Практическое занятие 26: Material-UI - Подробное объяснение

Этот документ содержит подробные пошаговые объяснения для каждого примера из практической работы 26.

Установка Material-UI

Команда установки

```
npm install @mui/material @emotion/react @emotion/styled @mui/icons-material
```

Что устанавливаем:

- `@mui/material` - основная библиотека компонентов Material-UI
- `@emotion/react` и `@emotion/styled` - зависимости для стилизации (CSS-in-JS)
- `@mui/icons-material` - набор иконок Material Design

Зачем это нужно:

- Material-UI использует Emotion для стилизации компонентов
- без этих пакетов компоненты не будут работать
- иконки нужны для визуального улучшения интерфейса

Пример 1. Карточка технологии (SimpleTechCard.jsx)

Описание задачи

Создать карточку для отображения информации о технологии используя готовые компоненты Material-UI:

- Красивая карточка с тенью
- Типографика для текста
- Цветные чипы для категории и статуса
- Кнопки для действий

Пошаговое объяснение реализации

Шаг 1: Импорт необходимых компонентов MUI

```
import {
  Card,
  CardContent,
  CardActions,
```

```
  Typography,
  Button,
  Chip,
  Box
} from '@mui/material';
```

Что импортируем:

- `Card, CardContent, CardActions` - структура карточки (контейнер, содержимое, действия)
- `Typography` - компонент для текста с предустановленными стилями
- `Button` - кнопка с различными вариантами (contained, outlined, text)
- `Chip` - маленький компонент-метка для категорий и статусов
- `Box` - универсальный контейнер для flex/grid раскладок

Зачем это нужно:

- готовые компоненты с Material Design стилями
- не нужно писать CSS вручную
- единообразный дизайн во всем приложении

Шаг 2: Функции для определения цвета и текста статуса

```
const getStatusColor = (status) => {
  switch (status) {
    case 'completed': return 'success';
    case 'in-progress': return 'warning';
    default: return 'default';
  }
};

const getStatusText = (status) => {
  switch (status) {
    case 'completed': return 'Завершено';
    case 'in-progress': return 'В процессе';
    default: return 'Не начато';
  }
};
```

Что делаем:

- `getStatusColor` возвращает цвет MUI (success = зеленый, warning = оранжевый)
- `getStatusText` преобразует статус на английском в русский текст

Зачем это нужно:

- визуальная индикация статуса через цвет
- читаемый текст на русском языке
- централизованная логика преобразования

Шаг 3: Использование компонента Card

```
<Card sx={{ maxWidth: 345, margin: 2 }}>
  <CardContent>
    {/* содержимое карточки */}
  </CardContent>

  <CardActions>
    {/* кнопки действий */}
  </CardActions>
</Card>
```

Что делаем:

- `Card` - основной контейнер с тенью и скругленными углами
- `sx` prop - объект для inline стилей (Material-UI система стилизации)
- `CardContent` - область для основного содержимого
- `CardActions` - область для кнопок внизу карточки

Зачем это нужно:

- готовая структура карточки
- автоматические отступы и выравнивание
- `sx` позволяет быстро добавить кастомные стили

Шаг 4: Typography для типографики

```
<Typography variant="h5" component="h2" gutterBottom>
  {technology.title}
</Typography>

<Typography variant="body2" color="text.secondary" sx={{ mb: 2 }}>
  {technology.description}
</Typography>
```

Что делаем:

- `variant="h5"` - визуальный стиль (размер, вес шрифта)
- `component="h2"` - HTML-тег который будет отрендерен
- `gutterBottom` - добавляет отступ снизу
- `color="text.secondary"` - использует вторичный цвет текста из темы

Зачем это нужно:

- разделяем визуальный стиль и семантику HTML
- согласованная типографика во всем приложении

- доступность: правильная иерархия заголовков
- цвета из темы для единообразия

Шаг 5: Box для flex-раскладки чипов

```
<Box sx={{ display: 'flex', gap: 1, flexWrap: 'wrap' }}>
  <Chip
    label={technology.category}
    variant="outlined"
    size="small"
  />
  <Chip
    label={getStatusText(technology.status)}
    color={getStatusColor(technology.status)}
    size="small"
  />
</Box>
```

