Лабораторная работа
«Создание HTML-страницы – TypeScript, React и библиотека MUI »

Практическое задание. Используя TypeScript, React и библиотека MUI создать адаптивную страницу, которая включает:

- панель навигации;
- галерею рисунков;
- основной контент;
- футтер.

Задание 1. Создание проекта

Создать проект, предварительно импортировав необходимые модули.

Порядок выполнения практического задания

1. Создать проект **practicum_MUI** с помощью **Create React App**, указать, что будем использовать TypeScript. Также необходимо инсталлировать компоненты MUI. Для этого перейти в папку, в который Вы хотите создать проект, и в командной строке ввести:

```
npx create-react-app practicum_mui --template typescript
cd practicum_mui
npm install @mui/material @emotion/react @emotion/styled
npm install @mui/icons-material
npm start
```

В результате будет создан проект в папке **practicum_mui**. И в автоматически откроется вкладка (**http://localhost:3000/**) со стартовой страницей проекта React.

2. Будем работать с папкой **src** проекта. Добавить папки и перенести файлы, так чтобы получилась следующая структура проекта:

```
practicum_mui
|——— node_modules
|——— public
|——— src
|      |—— components
|      |
|      |—— styles
|      |    |——— index.css /* переместить из папки src */
|      |    |——— app.css /* переместить из папки src */
|      |
|      |——— images
|      |     |——— logo.svg /* переместить из папки src */
|      |
|      |——— App.tsx
|      |——— index.tsx
|      |——— data.tsx /* скопировать из архива к Лабораторной работе */
```

3. Внести изменения в файл **index.tsx** (исправить пути в импорте):

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './styles/index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(
  document.getElementById('root') as HTMLElement
);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);

reportWebVitals();
```

4. Внести изменения в файл **App.tsx** (исправить пути в импорте):

```
import React from 'react';
import logo from './images/logo.svg';
import './styles/App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p>
          Edit <code>src/App.tsx</code> and save to reload.
        </p>
        <a
          className="App-link"
          href="https://reactjs.org"
          target="_blank"
          rel="noopener noreferrer"
        >
          Learn React
        </a>
      </header>
    </div>
  );
}

export default App;
```

В результате после сохранения файлов в браузере по адресу **http://localhost:3000/** отобразиться стартовая страница React-проекта.

Задание 2. Панель навигации

Создать панель навигации, которая будет включать заголовок сайта, ссылки для перехода по страницам. При уменьшении экрана пункты меню должны сворачиваться в кнопку, по клику на которую будут раскрываться пункты меню (рисунок 1).

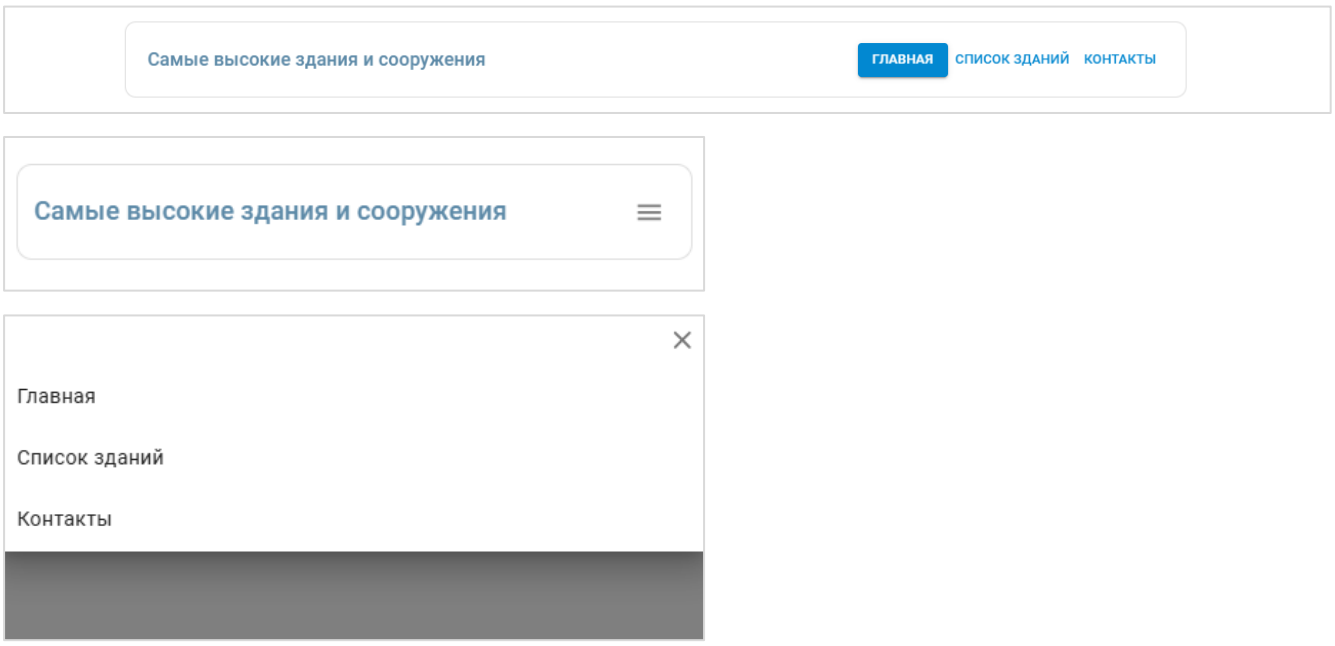

Рисунок 1. Панель навигации для разных размеров экрана

Порядок выполнения практического задания

1. Создать компонент **Navbar** в файле **Navbar.tsx** (папка **components**). Для создания панели будем использовать компоненты **AppBar** и **ToolBar**, импортируем их из компонент MUI:

```
import AppBar from '@mui/material/AppBar';
import Toolbar from '@mui/material/Toolbar';

function Navbar() {

    return (
        <AppBar>
            <Toolbar>

            </Toolbar>
        </AppBar>
      );
    }

export default Navbar;
```

2. В файле **App.tsx** вызовем этот компонент, предварительно импортировав его:

