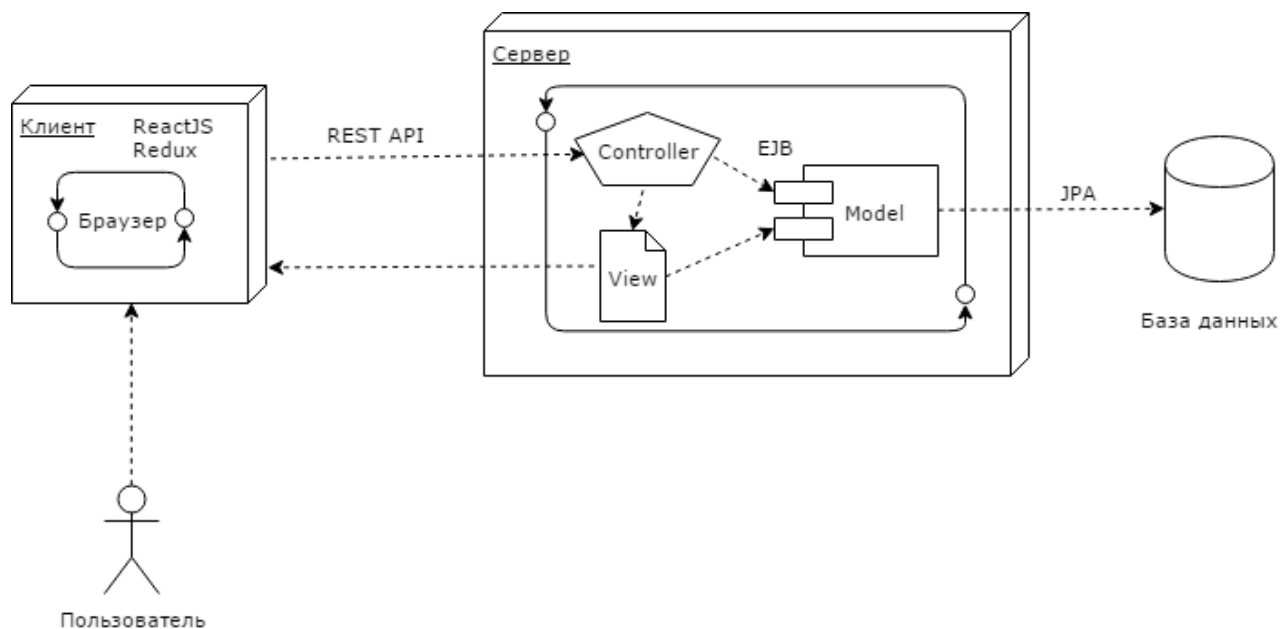
<b>Comment</b>
<b>ID:</b> 3
<b>Краткое описание:</b> написание комментария
<b>Актёры:</b> пользователь
<b>Предусловия:</b> пользователь вошёл через аккаунт соцсети
<b>Основной поток:</b> <ol style="list-style-type: none"> <li>Пользователь набирает текст комментария и нажимает «Отправить»</li> </ol>

<b>Post</b>
<b>ID:</b> 4
<b>Краткое описание:</b> добавление материала
<b>Актёры:</b> администратор
<b>Предусловия:</b> пользователь является админом
<b>Основной поток:</b> <ol style="list-style-type: none"> <li>Администратор нажимает «Новая статья» <ol style="list-style-type: none"> <li>Администратор вводит текст и название</li> <li>В базе данных появляется новая статья</li> </ol> </li> </ol>

<b>Edit</b>
<b>ID:</b> 5
<b>Краткое описание:</b> редактирование материала
<b>Актёры:</b> администратор
<b>Предусловия:</b> пользователь является админом
<b>Основной поток:</b> <ol style="list-style-type: none"> <li>Администратор нажимает «Редактировать» <ol style="list-style-type: none"> <li>Администратор вводит новые текст и название</li> <li>В базе данных меняются соответствующие значения</li> </ol> </li> </ol>

