

23. Учебный проект: меняй-удаляй (часть 2)

Рабочая ветка `module7-task2`

Задание

Продолжаем расширять функциональность нашего приложения. Нам осталось решить одну задачу — научиться показывать статистику.

Смена экранов

У нас в приложении несколько экранов, и нужно научиться корректно их переключать.

1. Создайте компонент для экрана со статистикой. Разметку вы найдёте в папке `markup`. Обратите внимание, что по умолчанию компонент должен быть показан, поэтому удалите в разметке CSS-класс, который его скрывает (если он там уже есть).
2. Подключите в `main.js` и отрисуйте компонент со статистикой. Пока что доска с задачами и статистика покажутся одна под другой, сейчас мы с этим разберёмся.
3. Реализуйте в `AbstractComponent` методы `show` и `hide` для показа и скрытия компонента соответственно. Для этого будет

достаточно добавлять и удалять с корневого элемента CSS-класс, который его скрывает.

4. Аналогично допишите в `BoardController` методы, которые умеют скрывать и показывать его. Для этого будет достаточно добавлять и удалять с корневого элемента CSS-класс.
5. Теперь, когда в `main.js` есть все необходимые компоненты, реализуйте здесь логику переключения экранов при выборе соответствующего пункта в меню.

Обратите внимание, что при переключении на экран статистики и обратно сбрасывается выбранная сортировка.

Безжалостная статистика

Начнём с программирования статистики.

В ней отображается сводная информация о выполненных задачах.

Для удобства просмотра и анализа эта информация представлена в виде графиков и диаграмм. Построить диаграммы полностью самостоятельно — нетривиальная задача, поэтому мы воспользуемся пакетом `chart.js`.

1. Установите из npm пакет `chart.js` и плагин `chartjs-plugin-datalabels`.
2. Импортируйте `chart.js` и `chartjs-plugin-datalabels` в модуль, который будет отвечать за формирование статистики.
3. При помощи пакета `chart.js` отрисуйте:
 - график выполненных задач;
 - круговую диаграмму «Done by: Colors».
4. Подробное описание каждой диаграммы есть в техническом задании.
5. Подключите для поля выбора периода `flatpickr`. Настройте его таким образом, чтобы пользователь мог выбрать диапазон: дату начала и дату окончания периода за который требуется получить статистику. При переходе на экран статистики в поле ввода должен подставляться период по умолчанию. Подробности в техническом задании.

Обратите внимание, фильтры на графики и диаграммы никак не влияют.

Статистика всегда должна быть актуальной!

25. Личный проект: меняй-удаляй (часть 2)

Задание

Продолжаем расширять функциональность нашего приложения. Нам осталось решить одну задачу — научиться показывать статистику.

Смена экранов

У нас в приложении несколько экранов, и нужно научиться корректно их переключать.

Киноман

1. Создайте компонент для экрана со статистикой. Разметку вы найдёте в папке `markup`. Обратите внимание, что по умолчанию компонент должен быть показан, поэтому удалите в разметке CSS-класс, который его скрывает (если он там уже есть).
2. Подключите в `main.js` и отрисуйте компонент со статистикой. Пока что список фильмов и статистика покажутся друг под другом, сейчас мы с этим разберёмся.
3. Реализуйте в `AbstractComponent` методы `show` и `hide` для показа и скрытия компонента соответственно. Для этого будет достаточно добавлять и удалять с корневого элемента CSS-класс, который его скрывает.
4. Аналогично допишите в `PageController` методы, которые умеют скрывать и показывать его. Для этого будет достаточно добавлять и удалять с корневого элемента CSS-класс.
5. Теперь, когда в `main.js` есть все необходимые компоненты, реализуйте здесь логику переключения экранов при выборе соответствующего пункта в меню.

Обратите внимание, что при переключении на экран статистики и обратно сбрасывается выбранная сортировка.

Безжалостная статистика

Начнём с программирования статистики.

В ней отображается сводная информация о предпочтениях пользователя.

Для удобства просмотра и анализа эта информация представлена в виде графиков и диаграмм. Построить диаграммы полностью самостоятельно — нетривиальная задача, поэтому мы воспользуемся пакетом `chart.js`.