```
import Navbar from "./components/Navbar";


function App() {
  return (
    <div>
      <Navbar/>
    </div>
  );
}
```

```
export default App;
```

В результате страница будет выглядеть, как показано на рисунке 2.

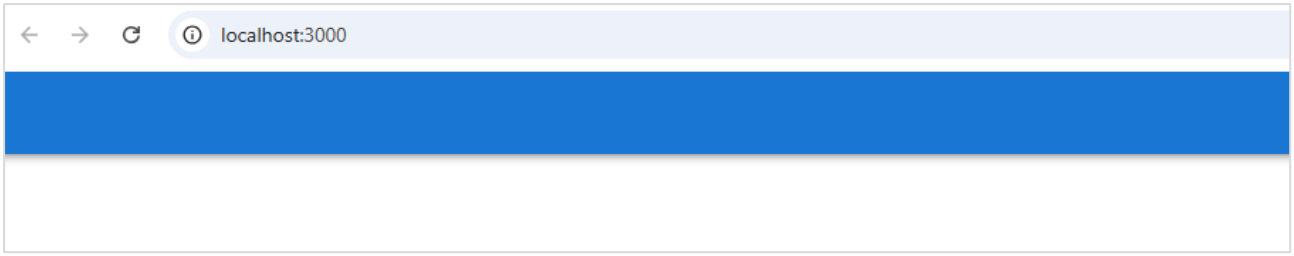

Рисунок 2. Навигационная панель

3. Сделаем прозрачным фон компонента **AppBar**. На основе компонента **ToolBar** создадим компонент **StyledToolBar**, чтобы изменить оформление этого элемента. Зафиксируем положение компонента на странице (он будет всегда располагаться на 28 пикселей ниже, чем верхняя граница окна браузера).

```
import AppBar from '@mui/material/AppBar';
import Toolbar from '@mui/material/Toolbar';
import { styled } from '@mui/material/styles';

const StyledToolbar = styled(Toolbar)(({ theme }) => ({
    display: 'flex',
    alignItems: 'center',
    justifyContent: 'space-between',
    flexShrink: 0,
    borderRadius: `calc(${theme.shape.borderRadius}px + 8px)`,
    border: '1px solid',
    borderColor: theme.palette.divider,
    padding: '8px 12px',
  }));

function Navbar() {

    return (
        <AppBar
          position="static"
          sx={{
            boxShadow: 0,
            bgcolor: 'transparent',
            mt: '28px',
          }}
        >
            <StyledToolbar>

            </StyledToolbar>
        </AppBar>
    );
}
```

В результате страница будет выглядеть, как показано на рисунке 3.

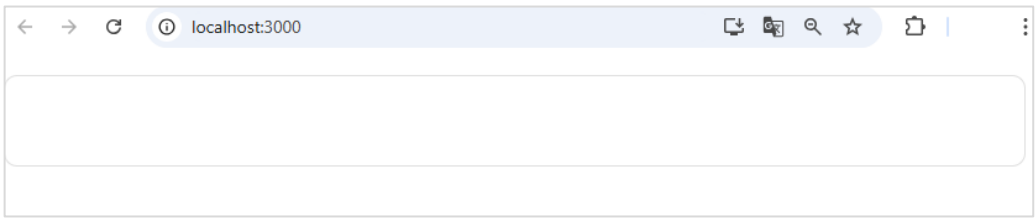

Рисунок 3. Навигационная панель со стилями

4. Чтобы навигационная панель занимала не всю ширину окна браузера, а адаптивно подстраивалась на различную ширину, например, не превышала размера контрольной точки **xl**, обернем ее в контейнер (импортируем компонент из MUI):

```
...
import Container from '@mui/material/Container';
...

function Navbar() {

    return (
        <AppBar
          ...
        >
          <Container maxWidth="xl">
            <StyledToolbar>

            </StyledToolbar>
          </Container>
        </AppBar>
    );
}
```

В результате ширина панели при изменении ширины окна браузера будет адаптивно изменяться, навигационная панель расположится по центру (рисунок 4).

Рисунок 4. Адаптивная навигационная панель

5. Добавим в панель заголовок сайта с помощью компонента **Typography**. Все компоненты перед использованием импортируем:

```
…
import Typography from '@mui/material/Typography';
…

function Navbar() {

    return (
        <AppBar
          ...
        >
          <Container maxWidth="xl">
            <StyledToolbar>
              <Typography variant="h6" sx={{ color: '#5d8aa8' }}>
                Самые высокие здания и сооружения
              </Typography>

            </StyledToolbar>
          </Container>
        </AppBar>
    );
}
```

Результат показан на рисунке 5.

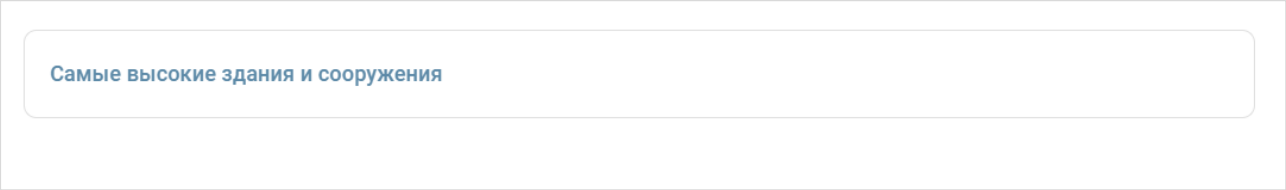

Рисунок 5. Навигационная панель с заголовком

6. Добавим пункты меню, которые обернем в компонент **box**. Каждый пункт меню опишем компонентом **Button**. Первый пункт меню выделим.