Что делаем:

- `Box` с flex-раскладкой и gap между элементами
- первый `Chip` с outline стилем для категории
- второй `Chip` с цветом в зависимости от статуса
- `size="small"` делает чипы компактными

Зачем это нужно:

- flex-раскладка с gap упрощает выравнивание
- `flexWrap: 'wrap'` переносит чипы на новую строку при необходимости
- визуальное различие: категория outlined, статус filled
- размер small экономит пространство

Шаг 6: Кнопки с условным рендерингом

```
<CardActions>
  {technology.status !== 'completed' && (
    <Button
      size="small"
      variant="contained"
      onClick={() => onStatusChange(technology.id, 'completed')}
    >
      Завершить
    </Button>
  )}

  <Button
    size="small"
```

```
      variant="outlined"
      onClick={() => onStatusChange(technology.id,
        technology.status === 'in-progress' ? 'not-started' : 'in-progress')}
    >
      {technology.status === 'in-progress' ? 'Приостановить' : 'Начать'}
    </Button>
  </CardActions>
```

Что делаем:

- показываем кнопку "Завершить" только если статус не 'completed'
- кнопка "Завершить" с `variant="contained"` (заполненная)
- кнопка переключения статуса с `variant="outlined"` (обведенная)
- текст кнопки меняется в зависимости от текущего статуса

Зачем это нужно:

- условный рендеринг убирает лишние кнопки
- визуальная иерархия: главное действие contained, второстепенное outlined
- динамический текст показывает следующее действие
- компактный размер small для экономии места

Полный исходный код SimpleTechCard.jsx

```
import React from 'react';
import {
  Card,
  CardContent,
  CardActions,
  Typography,
  Button,
  Chip,
  Box
} from '@mui/material';

// компонент карточки технологии с использованием Material-UI
function SimpleTechCard({ technology, onStatusChange }) {
  // функция определения цвета чипа в зависимости от статуса
  const getStatusColor = (status) => {
    switch (status) {
      case 'completed': return 'success';
      case 'in-progress': return 'warning';
      default: return 'default';
    }
  };

  // функция получения текста статуса на русском языке
  const getStatusText = (status) => {
    switch (status) {
      case 'completed': return 'Завершено';
```

```
      case 'in-progress': return 'В процессе';
      default: return 'Не начато';
    }
  };

  return (
    <Card sx={{ maxWidth: 345, margin: 2 }}>
      <CardContent>
        {/* заголовок карточки */}
        <Typography variant="h5" component="h2" gutterBottom>
          {technology.title}
        </Typography>

        {/* описание технологии */}
        <Typography variant="body2" color="text.secondary" sx={{ mb: 2 }}>
          {technology.description}
        </Typography>

        {/* чипы с категорией и статусом */}
        <Box sx={{ display: 'flex', gap: 1, flexWrap: 'wrap' }}>
          <Chip
            label={technology.category}
            variant="outlined"
            size="small"
          />
          <Chip
            label={getStatusText(technology.status)}
            color={getStatusColor(technology.status)}
            size="small"
          />
        </Box>
      </CardContent>

      {/* кнопки действий */}
      <CardActions>
        {technology.status !== 'completed' && (
          <Button
            size="small"
            variant="contained"
            onClick={() => onStatusChange(technology.id, 'completed')}
          >
            Завершить
          </Button>
        )}

        <Button
          size="small"
          variant="outlined"
          onClick={() => onStatusChange(technology.id,
            technology.status === 'in-progress' ? 'not-started' : 'in-progress')}
        >
          {technology.status === 'in-progress' ? 'Приостановить' : 'Начать'}
        </Button>
      </CardActions>
```

```
      </Card>
    );
  }

  export default SimpleTechCard;
```

Пример 2. Dashboard с вкладками (Dashboard.jsx)

Описание задачи

Создать панель управления с:

- Шапкой приложения (AppBar)
- Вкладками для переключения разделов
- Статистическими карточками
- Списками технологий
- Прогресс-баром

