1 лекция

1.1

Задание состоит из нескольких шагов:

1. Сделайте форк личного проекта. Если в разделе личного проекта нет ссылки на репозиторий, то надо немного подождать, пока наш робот её создаст.

2. Затем клонируйте форк личного проекта себе на компьютер.

3. В форке личного проекта создайте ветку `module1-task1`.

4. В этой ветке внесите изменения в файл `Readme.md`: укажите имя вашего наставника.

5. Сделайте коммиты ваших изменений локально, а затем синхронизируйте локальный и удалённый репозитории (ваш форк).

6. Создайте пулреквест из ветки `module1-task1` вашего форка в мастер-репозиторий и прикрепите задание с помощью кнопки «Найти пулреквест», расположенной над этим заданием в блоке деталей проекта.

7. Нажмите зелёную кнопку «Всё выполнил, можно проверять», расположенную сразу после этого задания.

1.2

В этом задании разомнёмся в использовании **ES2015** и вспомним, как программировать: мы напишем код, который найдёт на странице все шаблоны, которые представляют все доступные экраны в приложения и сделаем так, чтобы при помощи клавиш-стрелок эти экраны переключались между собой

На этом шаге вам не нужно программировать какое-то сложное поведение по переключению экранов, мы сделаем это в последующих заданиях. Прямо сейчас наша задача заключается в том, чтобы понять, как выглядит каждый из шагов приложения и попрактиковаться в **ES2015**

Подготовка

Проверьте, что у вас:

- Установлены nodejs 10 и npm

Для комфортной работы с проектом мы так же рекомендуем установить:

- Плагин editorconfig
- Плагин eslint для вашего редактора

Для работы с проектом вам пригодятся следующие команды:

- `npm install` — эта команда скачивает и устанавливает все необходимые зависимости для вашего проекта (необходимо выполнить как минимум один раз)
- `npm run eslint` — эта команда запускает проверки **ESLint** локально
- `npm start` — эта команда собирает ваш проект в папку **build** и открывает страницу с собранным проектом

В форке личного проекта создайте ветку `module1-task2` и в этой ветке выполните следующие шаги

Задание

1. Подключите файл `js/main.js` с помощью тега `<script/>` в конце файла `index.html`
2. Соберите ссылки на все возможные DOM-элементы экранов приложения в массив

   Все экраны записаны как отдельные шаблоны `template` в `index.html`.

   Обратите внимание

   Экраны в исходном коде разметки могут идти в произвольной последовательности и не соответствовать той, в которой они описаны в техническом задании, обратите на этот момент особое внимание: переключаться экраны должны в соответствии с ТЗ
3. Создайте функцию, которая по переданному номеру будет показывать экран из массива, созданного в прошлом задании
   - Пиксель хантер
   - Угадай мелодию

   Все содержимое шаблона, описывающего экран приложения, должно заменять содержимое блока `#main`. Обратите внимание на то, что некоторые блоки могут повторяться, например `footer` или частично `header`. В будущем, при желании, вы сможете оптимизировать переключение экранов, сделав частичное обновление элементов, пока же этого делать не нужно

---

4. Покажите первый экран приложения. Какой из экранов приложения является приветственным, написано в техническом задании
5. Добавьте обработчик клавиатурных событий на `document`, который будет по нажатию на клавиши-стрелок `←` и `→` переключать экраны на предыдущий и следующий соответственно

   Обратите внимание

   Нажатия на клавиши не должны отменять системных клавиатурных действий — страница должна перезагружаться по нажатию на `Ctrl + R`(*`Cmd + R`*), вкладки браузера должны переключаться, клавиатурная навигация — работать

6. Добавьте на страницу визуальные стрелки, которые будут дублировать поведение с клавиатуры и помогут переключать экраны мышкой:

Для Пиксельхантер:

Вставьте при помощи *javascript* перед закрывающимся тегом `<body>` следующий контент:

```
<div class="arrows__wrap">
  <style>
    .arrows__wrap {
      position: absolute;
      top: 95px;
      left: 50%;
      margin-left: -56px;
    }
    .arrows__btn {
      background: none;
      border: 2px solid black;
      padding: 5px 20px;
    }
  </style>
  <button class="arrows__btn"><-</button>
  <button class="arrows__btn">-></button>
</div>
```

Для Угадай мелодию:

Вставьте при помощи *javascript* последним элементом в `<div class="app">` следующий контент:

```
<div class="arrows__wrap">
    <style>
      .arrows__wrap {
        position: absolute;
        top: 135px;
        left: 50%;
        margin-left: -56px;
      }
      .arrows__btn {
        background: none;
        border: 2px solid black;
        padding: 5px 20px;
      }
    </style>
    <button class="arrows__btn"><-</button>
    <button class="arrows__btn">-></button>
</div>
```

Повесьте на них правильные обработчики событий нажатия так, чтобы при клике на соответствующий элемент показывался следующий или предыдущий экран соответственно.

2 лекция

В этом разделе мы перенесём все шаблоны из `index.html` в отдельные JS-модули, чтобы избавиться от монолитного кода.

Задача

1. Доработайте сборку проекта так, чтобы зависимости в JS собирались при помощи сборщика модулей Rollup. Для этого выполните следующие шаги:
    1. Установите в качестве npm-зависимости **Rollup**: `npm i gulp-better-rollup -DE`
    2. Установите в качестве npm-зависимости инструмент для создания сурсмапов (*sourcemap*): `npm i gulp-sourcemaps -DE`
    3. Подключите **Rollup** как зависимость в своём `gulpfile.js`: `const rollup = require(`gulp-better-rollup`);`
    4. Подключите **sourcemaps** как зависимость в своём `gulpfile.js`: `const sourcemaps = require(`gulp-sourcemaps`);`
    5. В `gulpfile.js` доработайте задачу **scripts** следующим образом:

```
gulp.task(`scripts`, () => {
  return gulp.src(`js/main.js`)
    .pipe(plumber())
    .pipe(sourcemaps.init())
      .pipe(rollup({}, `iife`))
      .pipe(sourcemaps.write(``))
      .pipe(gulp.dest(`build/js`));
  });
```

2. Настройте линтер (**ESLint**) так, чтобы он работал с вашими файлами как с модулями **ES2015**, для этого укажите в начале файла `.eslintrc.yml`:

```
parserOptions:
  ecmaVersion: 2015
  sourceType: 'module'
```

После того, как вы включили поддержку модулей в вашем проекте, директива `'use strict';` вам больше не нужна.

3. Создайте модуль с функцией, которая будет создавать DOM-элемент на основе переданной в виде строки разметки. Функция должна принимать на вход строку с разметкой и возвращать DOM-элемент. Пример, как может работать функция:

```
  // в myDiv будет записан DOM-элемент
const myDiv = getElementFromTemplate(`<div>Разметка DOM-элемента</div>`
);
```

4. Разбейте на модули шаблоны из `index.html` используя строки-шаблоны (Template literal). Перенесите код блоков, описанных в `template`, в отдельные модули и экспортируйте из этих модулей DOM-элементы, созданные с помощью функции из третьего пункта этого задания.