```
...
import Box from '@mui/material/Box';
import Button from '@mui/material/Button';
...
function Navbar() {

    return (
        <AppBar
          ...
        >
          <Container maxWidth="xl">
            <StyledToolbar>

              <Typography variant="h6" sx={{ color: '#5d8aa8' }}>
                Самые высокие здания и сооружения
              </Typography>

              <Box>
                <Button variant="contained" color="info" size="medium">
                  Главная
                </Button>
                <Button color="info" size="medium">
                  Список зданий
                </Button>
                <Button color="info" size="medium">
                  Контакты
                </Button>
              </Box>

            </StyledToolbar>
          </Container>
        </AppBar>
    );
}
```

В результате страница будет выглядеть, как показано на рисунке 6.

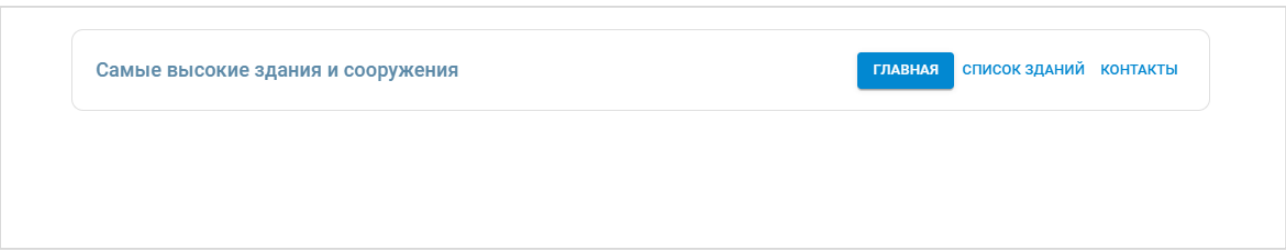

Рисунок 6. Навигационная панель с заголовком и меню

7. При достижении ширины экрана размера **xs** скроем пункты меню, вместо них выведем кнопку. При размерах **md** и шире кнопка будет скрыта, а меню показано. Для вывода кнопки будем использовать компоненты **IconButton** и **MenuIcon**.

```
…
import IconButton from '@mui/material/IconButton';
import MenuIcon from '@mui/icons-material/Menu';
…

function Navbar() {

    return (
        <AppBar
          ...
        >
          <Container maxWidth="xl">
            <StyledToolbar>
              <Typography variant="h6" sx={{ color: '#5d8aa8' }}>
                Самые высокие здания и сооружения
              </Typography>

              <Box sx={{ display: { xs: 'none', md: 'flex' } }}>
                <Button variant="contained" color="info" size="medium">
                  Главная
                </Button>
                <Button color="info" size="medium">
                  Список зданий
                </Button>
                <Button color="info" size="medium">
                  Контакты
                </Button>
              </Box>

              <Box sx={{ display: { xs: 'flex', md: 'none' }}}>
                <IconButton aria-label="Menu button">
                  <MenuIcon />
                </IconButton>

             </Box>

            </StyledToolbar>
          </Container>
        </AppBar>
      );
}
```

В результате при изменении ширины окна браузера до **xs** вместо меню отобразится кнопка (рисунок 7).

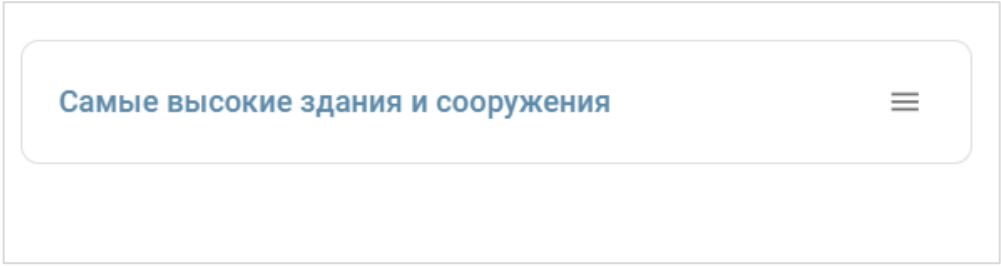

Рисунок 7. Навигационная панель с заголовком и свернутым меню

8. По клику на кнопку **Свернутое меню** необходимо «раскрыть» пункты меню. Для создания выпадающего меню воспользуемся компонентом **Drawer**, который представляет собой панель. Эту панель можно отобразить сверху экрана, при этом остальную часть страницы делает неактивной.

```
function Navbar() {

    return (
        <AppBar
          ...
        >
          <Container maxWidth="xl">
            <StyledToolbar>
             ...
              <Box sx={{ display: { xs: 'flex', md: 'none' }}}>
                <IconButton aria-label="Menu button">
                  <MenuIcon />
                </IconButton>

                <Drawer
                  anchor="top"
                  open={ true }
                >
                  <Box>
                    !!!! Выпадающее меню
                  </Box>
                </Drawer>

              </Box>

            </StyledToolbar>
          </Container>
        </AppBar>
      );
}
```

В результате сверху окна браузера для любого его размера будет отображаться панель с текстом «!!!Выпадающее меню» (рисунок 8).

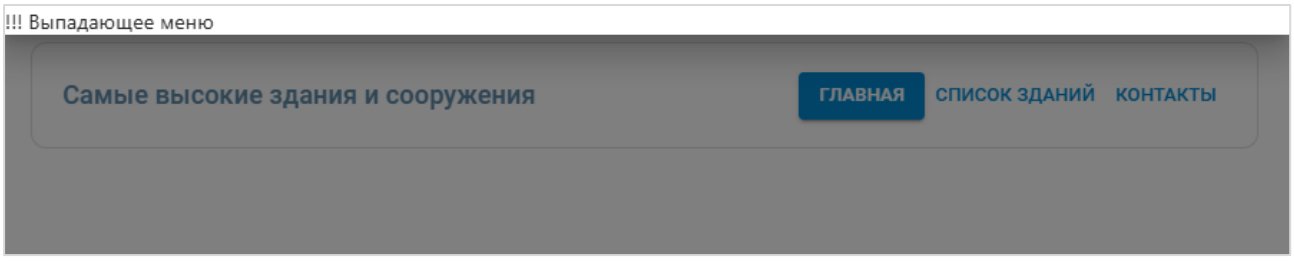

Рисунок 8. Навигационная панель с открытым выпадающим меню:

9. Чтобы управлять состоянием панели (открыта / закрыта) в компонент **Navbar** добавим состояние **open**, которое будет иметь значение **false**, если выпадающее меню скрыто, и **true** в противном случае. Также добавим функцию изменения состояния. Панель со свернутым меню будет выводиться только в том случае, если меню свернуто. При клике по кнопке **Свернутое меню** панель будет отображаться в браузере.