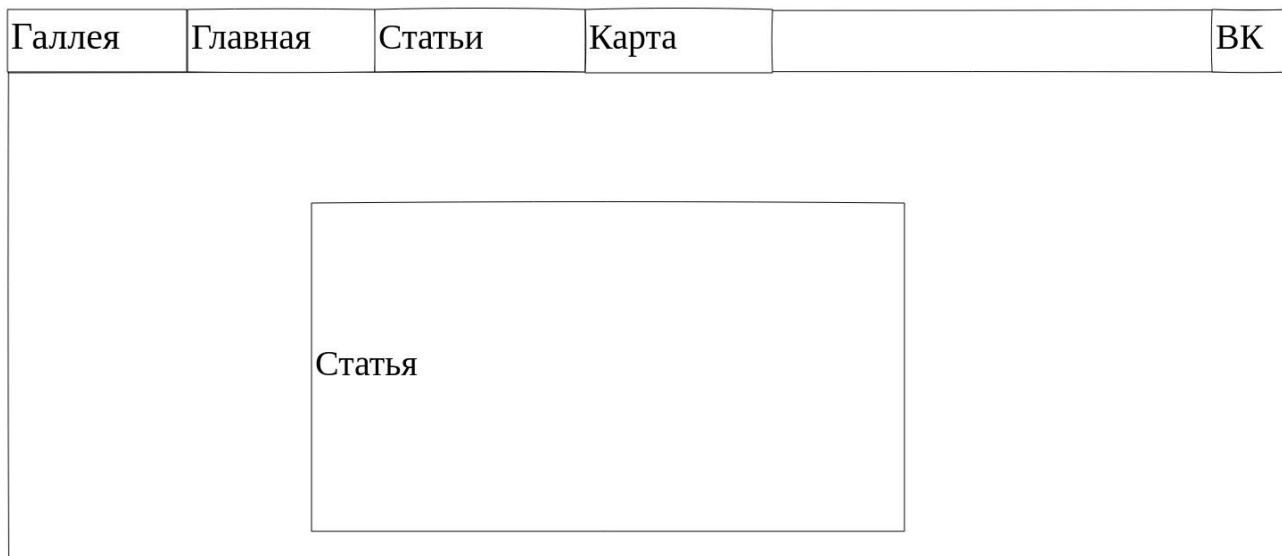
### 3. Архитектура системы

- 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.



### 4. Прототипы интерфейсов

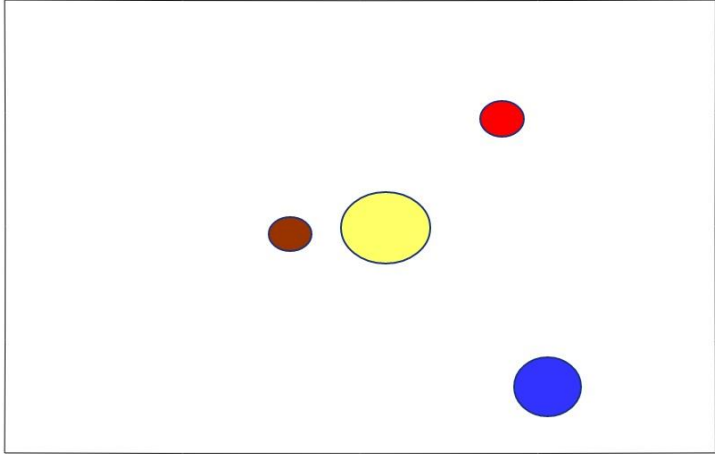
Главная страница



## Просмотр статей

Галлея	Главная	Статьи	Карта		ВК
<div>Статья</div> <div>Статья</div> <div>Статья</div>					

## Интерактивная карта (пользователь авторизован)

Галлея	Главная	Статьи	Карта		Выход
<div></div>					

## Просмотр отдельной статьи

Галлея	Главная	Статьи	Карта		Выход
<div>Статья</div> <div>Комментарии</div>					

## Меню для мобильных устройств

≡	Гал	Главная
		Статьи
		Карта
		Выход
	Стат	

## 2. 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);

```

## 3. 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 = "6698669";
        final String clientSecret = "Lj7j7iY7ShZgedvSLuel";
        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 + ";";
}
}

```

## 4. 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.