**3. Проектно-конструкторская часть**

# 3.1 Разработка структуры программной системы

Приложение представляет собой клиент-серверное приложение, которое является трехуровневым. Клиент-серверное приложение — это вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение.

Трехуровневое клиент-серверное (рис. 5) приложение содержит в себе:

* Представление данных (клиент). В данной курсовой работе клиентом будет являться веб-браузер, который будет передавать веб-серверу запросы на получение HTML-страниц, изображений, файлов и других данных, обозначенных URL-адресами.
* Сервер приложений располагается на втором уровне, на нём сосредоточена большая часть бизнес-логики. Вне его остаются только фрагменты, экспортируемые на клиента, а также элементы логики, погруженные в базу данных. Реализация данного компонента обеспечивается связующим программным обеспечением.
* Сервер базы данных, выполняет обслуживание и управление базой данных и СУБД, а также отвечает за целостность и сохранность данных и обеспечивает операции ввода-вывода при доступе клиента к информации.

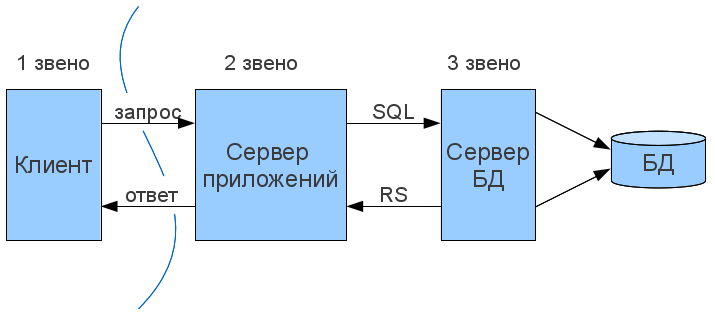


Рисунок 5 – Компоненты системы

Последовательность разработки системы:

1) Проектирование базы данных.

2) Создание логики серверной части.

3) Создание интерфейса.

4) Заполнение базы данных.

5) Тестирование.

# 3.2. Разработка пользовательского интерфейса

Разработку пользовательского интерфейса можно разделить на несколько этапов:

• Создание формы регистрации

• Создание формы для входа в аккаунт (профиль на сайте)

• Создание главной страницы сайта

• Создание страницы для товаров

• Создание адаптивного интерфейса

# 3.2.1 Создание макета интерфейса.

При проектировании макета сайта с помощью Figma активно использовалась возможность создавать элементы интерфейсов в качестве компонентов, такие как: AppBar (меню приложения), кнопки, карточки с информацией и т.д. Данный подход позволяет сократить время при повторном создании одинаковых компонентов, а так же вносить, при необходимости, изменения только в родительский компонент, тем самым изменив стиль всех его наследников.

В качестве демострации работы с компонентами (рис. 6, 7), продемострированна реализация темной темы для списка карточек с информацией о сдаваемых в аренду площадках. Слева отображено дерево элементов компонентов, родительский элемент отмечен заполненным ромбом.

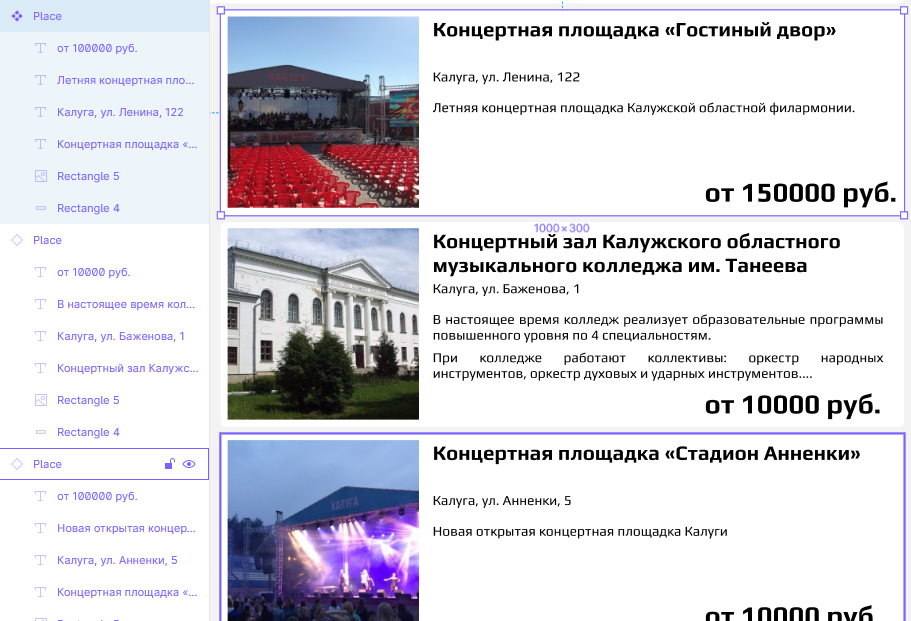


Рисунок 6 – Макет списка сдаваемых в аренду помещений.