```
function Navbar() {
    const [open, setOpen] = React.useState(false);
```

```
    const toggleDrawer = (newOpen: boolean) => () => {
      setOpen(newOpen);
    };

    return (
        <AppBar
          ...
        >
          <Container maxWidth="xl">
            <StyledToolbar>
             ...
              <Box sx={{ display: { xs: 'flex', md: 'none' }}}>
                <IconButton aria-label="Menu button" onClick={toggleDrawer(true)}>
                  <MenuIcon />
                </IconButton>

                <Drawer
                  anchor="top"
                  open={ open }
                  onClose={toggleDrawer(false)}
                >
                  <Box>
                    !!!! Выпадающее меню
                  </Box>
                </Drawer>

              </Box>

            </StyledToolbar>
          </Container>
        </AppBar>
      );
}
```

В результате свернутая навигационная панель будет выглядеть, как показано на рисунке 9.

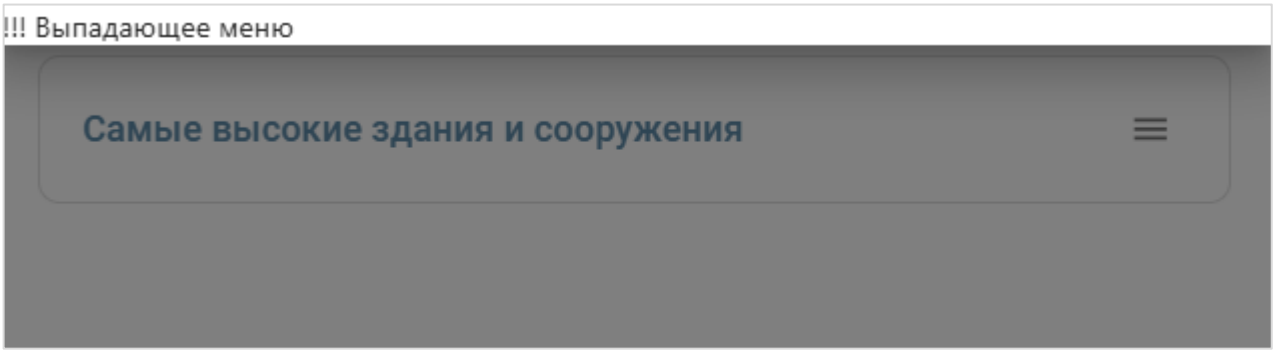

Рисунок 9. Навигационная панель после клика по кнопке Свернутое меню

10. Добавим пункты, которые будут отображаться в раскрытом меню. Для каждого пункта будем использовать компонент **MenuItem**:

```
...
import MenuItem from '@mui/material/MenuItem';
...

                <Drawer
                  anchor="top"
                  open={ open }
```

```
                onClose={toggleDrawer(false)}
              >
                <Box>
                 <MenuItem> Главная </MenuItem>
                 <MenuItem>Список зданий</MenuItem>
                 <MenuItem>Контакты</MenuItem>
                </Box>
              </Drawer>
...
```

В результате при клике по кнопке Свернутое меню будет отображаться выпадающее меню, как показано на рисунке 10.

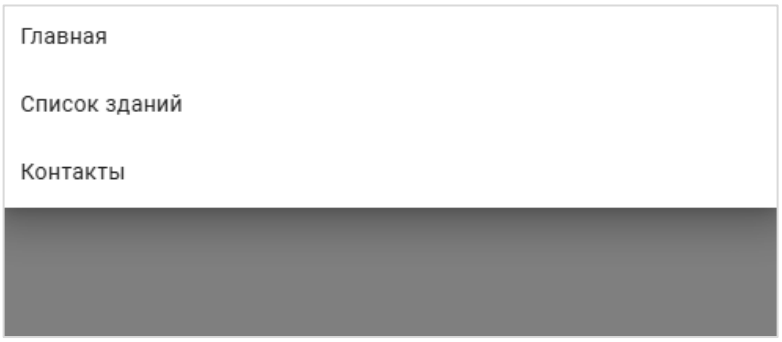

Рисунок 10. Навигационная панель с выпадающим меню

11. Закрыть выпадающее меню можно только по клику в произвольном месте страницы. Добавим кнопку, закрывающую меню. Кнопка будет выводиться над пунктами меню.

```
...
import CloseRoundedIcon from '@mui/icons-material/CloseRounded';
...