Пример того, как у вас должно получиться. Скажем, у вас в разметке есть шаблон `module-1`:

```
<!-- index.html -->

<template id="module-1">
    <div>Разметка исходного шаблона</div>
```

```
</template>
```

В этом случае вам нужно создать модуль `module-1.js` примерно со следующим содержанием:

```
// module-1.js
const moduleOneElement = getElementFromTemplate(`<div>Разметка исходного шаблона</div>`);
export default moduleOneElement;
```

где функция `getElementFromTemplate` создаёт DOM-элемент на основе переданной ей разметки. В итоге у вас должны получиться следующие модули:

Для ПиксельХантер:

1. Главный экран, на основе блока `#intro`.
2. Экран приветствия, блок `#greeting`.
3. Экран правил игры, блок `#rules`.
4. Экран вопроса первого типа, блок `#game-1`.
5. Экран вопроса второго типа, блок `#game-2`.
6. Экран вопроса третьего типа, блок `#game-3`.
7. Экран с результатами, блок `#stats`.

Для Угадай мелодию:

1. Главный экран, на основе блока `#welcome`.
2. Экран вопроса первого типа, блок `#game-genre`.
3. Экран вопроса второго типа, блок `#game-artist`.
4. Экран результатов, когда игрок выиграл блок `#result-success`.
5. Экран результатов, когда вышло время блок `#fail-time`.
6. Экран результатов, когда превышено число попыток блок `#fail-tries`.

Имена модулей, файлов и переменных могут называться произвольно, использовать название из примера необязательно.

5.Доработайте модули так, чтобы переключение между разделами происходило не с помощью нажатия на кнопки-стрелки, а из самих модулей.

1. Вынесите из `main.js` в отдельный модуль функцию, выводящую переданный блок на страницу.
2. Уберите из модуля `main.js` всю логику переключения между экранами и кнопки переключения экранов.
3. Доработайте созданные в прошлом задании модули игровых экранов. Экспортируйте из каждого модуля DOM-элемент экрана и добавьте в каждый из модулей логику, которая включает следующий игровой экран.

Экраны должны переключаться следующим образом:

Для Пиксель:

Первый экран, блок `#intro`, должен показываться из модуля `main.js` при загрузке страницы.

Экран приветствия, блок `#greeting`, должен показываться по нажатию на символ звёздочки, элемент `.intro__asterisk` на главном экране.

Экран с правилами игры, блок `#rules`, должен показываться по нажатию на блок со стрелкой вправо, элемент `.greeting__continue` на экране приветствия.

Экран первой игры, блок `#game-1`, должен показываться по отправке формы на экране правил игры. Кнопка отправки `.rules__button`. Кнопка отправки должна быть отключена, `disabled`, пока в поле с именем игрока ничего не введено.

Экран второй игры, блок `#game-2`, должен показываться после того, как будут выбраны оба варианта ответа на экране `#game-1`.

Экран третьей игры, блок `#game-3`, должен показываться по нажатию на любой ответ на втором игровом экране, любой блок `.game__answer`.

Экран с результатами, блок `#stats`, должен показываться по нажатию на любой ответ на третьем игровом экране, любой блок `.game__option`.

Нажатие на кнопку «Назад» в левом верхнем углу должно с любого экрана возвращать на экран приветствия.

Для Угадай:

Первый экран `#welcome` должен показываться из модуля `main.js` при загрузке страницы.

Экран первой игры `#game-genre` должен показываться по нажатию на кнопку Play — блок `.welcome__button` на главном экране.

Экран второй игры `#game-artist` должен показываться по нажатию на кнопку «Ответить», блок `.game__submit`. Кнопка «Ответить» должна быть отключена, `disabled`, пока не выбран ни один из возможных вариантов ответа.

Экран результатов `#result-success` должен показываться по нажатию на любой из вариантов ответа — блок `.game__artist` на игровом экране.

Какой из последних экранов: экран поражения или экран результатов показывать, определяйте случайным образом.

Кнопка «Сыграть ещё раз» на последнем экране (экран результатов или экран поражения) должна открывать первый экран.

Нажатие на кнопку «Угадай Мелодию» в левом верхнем углу должно с любого экрана возвращать на экран приветствия.

3 лекция

3.1

В этом задании мы подключим все необходимые для тестирования библиотеки и настроим наше окружение.

Задача

Подключить, скачать и настроить все необходимые зависимости.

1. Установим плагин Mocha для нашей Gulp конфигурации в наш проект: `npm i --save-dev --save-exact gulp-mocha`.
2. Добавим поддержку загрузки модулей ES2015 `import/export` из **Node.js** в наши тесты:
    - `npm i rollup-plugin-commonjs -DE`
3. Установим assert-библиотеку Chai:

    ```
    npm i chai -DE
    ```

4. Настроим запуск тестов в нашей сборке. Для этого нужно настроить нашу задачу, которая запускает тесты `gulpfile.js`:

    ```
    const mocha = require(`gulp-mocha`);                 // Добавим установленный gulp-mocha плагин
    const commonjs = require(`rollup-plugin-commonjs`); // Добавим плагин для работы с `commonjs` модулями
    gulp.task(`test`, function () {
      return gulp
      .src([`js/**/*.test.js`])
          .pipe(rollup({
     plugins: [
          commonjs()           // Сообщает Rollup, что модули можно загружать из node_modules
        ]}, `cjs`))            // Выходной формат тестов – `CommonJS` модуль
      .pipe(gulp.dest(`build/test`))
      .pipe(mocha({
        reporter: `spec`       // Вид в котором я хочу отображать результаты тестирования
      }));
       });
    ```

5. Создадим первый тест `js/data/game-data.test.js`, который сообщит нам, что всё работает.

    ```
    import {assert} from 'chai';

    describe(`Array`, () => {
      describe(`#indexOf()`, () => {
        it(`should return -1 when the value is not present`, () => {
          assert.equal(-1, [1, 2, 3].indexOf(4));
        });
      });
    });
    ```

6. Добавим в `ESLint` информацию о том, что у нас в проекте используется фреймворк **Mocha**. Для этого добавим `mocha: true` в настройки `.eslint.yml`:

```
parserOptions:
  ecmaVersion: 2015
  sourceType: 'module'

env:
  es6: true
  browser: true
  commonjs: true
  mocha: true

extends: 'htmlacademy/es6'
```

7. Попробуем запустить наши тесты: `npm test`

Обратите внимание

Чтобы тесты запускались, когда вы вызываете команду `npm test`, нужно убедиться что, объект `"scripts"` из `package.json` имеет следующий вид:

```
"scripts": {
    "build": "gulp build",
    "start": "gulp serve",
    "eslint": "eslint ./gulpfile.js js/", // Запустить проверки eslint на пап
ке js/ и файле ./gulpfile.js
    "test": "npm run eslint && npm run stylelint && gulp test"  // Запустить
скрипт eslint
                                                // если eslint не упал с ошибкой,
то запустить npm run stylelint
                                                // и если он не сломается, то запу
стить gulp test
  },
```

3.2

В этом задании мы опишем поведение наших игровых объектов и попрактикуемся в использовании TDD.