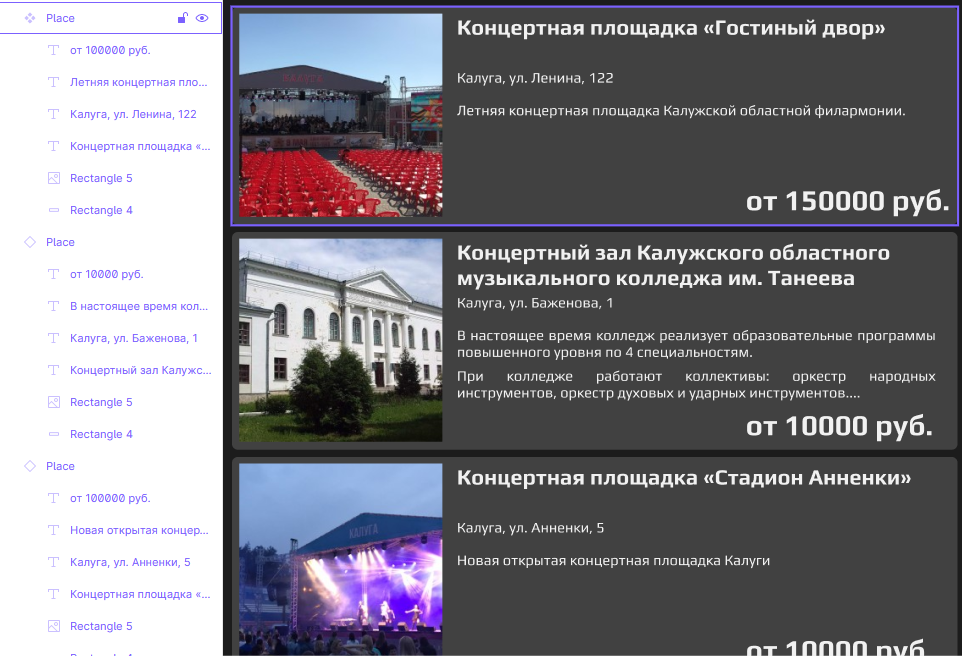


Рисунок 7 – Макет списка сдаваемых в аренду помещений (темная тема).

Как видно из приведенно выше примера компоненты наследники могут иметь отличное от родителей содержимое, но при этом наследуют его стили. В данном случае для создания темной версии карточек с информацией потребовалось всего два изменения у родительского элемента — изменение цвета текста и цвета фона.

# 3.2.2 Создание React приложения.

На основе разработанного макета были сформированы основные React-компоненты приложения. Для задания CSS стилей компонентов использовалась библиотека styled-components, что позволяет создать стилизованный компонент готовый к интеграции в DOM-модель, на основе уже существующих компонентов, либо простых HTML элементов.

Листинг 1 – описания стиля для меню приложения при помощи styled-components.

const AppBar = styled.div`

  background-color: #9e2020;

  height: 60px;

  width: 100%;

  margin: 0;

  display: -webkit-flex;

  display: flex;

`;



Рисунок 8 – Компонет AppBar.

На примере стартовой страницы приложения можно выделить следующие компоненты (рис. 9, 10):

* Header (AppBar) – содержит в себе основную навигацию и кнопку регистрации. Присутствует на всех страницах приложения.
* SearchArea – превью проводимых общественных мероприятий со строкой поиска.
* Description – краткое описание работы приложение.
* DescriptionCard – карточка содержащая в себе расширенную информацию об одной из возможности приложения.
* Footer – блок в нижней части страницы, куда вынесена полезная, но не первостепенная информация.