1. Установите из npm пакет `chart.js` и плагин `chartjs-plugin-datalabels`.
2. Импортируйте `chart.js` и `chartjs-plugin-datalabels` в модуль, который будет отвечать за формирование статистики.
3. При помощи пакета `chart.js` отрисуйте:
 - диаграмму, которая показывает количество просмотренных фильмов в разрезе жанров.
4. Подробное описание каждой диаграммы есть в техническом задании.

Обратите внимание, что статистику над диаграммой не нужно выводить с помощью `chart.js`, её нужно реализовать простым подставлением данных в шаблон.

4. Чтобы упростить вам задачу, ваш коллега написал код самой диаграммы, но не успел его встроить в проект. Закончите за него работу: разберитесь в чужом коде и настройте вывод актуальной информации. Если возникнут трудности, подробное описание всех параметров вы найдёте в документации к [chart.js](#).

5. Код от коллеги:

```
const BAR_HEIGHT = 50;
const statisticCtx =
document.querySelector(`.statistic__chart`);

// Обязательно рассчитайте высоту canvas, она
зависит от количества элементов диаграммы
statisticCtx.height = BAR_HEIGHT * 5;

const myChart = new Chart(statisticCtx, {
  plugins: [ChartDataLabels],
  type: `horizontalBar`,
  data: {
    labels: [`Sci-Fi`, `Animation`, `Fantasy`,
`Comedy`, `TV Series`],
    datasets: [{
      data: [11, 8, 7, 4, 3],
      backgroundColor: `#ffe800`,
      hoverBackgroundColor: `#ffe800`,
      anchor: `start`
    }]
  }
});
```

```
},
options: {
  plugins: {
    datalabels: {
      font: {
        size: 20
      },
      color: `#ffffff`,
      anchor: 'start',
      align: 'start',
      offset: 40,
    }
  },
  scales: {
    yAxes: [{
      ticks: {
        fontColor: `#ffffff`,
        padding: 100,
        fontSize: 20
      },
      gridLines: {
        display: false,
        drawBorder: false
      },
      barThickness: 24
    }],
    xAxes: [{
      ticks: {
        display: false,
        beginAtZero: true
      },
      gridLines: {
        display: false,
        drawBorder: false
      }
    }],
  }
}
```



```
        },  
        },  
    },  
    legend: {  
        display: false  
    },  
    tooltips: {  
        enabled: false  
    }  
}  
});
```

6. Настройте выбор периода, за который нужно отобразить статистику. Графики должны обновляться при изменении периода в поле выбора периода.

Обратите внимание, фильтры на графики и диаграммы никак не влияют.

Статистика всегда должна быть актуальной!

Пример диаграмм

All movies

Watchlist 13

History 4

Favorites 8

Stats

Your rank Sci-Fighter

Show stats: All time Today Week Month Year

You watched

22 movies

Total duration

130 h 22 m

Top genre

Sci-Fi



Bigtrip

1. Создайте компонент для экрана со статистикой. Разметку вы найдёте в папке `markup`. Обратите внимание, что по умолчанию компонент должен быть показан, поэтому удалите в разметке CSS-класс, который его скрывает (если он там уже есть).
2. Подключите в `main.js` и отрисуйте компонент со статистикой. Пока что список точек маршрута и статистика покажутся друг под другом, сейчас мы с этим разберёмся.
3. Реализуйте в `AbstractComponent` методы `show` и `hide` для показа и скрытия компонента соответственно. Для этого будет достаточно добавлять и удалять с корневого элемента CSS-класс, который его скрывает.
4. Аналогично допишите в `TripController` методы, которые умеют скрывать и показывать его. Для этого будет достаточно добавлять и удалять с корневого элемента CSS-класс.
5. Теперь, когда в `main.js` есть все необходимые компоненты, реализуйте здесь логику переключения экранов при выборе соответствующего пункта в меню.

Обратите внимание, что при переключении на экран статистики и обратно сбрасывается выбранная сортировка.

Безжалостная статистика

Начнём с программирования статистики.

В ней отображается сводная информация о путешествии.

Для удобства просмотра и анализа эта информация представлена в виде графиков и диаграмм. Построить диаграммы полностью самостоятельно — нетривиальная задача, поэтому мы воспользуемся пакетом `chart.js`.