Пошаговое объяснение реализации

Шаг 1: Импорт компонентов и иконок

```
import {
  Box,
  AppBar,
  Toolbar,
  Typography,
  IconButton,
  Badge,
  Tabs,
  Tab,
  Card,
  CardContent,
  Grid,
  List,
  ListItem,
  ListItemText,
  LinearProgress
} from '@mui/material';

import {
  Notifications as NotificationsIcon,
  CheckCircle as CheckCircleIcon,
  Schedule as ScheduleIcon,
  TrendingUp as TrendingUpIcon
} from '@mui/icons-material';
```

Что импортируем:

Компоненты структуры:

- `AppBar, Toolbar` - шапка приложения
- `Tabs, Tab` - система вкладок
- `Grid` - сетка для раскладки (12-колоночная система)

Компоненты данных:

- `Card, CardContent` - карточки для статистики
- `List, ListItem, ListItemText` - списки
- `LinearProgress` - горизонтальный прогресс-бар

Компоненты UI:

- `IconButton` - кнопка-иконка
- `Badge` - бейдж (красный кружок с числом)

Иконки:

- `NotificationsIcon` - колокольчик для уведомлений
- `CheckCircleIcon, ScheduleIcon, TrendingUpIcon` - иконки для статистики

Зачем это нужно:

- AppBar создает профессиональную шапку
- вкладки организуют контент
- Grid упрощает адаптивную раскладку
- иконки улучшают визуальное восприятие

Шаг 2: Компонент TabPanel для содержимого вкладок

```
function TabPanel({ children, value, index }) {
  return (
    <div role="tabpanel" hidden={value !== index}>
      {value === index && <Box sx={{ p: 3 }}>{children}</Box>}
    </div>
  );
}
```

Что делаем:

- создаем компонент-обертку для содержимого вкладок
- `role="tabpanel"` для доступности
- `hidden={value !== index}` скрывает неактивные вкладки
- рендерим содержимое только если вкладка активна

Зачем это нужно:

- правильная семантика для скринридеров

- оптимизация: неактивный контент не рендерится
- единообразная структура всех вкладок
- отступы через `p: 3` (padding)

Шаг 3: Расчет статистики из данных

```
const stats = {
  total: technologies.length,
  completed: technologies.filter(t => t.status === 'completed').length,
  inProgress: technologies.filter(t => t.status === 'in-progress').length,
  notStarted: technologies.filter(t => t.status === 'not-started').length
};

const completionPercentage = stats.total > 0
  ? Math.round((stats.completed / stats.total) * 100)
  : 0;
```

Что делаем:

- подсчитываем общее количество технологий
- фильтруем по каждому статусу для подсчета
- вычисляем процент выполнения с проверкой деления на ноль
- округляем процент до целого числа

Зачем это нужно:

- динамическая статистика на основе реальных данных
- защита от деления на ноль при пустом массиве
- один объект stats для всех карточек статистики

Шаг 4: AppBar с Badge для уведомлений

```
<AppBar position="static" color="default" elevation={1}>
  <Toolbar>
    <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
      Панель управления технологиями
    </Typography>

    <IconButton color="inherit">
      <Badge badgeContent={notificationCount} color="error">
        <NotificationsIcon />
      </Badge>
    </IconButton>
  </Toolbar>
</AppBar>
```

Что делаем:

- `AppBar` с позицией static (не фиксированная)
- `elevation={1}` - небольшая тень
- `Toolbar` - контейнер для содержимого шапки
- `flexGrow: 1` растягивает заголовок влево
- `Badge` с красным кружком показывает количество уведомлений

Зачем это нужно:

- профессиональный вид шапки
- Toolbar обеспечивает правильные отступы
- flexGrow прижимает иконку к правому краю
- Badge визуально выделяет непрочитанные уведомления

Шаг 5: Tabs для переключения разделов

```
const [tabValue, setTabValue] = React.useState(0);

const handleTabChange = (event, newValue) => {
  setTabValue(newValue);
};

<Box sx={{ borderBottom: 1, borderColor: 'divider' }}>
  <Tabs value={tabValue} onChange={handleTabChange}>
    <Tab label="Обзор" />
    <Tab label="Статистика" />
  </Tabs>
</Box>
```

Что делаем:

- храним индекс активной вкладки в состоянии
- обработчик получает новое значение от MUI Tabs
- добавляем нижнюю границу для визуального разделения
- каждый Tab имеет метку (label)

Зачем это нужно:

- контролируемый компонент (controlled component)
- MUI автоматически управляет визуальным состоянием
- индикатор под активной вкладкой
- доступность из коробки

Шаг 6: Grid для адаптивной раскладки карточек

```
<Grid container spacing={3}>
  <Grid item xs={12} sm={6} md={3}>
    <Card>
      <CardContent>
        <Box sx={{ display: 'flex', alignItems: 'center', mb: 1 }}>
          <CheckCircleIcon color="success" sx={{ mr: 1 }} />
          <Typography color="text.secondary" variant="body2">
            Завершено
          </Typography>
        </Box>
        <Typography variant="h4">{stats.completed}</Typography>
      </CardContent>
    </Card>
  </Grid>
  {/* еще 3 карточки */}
</Grid>
```

Что делаем:

- `Grid container` создает контейнер сетки
- `spacing={3}` добавляет отступы между элементами
- `Grid item` с брейкпоинтами: `xs={12} sm={6} md={3}`
- иконка и текст в flex-контейнере с выравниванием
- крупное число статистики с variant="h4"

Зачем это нужно:

- адаптивность:
  - xs (мобильный): 1 карточка в строку (12/12)
  - sm (планшет): 2 карточки в строку (6/12)
  - md+ (десктоп): 4 карточки в строку (3/12)
- spacing автоматически рассчитывает отступы
- цветные иконки для визуального кодирования

Шаг 7: LinearProgress для визуализации прогресса

```
<Card>
  <CardContent>
    <Typography color="text.secondary" variant="body2" gutterBottom>
      Общий прогресс
    </Typography>
    <Typography variant="h4" gutterBottom>
      {completionPercentage}%
    </Typography>
    <LinearProgress
      variant="determinate"
      value={completionPercentage}
      sx={{ height: 8, borderRadius: 4 }}
```

```
      />
    </CardContent>
  </Card>
```

Что делаем:

- `LinearProgress` с типом determinate (определенный прогресс)
- `value` передаем процент (0-100)
- увеличиваем высоту до 8px для лучшей видимости
- скругляем углы прогресс-бара

Зачем это нужно:

- визуальное представление процента
- determinate показывает конкретное значение (не анимацию загрузки)
- кастомная высота делает бар заметнее
- borderRadius для современного вида

Шаг 8: List для отображения технологий

```
<List>
  {technologies.slice(0, 5).map((tech) => (
    <ListItem key={tech.id}>
      <ListItemText
        primary={tech.title}
        secondary={tech.category}
      />
    </ListItem>
  ))}
</List>
```

Что делаем:

- берем первые 5 технологий с помощью `slice(0, 5)`
- используем `ListItem` для каждого элемента
- `ListItemText` с primary (основной текст) и secondary (вторичный текст)

Зачем это нужно:

- готовый стиль списков Material Design
- автоматические отступы и выравнивание
- primary крупнее и жирнее, secondary светлее
- ограничение до 5 элементов для компактности

Шаг 9: Фильтрация и отображение по категориям

```
{['frontend', 'backend', 'database', 'ui-library', 'other'].map(category => {
  const count = technologies.filter(t => t.category === category).length;
  return count > 0 ? (
    <ListItem key={category}>
      <ListItemText
        primary={category}
        secondary={`${count} технологий`}
      />
    </ListItem>
  ) : null;
})}
```

Что делаем:

- проходим по массиву категорий
- для каждой категории подсчитываем количество технологий
- отображаем только категории с count > 0
- показываем категорию и количество

Зачем это нужно:

- динамическая группировка по категориям
- не показываем пустые категории
- информативный вторичный текст с количеством