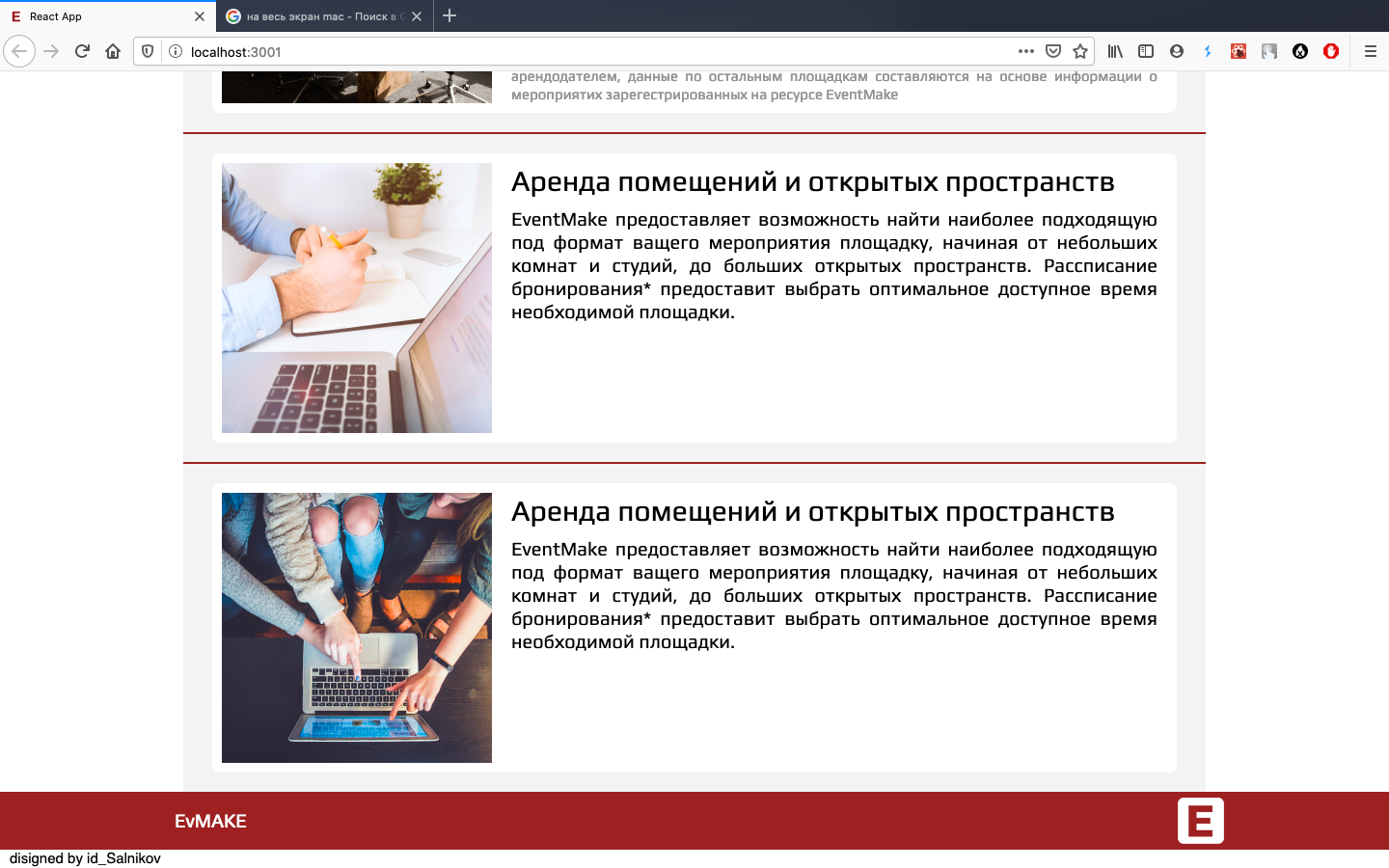


Рисунок 9 – Стартовая страницца сайта.

Листинг 2 – Основной компонент начальной страницы.

class Header extends Component {

  render() {

    return (

      <>

        <AppBar>

          <LogoP>EvMAKE</LogoP>

          <WrapBtn>

            <BtnText>Площадки</BtnText>

            <BtnText>Мероприятия</BtnText>

            <BtnText>О нас</BtnText>

          </WrapBtn>

          <WrapBtn end>

            <BtnText rounded>Войти</BtnText>

            <BtnText>

              <img src={accountImg} alt="AccountImg" />

            </BtnText>

          </WrapBtn>

        </AppBar>

        <SearchArea></SearchArea>

        <GridWithBottomRedLine></GridWithBottomRedLine>

        <Footer />

      </>

    );

  }

}

В качестве взаимодействия с веб-сервером использовался интерфейс JavaScript Fetch API, для работы с запросами и ответами HTTP. Он также предоставляет глобальный метод fetch(), который позволяет легко и логично получать ресурсы по сети асинхронно. При работе React-приложения данные необходимо получить до того как начнется построение интерфейса, для этого есть специальный метод componentDidMount(). В данном приложении данные передаются в формате JSON и записываются в состояние компонента (листинг 3).

Листинг 3 – Запрос данных у веб-сервера Node.

state ={

    data: []

  }

  componentDidMount() {

    fetch('http://localhost:4000/events/')

    .then(response => response.json())

    .then(jsonData => this.setState({data: jsonData}))

  }