1. Установите из npm пакет `chart.js` и плагин `chartjs-plugin-datalabels`.
2. Импортируйте `chart.js` и `chartjs-plugin-datalabels` в модуль, который будет отвечать за формирование статистики.
3. При помощи пакета `chart.js` отрисуйте:
 - диаграмму «Money»;
 - диаграмму «Transport»;

- диаграмму «Time spend».
4. Подробное описание каждой диаграммы есть в техническом задании.
 5. Чтобы упростить вам задачу, ваш коллега написал код самой диаграммы, но не успел его встроить в проект. Закончите за него работу: разберитесь в чужом коде и настройте вывод актуальной информации. Если возникнут трудности, подробное описание всех параметров вы найдёте в документации к [chart.js](#).
 6. Код от коллеги

```
const moneyCtx =  
document.querySelector(`.statistic__money`);  
const transportCtx =  
document.querySelector(`.statistic__transport`);  
const timeSpendCtx =  
document.querySelector(`.statistic__time-spend`);
```

```
// Рассчитаем высоту канваса в зависимости от  
того, сколько данных в него будет передаваться  
const BAR_HEIGHT = 55;  
moneyCtx.height = BAR_HEIGHT * 6;  
transportCtx.height = BAR_HEIGHT * 4;  
timeSpendCtx.height = BAR_HEIGHT * 4;
```

```
const moneyChart = new Chart(moneyCtx, {  
  plugins: [ChartDataLabels],
```

```
type: `horizontalBar`,
data: {
  labels: [`FLY`, `STAY`, `DRIVE`, `LOOK`,
`RIDE`],
  datasets: [{
    data: [400, 300, 200, 160 , 100],
    backgroundColor: `#ffffff`,
    hoverBackgroundColor: `#ffffff`,
    anchor: `start`
  }]
},
options: {
  plugins: {
    datalabels: {
      font: {
        size: 13
      },
      color: `#000000`,
      anchor: 'end',
      align: 'start',
      formatter: (val) => `€ ${val}`
    }
  },
  title: {
    display: true,
    text: `MONEY`,
    fontColor: `#000000`,
    fontSize: 23,
    position: `left`
  },
  scales: {
    yAxes: [{
      ticks: {
        fontColor: `#000000`,
```

```

        padding: 5,
        fontSize: 13,
    },
    gridLines: {
        display: false,
        drawBorder: false
    },
    barThickness: 44,
}],
xAxes: [{
    ticks: {
        display: false,
        beginAtZero: true,
    },
    gridLines: {
        display: false,
        drawBorder: false
    },
    minBarLength: 50
}],
},
legend: {
    display: false
},
tooltips: {
    enabled: false,
}
}
});

```

```

const transportChart = new Chart(transportCtx, {
    plugins: [ChartDataLabels],
    type: `horizontalBar`,
    data: {

```

```

labels: ['FLY', 'DRIVE', 'RIDE'],
datasets: [{
  data: [4, 2, 1],
  backgroundColor: '#ffffff',
  hoverBackgroundColor: '#ffffff',
  anchor: 'start'
}],
},
options: {
  plugins: {
    datalabels: {
      font: {
        size: 13
      },
      color: '#000000',
      anchor: 'end',
      align: 'start',
      formatter: (val) => `${val}x`
    }
  },
  title: {
    display: true,
    text: 'TRANSPORT',
    fontColor: '#000000',
    fontSize: 23,
    position: 'left'
  },
  scales: {
    yAxes: [{
      ticks: {
        fontColor: '#000000',
        padding: 5,
        fontSize: 13,
      },
    }],
  },
}

```



```

        gridLines: {
            display: false,
            drawBorder: false
        },
        barThickness: 44,
    }],
    xAxes: [{
        ticks: {
            display: false,
            beginAtZero: true,
        },
        gridLines: {
            display: false,
            drawBorder: false
        },
        minBarLength: 50
    }],
},
legend: {
    display: false
},
tooltips: {
    enabled: false,
}
}
});

```

Обратите внимание, фильтры на графики и диаграммы никак не влияют.

Статистика всегда должна быть актуальной!

Пример диаграмм

MONEY



TRANSPORT



TIME SPENT