Используя подход TDD, опишите работу функций подсчёта результата игры, игровых объектов: жизней, уровней и т.д.

Для Пиксель

Функция подсчёта очков при окончании игры:

- Функция на вход принимает массив ответов пользователя;
- Функция на вход принимает кол-во оставшихся жизней;
- Функция на выходе отдаёт кол-во набранных очков.

Массив ответов пользователя

Массив ответов должен хранить в себе данные об ответах пользователя на каждый вопрос по порядку — информацию об успешном или неуспешном ответе и времени, затраченном на ответ.
Правила подсчёта очков в соответствии с Техническим Заданием:

- Правильный ответ: 100 очков;
- Быстрый ответ: добавляется 50 очков;
- Медленный ответ: снимается 50 очков;
- За каждую оставшуюся к концу игры жизнь: дополнительные 50 очков.

Начните решение с описания тестовых случаев. Например:

- Если игрок ответил меньше, чем на 10 вопросов, то игра считается не пройденной и функция должна вернуть **-1**;
- Если игрок ответил на все вопросы и не быстро, и не медленно, и у него остались все жизни, то функция должна вернуть **1150** очков;
- Комбинируйте разные условия ответов и кол-ва жизней, чтобы описать максимальное кол-во вариантов.

Реализуйте работу функции и убедитесь, что все тесты проходят.

По такому же алгоритму опишите остальные бизнес-функции вашей игры:

- Управление жизнями игрока
- Переключение уровней
- Отсчёт времени

При написании этих функций помните о «чистоте» функции и о неизменяемости входных данных.

Обратите внимание

Все примеры реализаций, имена функций, параметров и методов даны в качестве примера и не обязательны для повторения.

Для Угадай:

Функция подсчёта набранных баллов игрока:

- Функция на вход принимает массив ответов пользователя;
- Функция на вход принимает кол-во оставшихся нот;
- Функция на выходе отдаёт кол-во набранных очков;

Массив ответов пользователя.

Массив ответов должен хранить в себе данные об ответах пользователя на каждый вопрос по порядку — информацию об успешном или неуспешном ответе и времени, затраченном на ответ.
Правила подсчёта баллов игрока в соответствии с Техническим Заданием:

- За правильный ответ 1 балл;
- За быстрый правильный ответ (менее 30 секунд) — 2 балла;
- За каждую ошибку вычитается 2 балла.

Начните решение с описания тестовых случаев. Например:

- Если игрок ответил меньше, чем на 10 вопросов, то игра считается не пройденной и функция должна вернуть **-1**;
- Если игрок ответил на все вопросы правильно и не быстро, и ни разу не ошибся, то функция должна вернуть **10** баллов;
- Комбинируйте разные условия ответов и кол-ва жизней, чтобы описать максимальное кол-во вариантов.

Реализуйте работу функции и убедитесь, что все тесты проходят проверки.

Функция вывода результата игрока:

- Функция на вход принимает массив результатов игр других игроков;
- Функция на вход принимает объект результата с кол-вом набранных баллов, кол-вом оставшихся нот и кол-вом оставшегося времени;
- Функция на выходе выдаёт строку результата:
    - Если игрок выиграл, то его результат должен быть выведен в виде фразы: *Вы заняли i место из t игроков. Это лучше, чем у n% игроков*, где **i** — место, которое занял пользователь, **t** — общее кол-во игроков, **n** — процент успеха игрока;
    - Если игрок проиграл и у него закончилось время, то должна быть выведена фраза: «Время вышло! Вы не успели отгадать все мелодии»;
    - Если игрок проиграл и у него закончились попытки, то должна быть выведена фраза: «У вас закончились все попытки. Ничего, повезёт в следующий раз!».

Алгоритм вывода результата игрока:

- Если игрок не смог пройти игру, то его результат — проигрыш;
- Результат игрока добавляется в список результатов;
- Список сортируется по убыванию количества набранных очков;
- Определяется позиция результата игрока в списке;
- Вычисляется процент успеха.

Например, последняя игра попала в статистику второй, с 10 набранными баллами:

```
const statistics = [4, 5, 8, 10, 11];
```

В этом случае результат лучше, чем у 60% игроков, потому что игра находится на 2 месте из 5, а это лучше, чем у троих игроков из 5: `3/5 = 0.6 = 60%`.

Начните решение с описания тестовых случаев. Реализуйте работу функции и убедитесь, что все тесты проходят написанные проверки.

Дополнительно

По желанию в этом же модуле можно выполнить дополнительное задание из Технического Задания.

По такому же алгоритму опишите остальные бизнес-функции вашей игры:

- Управление жизнями игрока
- Переключение уровней
- Отсчёт времени

При написании этих функций помните о «чистоте» функции и о неизменяемости входных данных.

Обратите внимание

Все примеры реализаций, имена функций, параметров и методов даны в качестве примера и не обязательны для повторения.

4 лекция

4.1

В этом задании мы отделим данные от представления: избавимся от статического контента в шаблонах модулей и создадим для каждого из модулей подходящую структуру данных

Задача

Доработайте модули отрисовки экранов, созданные в прошлых разделах так, чтобы данные, находящиеся в наших шаблонах («рыбе») были отделены от вёрстки

1. Разделите шаблоны на отдельные логические части. Если у вас в разметке модуля есть несколько логических блоков, выделите их в отдельные шаблоны внутри модуля. Например, в модуле игры можно отдельно выделить шаблоны для блока заголовка или логотипа
2. Опишите структуры данных, которые будут хранить информацию о вашей игре. Используйте способ дедукции: декомпозируйте сущности, начиная с большей, до частных
    - Начните с описания игры. Заведите подходящую структуру данных, которая описывает игру. Выделите свойства, которые описывают текущую игру (например, тип игры, список вопросов, статистику).
    - Опишите вопрос. Заведите под него подходящую структуру данных, выделите свойства, которые описывают вопрос (текст вопроса, список ответов).
    - Опишите ответ. Заведите под него подходящую структуру данных, выделите свойства, описывающие ответ (правильный ли это ответ, содержимое ответа: ссылка на изображение или аудиофайл).
    - Продолжайте, пока не опишите в виде структур все данные, которые необходимы для описания игры

    Обратите внимание

    Структуры данных не должны повторять разметку ваших модулей. Не стоит заводить данные вида`header`, `footer` и прочее. В этом задании важно выделить логические модули, которые мы используем для описания игры

    Чтобы понять, что должно оказаться в данных, а что нет, задайте себе вопрос: «Есть ли смысл скачивать эту информацию с сервера отдельно, может ли она измениться?». Не стоит заводить в данных структуру, которая описывает размеры логотипа или статический текст, в структурах должны храниться только те данные, которыми мы будем оперировать в проекте: получать их с сервера отдельно, изменять, отправлять на сервер
3. Доработайте шаблоны так, чтобы они отображали информацию из полученных структур данных:
    1. Перепишите модули, которые экспортировали DOM-элемент, на функцию, которая принимает данные, выводит их, навешивает обработчики и возвращает созданный DOM-элемент, который впоследствии будет вставлен в страницу.