              <Drawer
                anchor="top"
                open={ open }
                onClose={toggleDrawer(false)}
              >
                <Box>
                  <Box
                    sx={{
                      display: 'flex',
                      justifyContent: 'flex-end',
                    }}
                  >
                    <IconButton onClick={toggleDrawer(false)}>
                     <CloseRoundedIcon />
                    </IconButton>
                  </Box>
                 <MenuItem> Главная </MenuItem>
                 <MenuItem>Список зданий</MenuItem>
                 <MenuItem>Контакты</MenuItem>
                </Box>
              </Drawer>
...
```

В результате навигационная панель будет отображать так, как показано на рисунке 2.

Самостоятельное задание

1. В выпадающем меню сделать первый пункт (**Главная**) выделенным (рисунок 11).

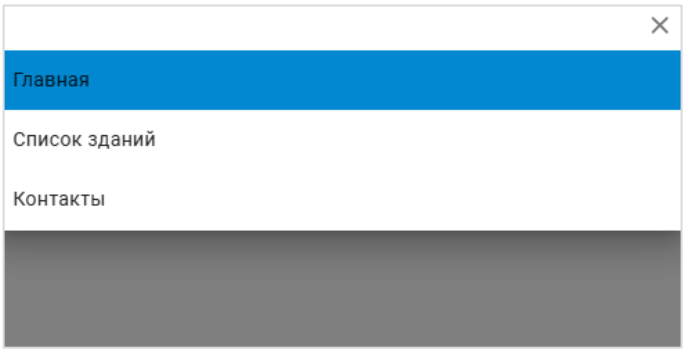

Рисунок 11. Свернутая навигационная панель с выделенным пунктом

2. При наведении курсора мыши на пункты свернутого меню изменить цвет этого пункта (рисунок 12):

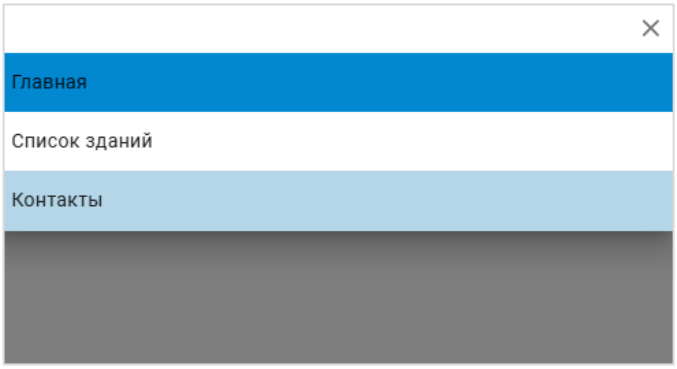

Рисунок 12. Свернутая навигационная панель при наведении на последний пункт

3. В компонент **`Navbar`** добавить пропс строкового типа, который делает активным пункт меню активным (**`variant="contained"`**), а остальные обычными (**`variant="text"`**). Если передана "1" – активной страницей становится **Главная**, если "2" – **Список зданий** и т.д.

Аналогичные изменения внести в выпадающее меню.

При вызове **`Navbar`** компонента в App добавить пропс со значением "1".

Пояснение. Чтобы описать пропс в компоненте **`Navbar`** необходимо тип пропса описать через **`interface`**:

```
interface ComponentProps {
    active: string;
}

function Navbar({ active } : ComponentProps) {
    ...
}
```

Задание 3. Галерея рисунков

На странице создать галерею рисунков, сделать ее адаптивной для различной ширины окна браузера (рисунок 13).

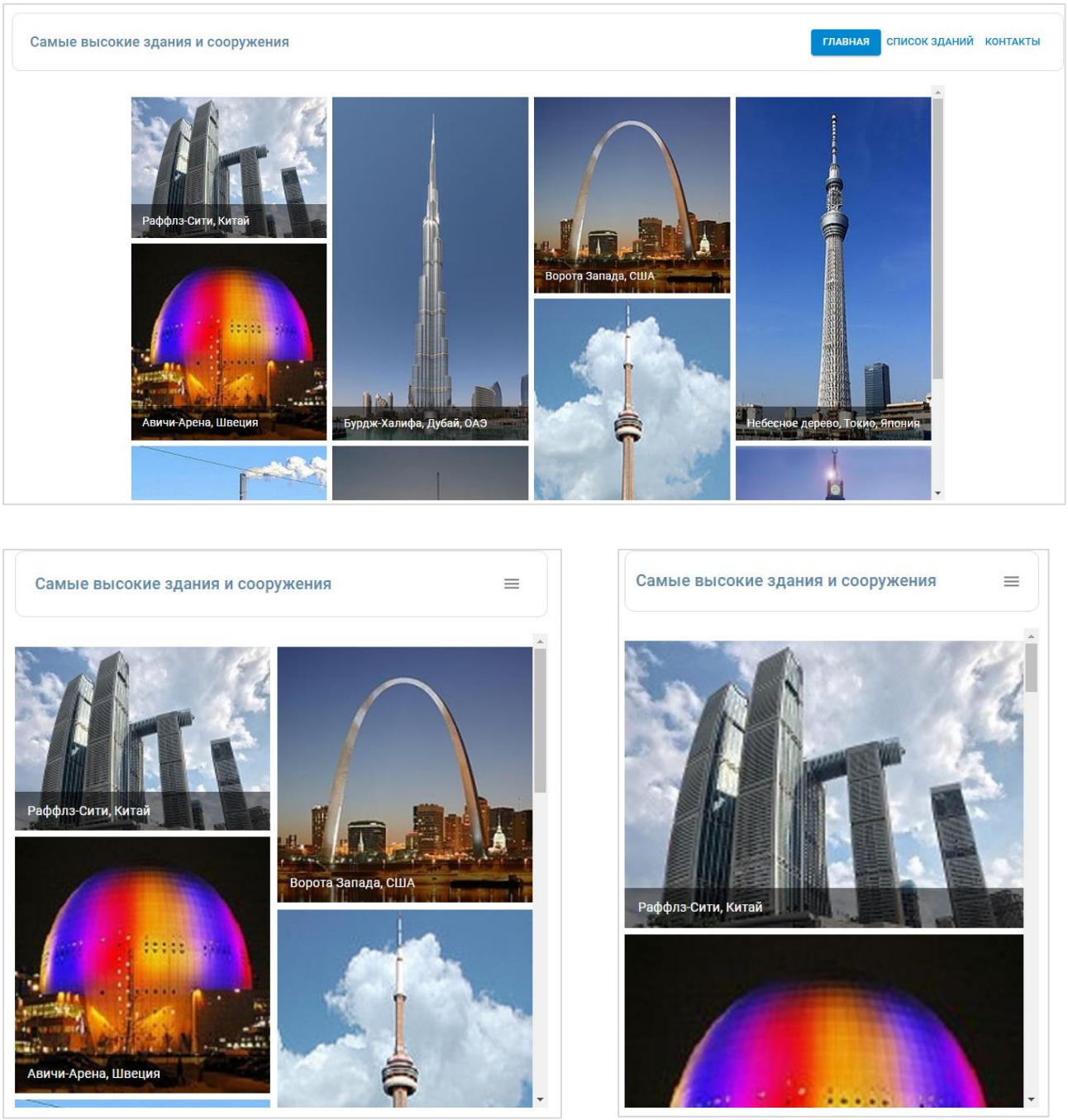

Рисунок 13. Адаптивная галерея

Порядок выполнения практического задания

1. В папку **`images`** вставить рисунки, необходимые для создания галереи (архив прикреплен к Лабораторной работе).

2. В файле **data.tsx** подготовлены данные для отображения на странице в виде массива **structure** (проверьте, чтобы этот файл из архива Лабораторной работы был скопирован в папку **src** проекта). Каждый элемент массива **structure** – объект, который включает название здания, рисунок и массив абзацев, с описанием здания или сооружения.