Полный исходный код Dashboard.jsx

```
import React from 'react';
import {
  Box,
  AppBar,
  Toolbar,
  Typography,
  IconButton,
  Badge,
  Tabs,
  Tab,
  Card,
  CardContent,
  Grid,
  List,
  ListItem,
  ListItemText,
  LinearProgress
} from '@mui/material';
import {
  Notifications as NotificationsIcon,
  CheckCircle as CheckCircleIcon,
  Schedule as ScheduleIcon,
```

```
  TrendingUp as TrendingUpIcon
} from '@mui/icons-material';

// компонент для содержимого вкладки
function TabPanel({ children, value, index }) {
  return (
    <div role="tabpanel" hidden={value !== index}>
      {value === index && <Box sx={{ p: 3 }}>{children}</Box>}
    </div>
  );
}

function Dashboard({ technologies }) {
  const [tabValue, setTabValue] = React.useState(0);
  const [notificationCount] = React.useState(3);

  // расчет статистики на основе массива technologies
  const stats = {
    total: technologies.length,
    completed: technologies.filter(t => t.status === 'completed').length,
    inProgress: technologies.filter(t => t.status === 'in-progress').length,
    notStarted: technologies.filter(t => t.status === 'not-started').length
  };

  // расчет процента выполнения
  const completionPercentage = stats.total > 0
    ? Math.round((stats.completed / stats.total) * 100)
    : 0;

  // обработчик переключения вкладок
  const handleTabChange = (event, newValue) => {
    setTabValue(newValue);
  };

  return (
    <Box sx={{ flexGrow: 1 }}>
      {/* шапка приложения */}
      <AppBar position="static" color="default" elevation={1}>
        <Toolbar>
          <Typography variant="h6" component="div" sx={{ flexGrow: 1 }}>
            Панель управления технологиями
          </Typography>

          {/* иконка уведомлений с бейджем */}
          <IconButton color="inherit">
            <Badge badgeContent={notificationCount} color="error">
              <NotificationsIcon />
            </Badge>
          </IconButton>
        </Toolbar>
      </AppBar>

      {/* вкладки */}
      <Box sx={{ borderBottom: 1, borderColor: 'divider' }}>
```

```
          <Tabs value={tabValue} onChange={handleTabChange}>
            <Tab label="Обзор" />
            <Tab label="Статистика" />
          </Tabs>
        </Box>

        {/* вкладка обзора */}
        <TabPanel value={tabValue} index={0}>
          <Grid container spacing={3}>
            {/* статистические карточки */}
            <Grid item xs={12} sm={6} md={3}>
              <Card>
                <CardContent>
                  <Box sx={{ display: 'flex', alignItems: 'center', mb: 1 }}>
                    <CheckCircleIcon color="success" sx={{ mr: 1 }} />
                    <Typography color="text.secondary" variant="body2">
                      Завершено
                    </Typography>
                  </Box>
                  <Typography variant="h4">{stats.completed}</Typography>
                </CardContent>
              </Card>
            </Grid>

            <Grid item xs={12} sm={6} md={3}>
              <Card>
                <CardContent>
                  <Box sx={{ display: 'flex', alignItems: 'center', mb: 1 }}>
                    <ScheduleIcon color="warning" sx={{ mr: 1 }} />
                    <Typography color="text.secondary" variant="body2">
                      В процессе
                    </Typography>
                  </Box>
                  <Typography variant="h4">{stats.inProgress}</Typography>
                </CardContent>
              </Card>
            </Grid>

            <Grid item xs={12} sm={6} md={3}>
              <Card>
                <CardContent>
                  <Box sx={{ display: 'flex', alignItems: 'center', mb: 1 }}>
                    <TrendingUpIcon color="info" sx={{ mr: 1 }} />
                    <Typography color="text.secondary" variant="body2">
                      Не начато
                    </Typography>
                  </Box>
                  <Typography variant="h4">{stats.notStarted}</Typography>
                </CardContent>
              </Card>
            </Grid>

            <Grid item xs={12} sm={6} md={3}>
              <Card>
```

```
              <CardContent>
                <Typography color="text.secondary" variant="body2" gutterBottom>
                  Общий прогресс
                </Typography>
                <Typography variant="h4" gutterBottom>
                  {completionPercentage}%
                </Typography>
                <LinearProgress
                  variant="determinate"
                  value={completionPercentage}
                  sx={{ height: 8, borderRadius: 4 }}
                />
              </CardContent>
            </Card>
          </Grid>

          {/* недавно добавленные технологии */}
          <Grid item xs={12} md={6}>
            <Card>
              <CardContent>
                <Typography variant="h6" gutterBottom>
                  Недавно добавленные
                </Typography>
                <List>
                  {technologies.slice(0, 5).map((tech) => (
                    <ListItem key={tech.id}>
                      <ListItemText
                        primary={tech.title}
                        secondary={tech.category}
                      />
                    </ListItem>
                  ))}
                </List>
              </CardContent>
            </Card>
          </Grid>

          {/* распределение по категориям */}
          <Grid item xs={12} md={6}>
            <Card>
              <CardContent>
                <Typography variant="h6" gutterBottom>
                  По категориям
                </Typography>
                <List>
                  {['frontend', 'backend', 'database', 'ui-library',
'other'].map(category => {
                    const count = technologies.filter(t => t.category ===
category).length;
                    return count > 0 ? (
                      <ListItem key={category}>
                        <ListItemText
                          primary={category}
                          secondary={`${count} технологий`}
```

```
                        />
                      </ListItem>
                    ) : null;
                  })}
                </List>
              </CardContent>
            </Card>
          </Grid>
        </Grid>
      </TabPanel>

      {/* вкладка статистики */}
      <TabPanel value={tabValue} index={1}>
        <Typography variant="h4" gutterBottom>
          Детальная статистика
        </Typography>
        <Grid container spacing={3}>
          <Grid item xs={12}>
            <Card>
              <CardContent>
                <Typography variant="h6" gutterBottom>
                  Общая информация
                </Typography>
                <Typography>Всего технологий: {stats.total}</Typography>
                <Typography>Завершено: {stats.completed}</Typography>
                <Typography>В процессе: {stats.inProgress}</Typography>
                <Typography>Не начато: {stats.notStarted}</Typography>
                <Typography sx={{ mt: 2 }}>
                  Процент выполнения: {completionPercentage}%
                </Typography>
              </CardContent>
            </Card>
          </Grid>
        </Grid>
      </TabPanel>
    </Box>
  );
}

export default Dashboard;
```