        Пример того, как может выглядеть модуль с данными:

```
export default {
  greeting: `Привет!`,
  content: {
    title: `Приготовься!`,
    text: `Тебе предстоит сразится с тремя соперниками!`
  },
  creationDate: `2016 — 2017`
};
```

Пример того, как может выглядеть модуль экрана:

```
import getHeader from './header';
import getFooter from './footer';
export default (data) => {

  const content = `<div>
    <h1>${data.content.title}</h1>
    <p>${data.content.text}</p>
  <div>`;

  const article = `
    ${getHeader(data.greeting)}
    <div class="content">
      ${content}
    </div>
    ${getFooter(data.creationDate)}`;

  return getElementFromTemplate(article);
}
```

2. Передайте нашим новым модулям соответствующие данные, необходимые для его отрисовки: модули, которые выводят вопросы, должны получить объект вопроса и показать его пользователю, модули, которые выводят статистику должны получить объект со статистикой и показать его и т.д.
3. Обобщите повторяющиеся модули так, чтобы они работали с уникальным набором данных и вёрстка в каждом модуле не дублировалась. Например, вместо трёх экранов с вопросами у вас теперь должен быть один, который показывает текущий вид вопроса в зависимости от переданных данных.

4. Реализуйте логику переключения экранов:
   - игрок теперь должен увидеть все 10 игровых экранов со статическими вопросами;
   - при выборе ответа — ответ должен сохраняться в соответствующую структуру данных;
   - так как таймера в данном задании ещё нет, то все ответы можно записывать как обычные и время для них считать среднее (время фиксировано — так что ответ оказывается не быстрым и не медленным);
   - при исчерпании лимита ошибок игра останавливается и игрок попадает на экран с результатами;
   - на экране результатов должен происходить подсчёт очков и вывод результата игры в соответствии с техническим заданием;

Для Пиксель:

Для тестирования игры вы можете использовать фиксированный набор изображений

Пример тестовых изображений:

```
module.exports = {
  paintings: [
    // People
    'https://k42.kn3.net/CF42609C8.jpg',

    // Animals
    'https://k42.kn3.net/D2F0370D6.jpg',

    // Nature
    'https://k32.kn3.net/5C7060EC5.jpg'
  ],
  photos: [
    // People
    'http://i.imgur.com/1KegWPz.jpg',

    // Animals
    'https://i.imgur.com/DiHM5Zb.jpg',

    // Nature
    'http://i.imgur.com/DKR1HtB.jpg'
  ]
};
```

Вы можете использовать собственные тестовые изображения, помимо тех, что представлены выше.

Для угадай:

Для тестирования игры вы можете использовать фиксированный набор мелодий

Пример тестовых мелодий:

```
// Music from https://www.youtube.com/audiolibrary/music?feature=blog
export default [
  {
    artist: `Kevin MacLeod`,
    name: `Long Stroll`,
    image: `https://yt3.ggpht.com/-fkDeGauT7Co/AAAAAAAAAAI/AAAAAAAAAAA/dkF5ZK
krxRo/s900-c-k-no-mo-rj-c0xffffff/photo.jpg`,
    src: `https://www.youtube.com/audiolibrary_download?vid=91624fdc22fc54ed`
,
    genre: `Jazz`
  },
  {
    artist: `Jingle Punks`,
    name: `In the Land of Rhinoplasty`,
    image: `https://i.vimeocdn.com/portrait/992615_300x300`,
    src: `https://www.youtube.com/audiolibrary_download?vid=dc3b4dc549becd6b`
,
```

```
    genre: `Rock`
  },
  {
    artist: `Audionautix`,
    name: `Travel Light`,
    image: `http://4.bp.blogspot.com/-kft9qu5ET6U/VPFUBi9W-MI/AAAAAAAACYM/UxXilXKYwOc/s1600/audionautix%2BHalf%2BSize.jpg`,
    src: `https://www.youtube.com/audiolibrary_download?vid=a127d9b7de8a17cf`,
    genre: `Country`
  },
  {
    artist: `Riot`,
    name: `    Level Plane`,
    image: `https://i.ytimg.com/vi/jzgM3m8Vp1k/maxresdefault.jpg`,
    src: `https://www.youtube.com/audiolibrary_download?vid=dfb828f40096184c`,
    genre: `R&B`
  },
  {
    artist: `Jingle Punks`,
    name: `Lucky Day`,
    image: `https://i.vimeocdn.com/portrait/992615_300x300`,
    src: `https://www.youtube.com/audiolibrary_download?vid=bcbe5be936a32fb1`,
    genre: `Pop`
  },
  {
    artist: `Quincas Moreira`,
    name: `Firefly`,
    image: `http://www.atribuna.com.br/fileadmin/_processed_/csm_Quincas-Moreira-Foto-Divulgacao_76d1a8b00e.jpg`,
    src: `https://www.youtube.com/audiolibrary_download?vid=79100e44c826e2f7`,
    genre: `Electronic`
  }
];
```

Обратите внимание

Для решения этого задания вам нужно работать с аудио-треками

Для этого удобнее всего будет завести специальный модуль, который будет отвечать за работу с треками и отображения его состояния — запущен/остановлен. Для работы с треками используйте тег HTMLAudioElement

---

4.2

Это дополнительное задание

Выполнение этого задания сделает ваш проект чуть более удобным с точки зрения UX и позволит вам набрать дополнительные очки во время защиты.

Код решения будет оцениваться по тем же критериям, что и основной код, поэтому выполнением этого задания может ухудшить оценку по другим критериям.

Для Пикселя

По ходу игры пользователи видят изображения, которые вписываются в определённые рамки. Проблема заключается в том, что не всегда пропорции изображений в данных совпадают с пропорциями блоков, в которые изображения должны быть показаны. В этом задании мы решим проблему таких изображений.

Чтобы решить проблему разных пропорций изображения и контейнера, вам предлагается написать метод, который будет отвечать за подгонку размеров изображения так, чтобы оно могло вписаться в определённые рамки.

Этот метод должен принимать на вход два объекта со свойствами `width` и `height`. Первый объект описывает размеры рамки, в которые должно быть вписано изображение, а второй объект описывает размеры изображения, которые нужно подогнать под рамку. Метод должен возвращать новый объект, который будет содержать изменённые размеры изображения.