```
...
import Image6 from './images/image6.jpg';
...

const structures = [
    ...
    {
        img: Image6,
        title: "Ворота Запада, США",
        description: [
            `Арка в Сент-Луисе, также известная ...`,
            `Арка была спроектирована ...`
        ]
    },
    ...
]
```

Создать файл **Gallery.tsx** (папка **components**). В файле импортировать массив с данными **structures** из файла **data.tsx**. Удалить последний элемент массива и сохранить результат в **imgData**. Рисунки из этого массива будут использоваться при создании галереи.

```
import structures from "../data";

const imgData=structures.slice(0, -1);
```

3. Создать компонент **Gallery**. Будем использовать компоненты **ImageList** и **ImageListItem**. Компонент **ImageList** позволяет располагать рисунки в указанное количество столбцов (в нашем случае 4). Также можно указать интервал между рисунками и выбрать вариант отображения (в нашем случае **masonry**). С помощью **ImageListItem** отображается отдельный рисунок, выбранный из массива **imgData**.

```
import ImageList from '@mui/material/ImageList';
import ImageListItem from '@mui/material/ImageListItem';
import structures from "../data";

const imgData=structures.slice(0, -1);

function Gallery() {
  return (

        <ImageList variant="masonry" cols={ 4 } gap={ 8 }>
          {imgData.map((item) => (
            <ImageListItem key={ item.img }>
              <img
                srcSet={ item.img }
                src={ item.img }
                alt={ item.title }
                loading="lazy"
              />
            </ImageListItem>
          ))}
```

```
        </ImageList>
  );
}

export default Gallery;
```

Добавить в компонент **App** вызов компонента **Gallery**:

```
import Navbar from "./components/Navbar";
import Gallery from "./components/Gallery";

function App() {
  return (
    <div>
      <Navbar/>
      <Gallery/>
    </div>
  );
}
```

В результате галерея займет всю ширину страницы (рисунок 14).

Рисунок 14. Галерея рисунков

4. Обернем галерею в компонент **Box**, укажем для него фиксированную высоту и ширину, добавим полосы прокрутки, расположим по центру экрана, выведем компонент ниже меню:

```
...
import Box from '@mui/material/Box';
...

function Gallery() {
  return (

      <Box sx={{width: 800, height: 585, overflowY: 'scroll', m: '20px auto'}}>
        <ImageList variant="masonry" cols={ 4 } gap={ 8 }>
        ...
        </ImageList>
      </Box>
  );
}
```

В результате галерея на странице будет выглядеть, как показано на рисунке 15. При изменении ширины окна браузера ее ширина остается фиксированной.

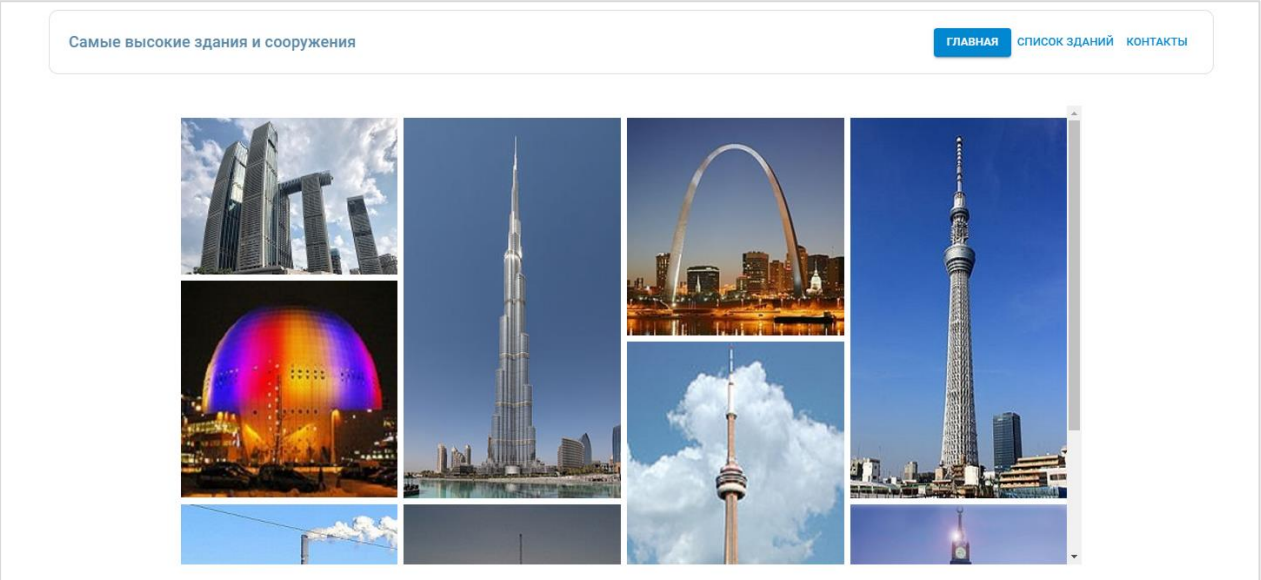

Рисунок 15. Галерея с фиксированной шириной и высотой

5. Сделаем галерею адаптивной. Для этого обернем ее в контейнер размера **lg**, удалим фиксированную ширину из **Box**, а вместо фиксированного количества колонок в **ImageList** добавим стили для четырех размеров экрана (на маленьком экране изображения перестоятся в одну колонку, на среднем в две (три), на большом в четыре).