Настройка темы Material-UI

Создание кастомной темы

```
import { ThemeProvider, createTheme } from '@mui/material/styles';
import CssBaseline from '@mui/material/CssBaseline';

const muiTheme = createTheme({
  palette: {
```

```
    primary: {
      main: '#1976d2',
    },
    secondary: {
      main: '#dc004e',
    },
  },
});

function App() {
  return (
    <ThemeProvider theme={muiTheme}>
      <CssBaseline />
      {/* остальное приложение */}
    </ThemeProvider>
  );
}
```

Что делаем:

- создаем тему с помощью `createTheme`
- определяем primary и secondary цвета
- оборачиваем приложение в `ThemeProvider`
- добавляем `CssBaseline` для нормализации стилей

Зачем это нужно:

- единый цветовой набор во всем приложении
- все MUI компоненты используют эту тему
- можно легко менять цвета во всем приложении
- CssBaseline убирает браузерные различия в стилях

Самостоятельная работа

Задание 1: Создайте компонент уведомлений с использованием Snackbar из MUI. Компонент должен корректно отображаться на всех размерах экрана, иметь понятные иконки и интерактивные элементы. Реализуйте различные типы уведомлений (success, error, warning, info) с автоматическим закрытием. Убедитесь, что компонент корректно работает на всех размерах экрана и имеет доступные и понятные интерактивные элементы.

Задание 2: Добавьте переключение темы (светлая/тёмная) с использованием ThemeProvider из MUI. Тема должна применяться ко всем компонентам приложения, включая модальные окна, формы и навигацию. Сохраняйте выбранную тему в localStorage для сохранения предпочтений пользователя. Убедитесь, что тема корректно переключается и сохраняется между перезагрузками страницы.

Задание 3: Проверьте адаптивность и корректность работы всех компонентов MUI. Убедитесь, что модальные окна правильно открываются и закрываются, адаптивный дизайн работает на различных размерах экрана (мобильные, планшеты, десктоп), и все интерактивные элементы доступны и понятны. Проверьте, что все компоненты корректно отображаются и работают на всех размерах экрана.

Обратите внимание, что данная практика - последняя в блоке по React и последняя для выполнения соответствующей контрольной работы №4.