Ниже представлен код теста, который должен проходить метод. Добавьте код теста в свой проект и сделайте так, чтобы этот тест проходил. Т.о. эта задача решается с помощью TDD:

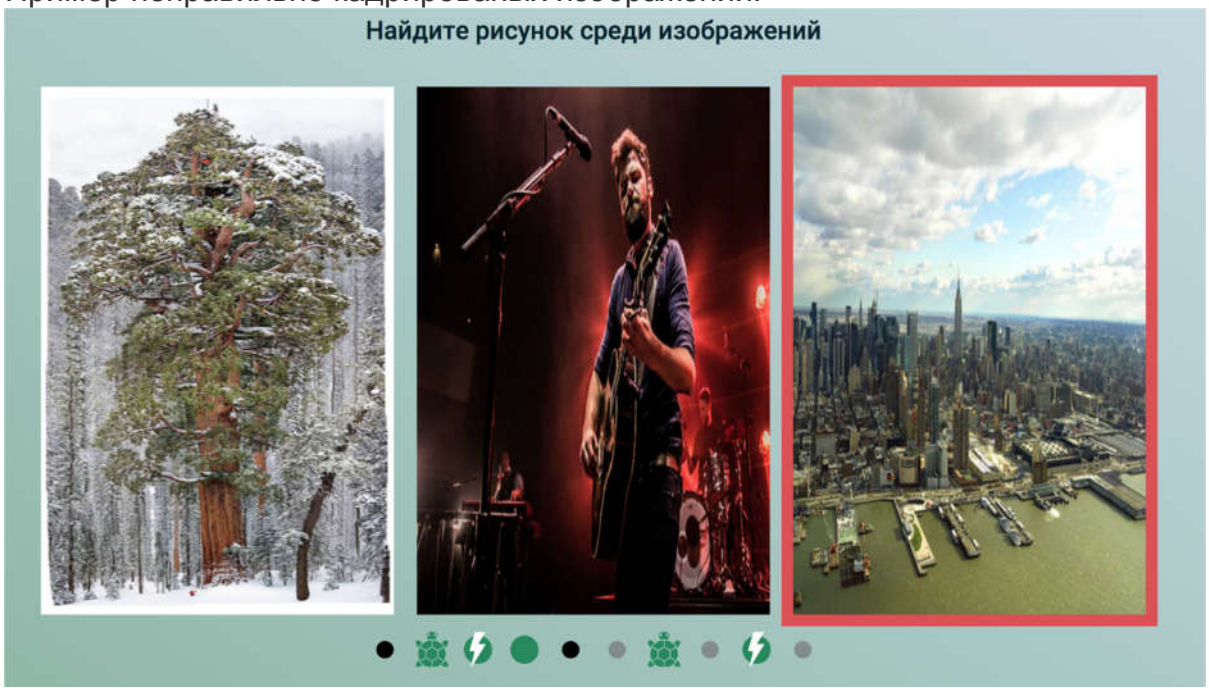
```
import assert from 'assert';

// NB! В этом тесте подразумевается, что модуль называется `resize`
// и экспортирует метод `resize`. Если вы назовёте метод иначе, учтите
// это в тексте этого теста
import {resize} from './resize';

const createTestForFrame = (frame) => {
  const assertRatio = (given, expected) => {
    const actual = resize(frame, given);
    assert.deepEqual(actual, expected);
  };

  const createTest = (expected, multiplier) => {
    const given = {
      width: Math.floor(expected.width * multiplier),
      height: Math.floor(expected.height * multiplier)
    };
    it(`shrink ${multiplier}x: ${given.width}x${given.height} => ${expected.w
idth}x${expected.height}`, () => {
      assertRatio(given, expected);
    });
  };

  const sequence = (expected) => {
    createTest(expected, 8);
    createTest(expected, 7);
    createTest(expected, 5);
    createTest(expected, 4);
    createTest(expected, 3);
    createTest(expected, 2);
    createTest(expected, 1);
  };
```

```
  describe(`Resize into frame: ${frame.width}x${frame.height}`, () => {

    describe(`when "width === height"`, () => {
      sequence({width: frame.width, height: frame.height});
    });

    describe(`when "width > height"`, () => {
      sequence({width: frame.width, height: Math.floor(frame.height / 2)});
    });

    describe(`when "width < height"`, () => {
      sequence({width: Math.floor(frame.width / 2), height: frame.height});
    });

  });
};

createTestForFrame({width: 256, height: 256});
createTestForFrame({width: 256, height: 128});

createTestForFrame({width: 468, height: 458});
createTestForFrame({width: 705, height: 455});
createTestForFrame({width: 304, height: 455});
```

Почему не подходит кадрирование

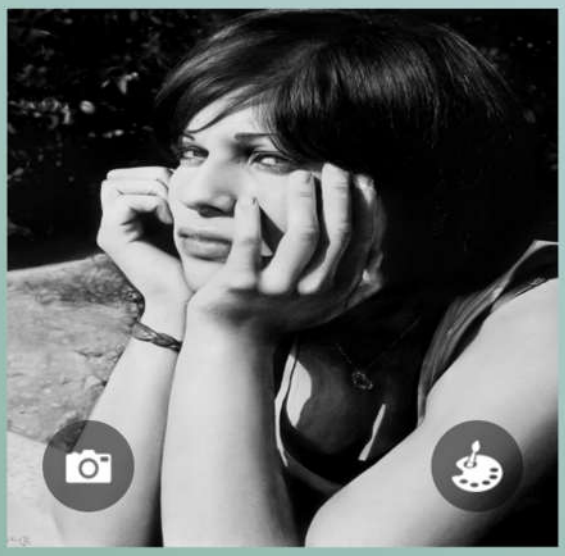
Существует альтернативный способ решить задачу с пропорциями: отрезать часть изображения так, чтобы его пропорции совпали с пропорциями блока, в который нужно вписать изображение. В нашем случае такое решение не подходит, потому что отрезанная часть изображения может быть важной для решения задачи и содержать какие-то признаки, по которым можно определить, является изображение картиной или фотографией

Пример неправильно кадрированых изображений:

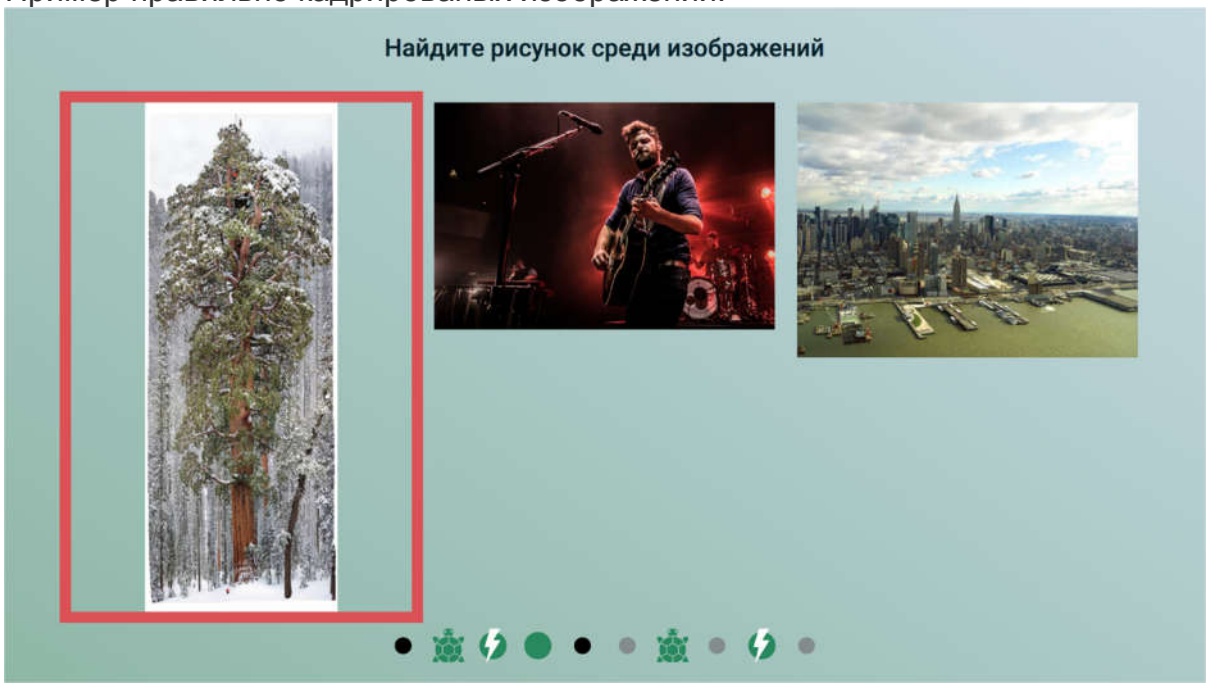


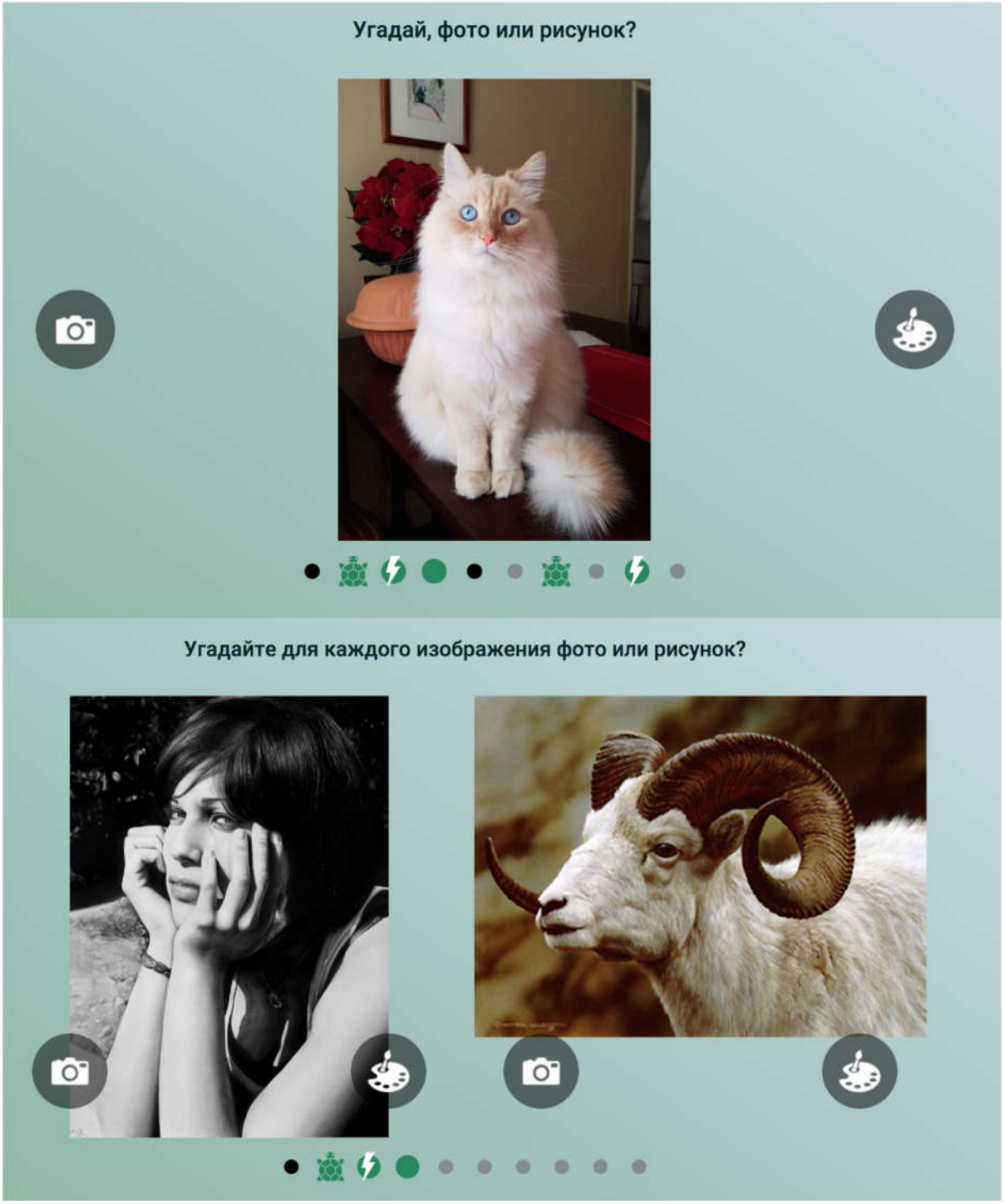
Угадай, фото или рисунок?

Угадайте для каждого изображения фото или рисунок?



Пример правильно кадрированых изображений:





Для угадай

На игровых экранах дизайнеры придумали показывать оставшееся время очень необычным способом: индикатор оставшегося времени представляет собой окружность, у которой с истечением времени постепенно уменьшается длина обводки. Это задание призвано помочь вам запрограммировать этот индикатор

В разметке вашего проекта уже есть блок для отрисовки таймера, это svg-элемент `.timer`, в котором находится элемент `.timer-line`. Остаётся только доработать этот блок так, чтобы он показывал оставшееся время

Блок `.timer-line` представляет собой окружность. Все стили, нужные для того, чтобы задать ей обводку, уже описаны, остаётся их правильно задать. Чтобы у блока появилась обводка, нужно добавить ему атрибут `stroke-dasharray`. Особенности обводки в svg таковы, что с помощью одного свойства можно описать как сплошную обводку, так и обводку штрихами. Атрибут `stroke-dasharray` задаёт длину штриха. Если он равен длине всего элемента, то обводка будет сплошной. Чтобы понять, как это работает, попробуйте задать блоку `.timer-line` разные значения атрибута `stroke-dasharray`

Задача написания индикатора сводится к тому, чтобы увеличивать со временем значение атрибута `stroke-dashoffset` на некий фиксированный шаг. Значение `stroke-dasharray`, которое должно быть равным полной длине окружности изменять не нужно. Если начать уменьшать со временем `stroke-dasharray`, браузер начнёт изменять начальную точку отрисовки штриховки, и индикатор перестанет быть понятным.

Теперь остаётся подобрать правильные значения `stroke-dasharray` и `stroke-dashoffset`. Особенность заключается в том, что блок, который нам нужно обвести, представляет собой окружность, поэтому для расчёта длины обводки нам нужно использовать формулу длины окружности.

Длина окружности равна `2 * Math.PI * radius`, где `radius` — это радиус окружности. Он нам известен, так как он задан в атрибуте `r` и равен `370`. Получается, что значение `stroke-dasharray` должно быть равным `2 * Math.PI * 370 === 2324.7785636564467`. Полученное значение можно округлить до `2325`.

Чтобы посчитать длину расстояния между штрихами, нужно полную длину окружности поделить на число шагов, за которые проходит анимация. Например, мы поставили таймер на одну минуту, при этом обновляться он будет раз в секунду. Значит, анимация будет состоять из 60 шагов. На каждом шаге размер `stroke-dashoffset` должен увеличиваться на `2325/60 = 38.75` пикселей.