```
...
import Container from '@mui/material/Container';
...

function Gallery() {
  return (
    <Container maxWidth="lg">
      <Box sx={{ height: 585, overflowY: 'scroll', m: '20px auto'}}>
        <ImageList
          variant="masonry"
          sx={{
            columnCount: {
              xs: '1 !important',
              sm: '2 !important',
              md: '3 !important',
              lg: '4 !important',
            },
          }}
          gap={8}>
        ...
        </ImageList>
      </Box>
    </Container>
  );
}
```

В результате галерея на странице для разных размеров экрана будет выглядеть, как показано на рисунке12 (без подписей к рисункам). При изменении ширины окна браузера – количество столбцов при выводе изображений будет изменяться.

6. Добавим подписи к рисункам с помощью компонентов **ImageListItemBar**, расположим их внизу картинок.

```
...
import ImageListItemBar from '@mui/material/ImageListItemBar';
...
          {imgData.map((item) => (
            <ImageListItem key={item.img}>
              <img
                srcSet={ item.img }
                src={ item.img }
                alt={ item.title }
                loading="lazy"
              />
              <ImageListItemBar position="bottom" title={ item.title } />
            </ImageListItem>
          ))}
...
```

В результате галерея будет выглядеть, как показано на рисунке 12.

Задание 4. Основной контент

В основном контенте на страницу вывести четыре карточки. В каждой карточке содержится информации об одном здании или сооружении: его название, изображение и краткое описание.

На больших экранах карточки должны выводиться в две колонки, на маленьких – в одну (рисунок 16).

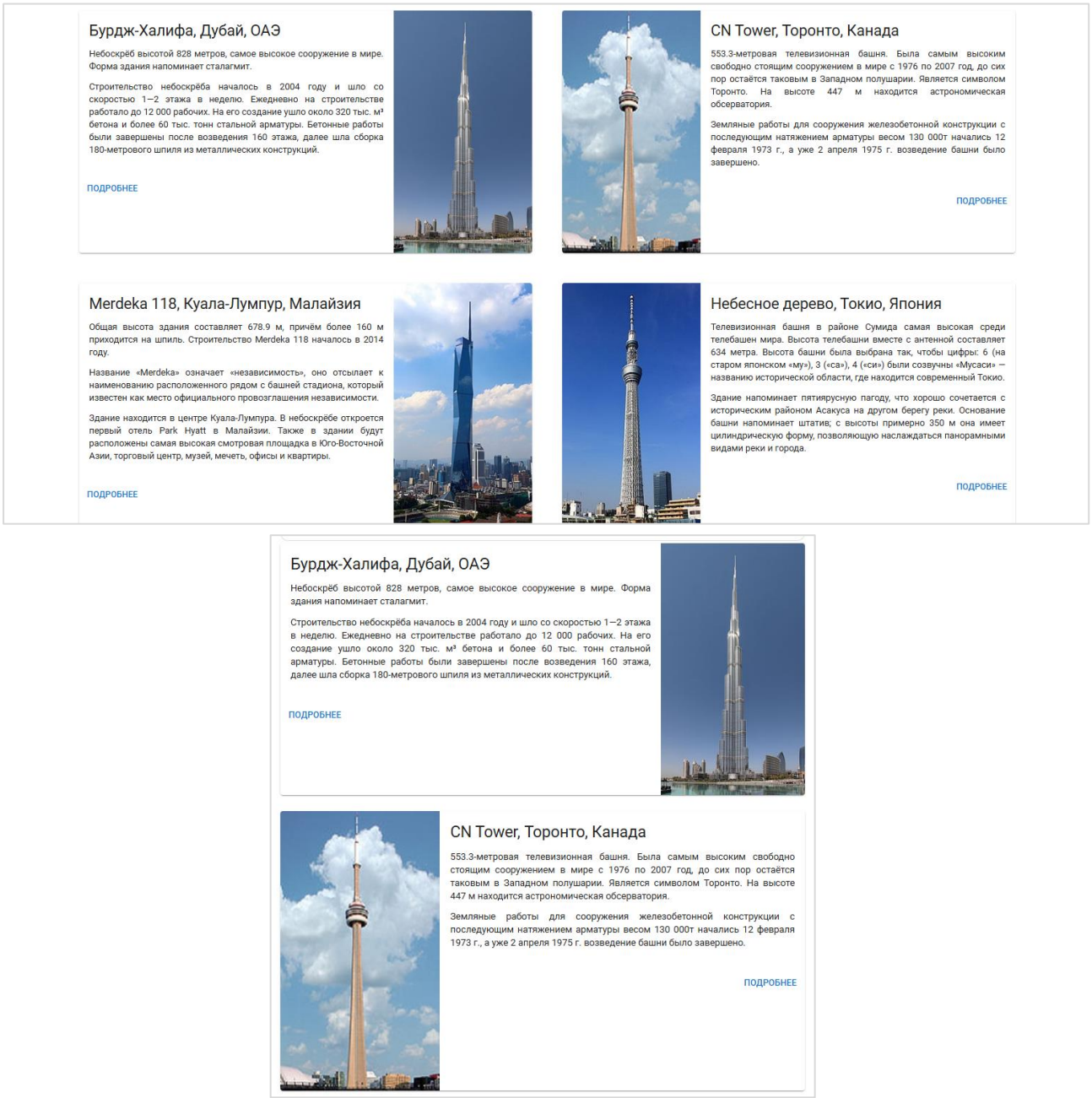

Рисунок 16. Основной контент страницы

Порядок выполнения практического задания

1. Создать файл **`Content.tsx`** (папка **`components`**). В файле импортировать массив данных для карточек из файла **`data.tsx`**. Каждый элемент массива **`structure`** – объект, который включает название здания, рисунок и массив абзацев, которые нужно вывести.

```
const structures = [
    ...
    {
        img: Image6,
        title: "Ворота Запада, США",
        description: [
            `Арка в Сент-Луисе, также известная ...`,
            `Арка была спроектирована ...`
        ]
    },
    ...
]
```

Для отображения на странице выделим четыре элемента массива.

```
import structures from "../data";

const cardData = [structures[3], structures[6], structures[9], structures[7]]
```

2. В файле **content.tsx** создадим компонент **Content**. Для вывода карточек в нем будем использовать компонент **Grid**, в котором для больших экранов определим два столбца, для маленьких – один. Обернем **Grid** в компонент **Contaner** размера **xl**. В каждый **Grid** элемент выведем название здания массива **cardData**.