Решение можно оформить по-разному. Во View известна полная длина окружности. Можно создать некий метод, который извне будет принимать на вход соотношение оставшегося времени к полному (должно получаться число от 0 до 1) и будет рассчитывать длину штриха и длину расстояния между штрихами.

Ниже приведены условные тесты условного метода `getRadius`, который принимает на вход соотношение прошедшего времени к полному и радиус окружности и возвращает объект, содержащий свойства `stroke`для длины штриха и `offset` для расстояния между штрихами.

```
import assert from 'assert';
import {getRadius} from './get-radius';

describe(`Function should correctly calculate circle length`, () => {
  describe(`Normal cases`, () => {
    it(`Should return full length and 0 in initial state`, () => {
      // 2 * 3.14 * 100 = 6.28 * 100 = 628
      assert.equal(getRadius(1, 100).stroke, 628);
      assert.equal(getRadius(1, 100).offset, 0);
    });
```

```
    it(`Should return 0 and full length in the final state`, () => {
      // 2 * 3.14 * 100 = 6.28 * 100 = 628
      assert.equal(getRadius(0, 100).stroke, 628);
      assert.equal(getRadius(0, 100).offset, 628);
    });

    it(`Offset and length should be equal on a half`, () => {
      // 2 * 3.14 * 100 / 2 = 3.14 * 100 = 314
      assert.equal(getRadius(0.5, 100).stroke, 628);
      assert.equal(getRadius(0.5, 100).offset, 314);
    });
  });
});
```

Если у вас возникли проблемы с отображением блока `.timer-line` и вы не видите круглую обводку вокруг блока уровня, проверьте стили блока, они должны содержать ссылку на svg-фильтр: "filter: url(#blur)" или "filter: url(.#blur)". Если стили есть, но блок не отображается, вы можете удалить фильтр.

5 лекция

В этом задании мы закончим описание поведения игры: свяжем структуры данных, созданные в прошлом разделе, с их отображениями *(View)* через промежуточные модули.

Перепишите код вашего проекта следующим образом:

1. Создайте базовый класс для представлений: `AbstractView`. Этот класс должен иметь следующие методы:
   - Геттер `template`, который возвращает строку, содержащую разметку. Метод должен быть абстрактным, то есть этот метод должен быть обязательно переопределён в объектах-наследниках
   - `render` — метод, который будет создавать DOM-элемент на основе шаблона, который возвращается геттером `template`
   - `bind` — метод, который будет добавлять обработчики событий. Метод по умолчанию ничего не делает. Если нужно обработать какое-то событие, то этот метод должен быть переопределён в наследнике с необходимой логикой
   - Геттер `element`, который возвращает DOM-элемент, соответствующий представлению. Метод должен создавать DOM-элемент с помощью метода `render`, добавлять ему обработчики, с помощью метода `bind` и возвращать созданный элемент. *Метод должен использовать **ленивые вычисления** — элемент должен создаваться при первом обращении к геттер с помощью метода `render`, должны добавляться обработчики (метод `bind`). При последующих обращениях должен использоваться элемент, созданный при первом вызове геттера.*
2. К текущему этапу, для каждого из игровых экранов у вас должен быть создан отдельный модуль. Например, `welcome.js` для приветственного экрана. Перепишите код каждого из модулей, описывающих экраны так, чтобы в них не осталось работы с DOM-деревом.

   В результате рефакторинга в модулях экранов должна остаться только логика, связывающая структуры данных, которые вы создали в прошлом разделе, с модулями отображений экранов (View). Модули экранов должны:

   - Создавать View, унаследованный от `AbstractView` для этого экрана
   - Добавлять в созданный View обработчики изменений
   - Вызывать при изменениях во View логику по изменению структур данных, описывающих состояние игры

   При изменении состояния игры не забудьте вести статистику: сохранять результат пользователя при каждом ответе.

Каждый из View должен быть пассивным, то есть в самом модуле не должно быть заложено никакой логики по переключению экранов, каждое отображение умеет только показать переданные данные. Все изменения передаются внешнему слушателю через обработчик событий или коллбэк, и вся работа с данными должна вестись снаружи

Пример

У вас есть модуль, который показывает некоторый экран `screen.js`. Этот модуль управляет DOM-элементом экрана: содержит кнопку и добавляет на неё обработчики, показывает диалоговое окно и реагирует на все изменения.

Вам нужно разделить логику этого модуля на две части: часть, отвечающую за отображение и работу с DOM, и часть, которая управляет переключением экранов. В итоге у вас должно получиться два модуля: `button-view.js` и `screen.js`.

- **Модуль `button-view.js`** будет экспортировать класс, который наследуется от `AbstractView`. Этот модуль будет создавать DOM-элемент кнопки и добавлять на него обработчики DOM-событий. Кроме этого, никакой логики в этом модуле заложено не будет

  Класс `ButtonView` должен содержать некий метод `onClick`, который будет переопределяться снаружи для того, чтобы мы могли описать изменения, которые должны произойти при нажатии на кнопку

- **Модуль `dialog-view.js`** будет экспортировать класс, который наследуется `AbstractView`. Этот модуль будет создавать DOM-элемент диалога и управлять необходимыми DOM-событиями. Кроме этого, никакой логики в этом модуле заложено не будет

  Класс `DialogView` должен содержать методы `onConfirm` и `onCancel`, который будет переопределяться снаружи для того, чтобы мы могли описать действия, которые нужно совершить при выборе пользователя