```
import Container from '@mui/material/Container';
import Grid from '@mui/material/Grid';
import structures from "../data";

const cardData = [structures[3], structures[6], structures[9], structures[7]]

function Content() {
  return (
    <Container maxWidth="xl">
      <Grid container spacing={{ xs: 3, md: 6 }}>
        {cardData.map((item, index) => (
          <Grid key={index} size={{ xs: 12, md: 6 }} >
           { item.title }
          </Grid>
        ))}
      </Grid>
    </Container>
  );
}
export default Content;
```

Добавим в компонент **App** вызов компонента **Content**:

```
import Navbar from "./components/Navbar";
import Gallery from "./components/Gallery";
import Content from "./components/Content";

function App() {
  return (
    <div>
      <Navbar/>
      <Gallery/>
      <Content/>
    </div>
  );
}
```

В результате после галереи на странице отобразятся названия зданий (рисунок 17). При уменьшении ширины окна – названия перестроятся в один столбец.

Рисунок 17. Grid с названиями зданий

3. Создадим файл **BuildCard.tsx** (папка **components**), а в нем компонент **BuildCard**, в котором выведем информацию о здании, используя компонент MUI **Card**. В качестве пропса компонент будет получать объект с описанием одного здания.

```
import Card from '@mui/material/Card';
import CardActions from '@mui/material/CardActions';
import CardContent from '@mui/material/CardContent';
import CardMedia from '@mui/material/CardMedia';
import Button from '@mui/material/Button';
import Typography from '@mui/material/Typography';
import Box from '@mui/material/Box';


interface ComponentProps {
    building: {
        img: string,
        title: string,
        description: string[],
    };
  }
function BuildCard({ building} : ComponentProps) {
    return (
      <Card sx={{  display: 'flex' }}>
        <CardMedia
            component="img"
            alt={ building.title }
            image={ building.img }
        />
        <Box>
          <CardContent>
            <Typography gutterBottom variant="h5" >
              { building.title }
            </Typography>
            { building.description.map((item, ind) => (
              <Typography key={ind} variant="body2">
                { item }
              </Typography>
            ))}
          </CardContent>
          <CardActions sx={{ justifyContent: 'end'}} >
            <Button size="small">Подробнее</Button>
          </CardActions>
        </Box>
      </Card>
    )
}

export default BuildCard;
```

4. В компонент **Content** добавить вызов компонента **BuildCard:**

```
function Content() {
  return (
    <Container maxWidth="xl">
      <Grid container spacing={{ xs: 3, md: 6 }}>
        {cardData.map((item, index) => (
          <Grid key={index} size={{ xs: 12, md: 6 }} >
            <BuildCard building={ item } index={ index }/>
          </Grid>
        ))}
      </Grid>
    </Container>
  );
}
```

В результате основной контент страницы будет выглядеть, как показано на рисунке 18.

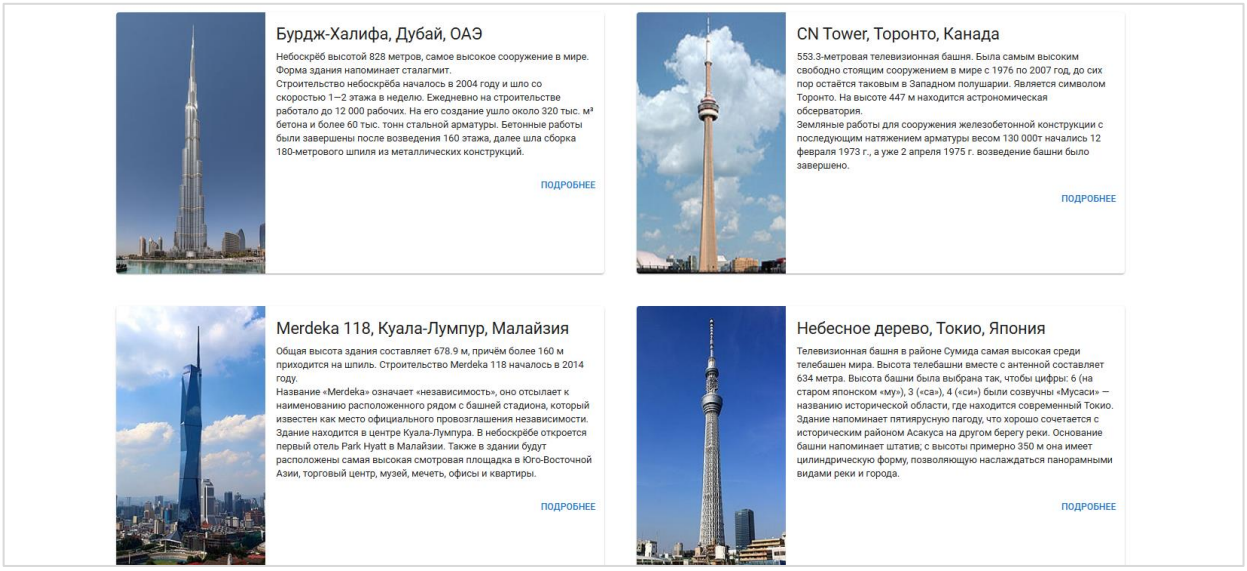

Рисунок 18. Основной контент страницы

Самостоятельные задания

1. Изменить компоненты **Content** и **BuildCard** так, чтобы страница выглядела, как показано на рисунке 16. Для этого:

- в файле **BuildCard.tsx** создать компонент **StyledTypography** для вывода описания здания (использовать **styled()**), для него установить:
  - цвет текста (**text.secondary**);
  - выравнивание по ширине;
  - отступы между абзацами.

Использовать этот компонент для вывода абзацев текста, описанных в поле **description**.

- в компонент **BuildCard** добавить props, в котором передать номер карточки из компонента **Content**;
- в зависимости от номера карточки установить порядок вывода рисунка в карточку и положение кнопки **Подробнее**.

2. Самостоятельно реализовать компонент **Footer** и включить его на страницу.