- **Модуль `screen.js`**, который будет экспортировать функцию. Эта функция будет создавать экземпляр класса `ButtonView` и переопределять метод `onClick` так, чтобы при каждом вызове `onClick` показывалось диалоговое окно, которое в свою очередь ждёт результата действий от пользователя

```
// button-view.js
import AbstractView from "./abstract-view";

/**
 * Модуль ButtonView описывает внешний вид и поведение
 * всех кнопок в нашем приложении.
 * Этот модуль ничего не знает про то действие, которое
 * будет выполнено при нажатии на кнопку и просто передаёт
 * её внешнему обработчику (onClick). В этом заключается
 * особенность подхода MV*: каждый модуль выполняет свою
 * роль, и модуль отображения описывает только логику отображения
 **/
export default class ButtonView extends AbstractView {
  constructor(title = `Нажми меня полностью`) {
    super();
    this.title = title;
  }

  get template() {
    return `<button>${this.title}</button>`;
  }

  /**
```

```
   * Метод bind описывает поведение кнопки при нажатии на нее.
   **/
  bind() {
    this.element.onclick = (evt) => {
      evt.preventDefault();
      this.onClick();
    };
  }

  onClick() {
  }
}

// dialog-view.js
import AbstractView from "./abstract-view";

/**
 * Модуль DialogView описывает внешний вид и поведение
 * диалоговых окон в нашем приложении
 **/
export default class DialogView extends AbstractView {
  constructor(title, content) {
    super();
    this.title = title;
    this.content = content;
  }

  /** Геттер template создаёт разметку экрана */
  get template() {
    return `<dialog>
<form class="dialog-form" autocomplete="off">

    <div class="dialog-header">
      ${this.title}
      <span class="dialog-close">×</span>
    </div>

    <div class="dialog-content">
      ${this.content}
    </div>

    <div class="dialog-footer">
      <button class="button dialog-submit">OK</button>
      <button class="button dialog-cancel">Отмена</button>
    </div>
  </form>
</dialog>`;
  }

  /**
   * Метод bind описывает поведение диалогового окна.
   * Обратите внимание, что метод не вызывает напрямую действия,
   * которые должны произойти по нажатию на кнопку, а вместо
   * этого вызывает коллбэк onConfirm или onCancel
   * в зависимости от выбора пользователя, который будет
   * переопределяться снаружи (паттерн «Слушатель»)
```

```
   **/
  bind() {
    const closeButton = this.element.querySelector(`.dialog-close`);
    const cancelButton = this.element.querySelector(`.dialog-cancel`);
    const confirmButton = this.element.querySelector(`.dialog-submit`);

    const cancelHandler = (evt) => {
      evt.stopPropagation();
      evt.preventDefault();

      this.onCancel();
    };

    cancelButton.addEventListener(`click`, cancelHandler);
    closeButton.addEventListener(`click`, cancelHandler);

    confirmButton.addEventListener(`click`, (evt) => {
      evt.stopPropagation();
      evt.preventDefault();

      this.onConfirm();
    });

  }

  onCancel() {
  }

  onConfirm() {
  }
}


// screen.js
/**
 * Файл button-view.js управляет тем, как выглядит и ведёт себя Button.
 * Файл dialog-view.js управляет тем, как выглядит и ведёт себя Dialog.
 * Для того, чтобы показать экран, мы создаём новые объекты —
 * ButtonView и DialogView и переопределяем интересующие нас коллбэки.
 * Таким образом ButtonView и DialogView описывают поведение кнопки и
 * диалога в нашем приложении и предоставляют инструменты,
 * которые помогают описать нужное нам поведение, которое задаёт этот модуль
 **/
import ButtonView from './button-view';
import DialogView from './dialog-view';
import {show} from './util';

export default () => {
  const myButton = new ButtonView(`Заманчивое предложение`);
  const myDialog = new DialogView(`Привет!`, `Придёшь к нам ещё?`);

  myDialog.onConfirm = () => console.log(`Ура, пользователь согласился!`);
  myDialog.onCancel = () => console.log(`Ой-ой, пользователь отказался =(`);

  myButton.onClick = () => show(myDialog);
```

```
  show(myButton);
};
```

Названия методов (например `onClick`, `onConfirm`, `onCancel` и прочие) условны и даны только для примера. В вашем случае методы должны называться соответственно логике вашего приложения.

6 лекция

В этом задании мы перепишем нетронутые пока что модули экранов так, чтобы они стали презентерами из паттерна MVP.

Перепишите в стиле MVP код вашего проекта:

1. Добавьте класс модели вашей игры `GameModel` — этот класс вам понадобится для хранения и управления текущим состоянием игры, а так же он зафиксирует модель (*Model*) вашего приложения и **API**, по которому работают компоненты в вашем приложении.
2. Создайте класс `GameScreen`. Этот класс будет выступать в роли контроллера (*Controller*) (или презентера (*Presenter*) и будет связывать модель вашей игры с представлением:
    - Конструктор должен принимать на вход модель игры `GameModel`
    - Конструктор должен создавать и управлять представлением игры `GameView`
    - Метод `init` или `start` должен устанавливать изначальное состояние игры и начинать игру
    - Запускать/останавливать отсчёт времени в игре и обновлять модель и представление соответствующим образом
    - Должен реагировать на действия, происходящие в представлении (выбор ответа игроком), обрабатывать его и обновлять модель и представление в соответствии с ответом
3. Создайте соответствующие контроллеры остальных экранов приложения (`IntroScreen`, `ResultsScreen`, и т.д.):
4. Создайте класс `Application`, который будет выполнять роль роутера (*Router*) в вашем приложении и управлять всеми экранами вашей игры:

```
export default class Application {

  static showWelcome() {
    const welcome = new WelcomeScreen();
    changeView(welcome.element);
    }

    static showGame(userName) {
      const model = new QuestModel(userName);
      const gameScreen = new GameScreen(model);
      changeView(gameScreen.element);
      gameScreen.startGame();
    }

    static showStats(stats) {
      const statistics = new StatsScreen(stats);
      changeView(statistics.element);
    }

}
```

24. Свяжите все ваши экраны через общий класс `Application`:
    - Модуль `main.js` должен вызывать метод `Application.showWelcome()`
    - Модуль игры при достижении конца игры должен переходить на экран статистики`Application.showStats(stats)`

    - Модуль статистки должен уметь возвращаться в начало игры `Application.showGame(userName)` или приветственный экран `Application.showWelcome()` в соответствии с техническим заданием
25. После завершения этого задания все игровые экраны должны работать в соответствии с техническим заданием: после старта игры должно отсчитываться время, сохраняться неправильные ответы и выводиться статистика с результатами игры.

Обратите внимание

Что имена классов и методов условны, т.е. в зависимости от принятого вами стиля именования презентеры могут называться как `GamePresenter`, `GameScreen` или `GamePage`. Основное правило, которого стоит придерживаться, — соблюдать стиль именования во всём проекте, т.е. похожие вещи должны называться похоже и не должны вводить в заблуждение

7 лекция

7,1

В этом задании мы загрузим данные для наших игр с сервера.

Воспользуйтесь объектом fetch и загрузите данные с сервера:

1. Загрузите данные при помощи `window.fetch` с сервера:

Для Пикселя

Данные лежат по адресу: `https://es.dump.academy/pixel-hunter/questions`

Для Угадай

Данные лежат по адресу: `https://es.dump.academy/guess-melody/questions`

2. Перепишите ваши модели так, чтобы они получали данные, загруженные с сервера, вместо фиктивных (**mocked**), которые использовались ранее

Пример формата данных:

Для Пикселя

```
[
  {
    "type": "two-of-two",
    "question": "Угадайте для каждого изображения фото или рисунок?",
    "answers": [
      {
        "image": {
          "url": "http://placehold.it/468x458",
          "width": 468,
          "height": 458
        },
        "type": "photo"
      },
      {
        "image": {
          "url": "http://placehold.it/468x458",
          "width": 468,
          "height": 458
        },
        "type": "painting"
      }
    ]
  },
  {
    "type": "tinder-like",
    "question": "Угадай, фото или рисунок?",
    "answers": [
      {
```

```
      "image": {
        "url": "http://placehold.it/705x455",
        "width": 705,
        "height": 455
      },
      "type": "photo"
    }
  ]
},
{
  "type": "one-of-three",
  "question": "Найдите рисунок среди изображений",
  "answers": [
    {
      "image": {
        "url": "http://placehold.it/304x455",
        "width": 304,
        "height": 455
      },
      "type": "photo"
    },
    {
      "image": {
        "url": "http://placehold.it/304x455",
        "width": 304,
        "height": 455
      },
      "type": "painting"
    },
    {
      "image": {
        "url": "http://placehold.it/304x455",
        "width": 304,
        "height": 455
      },
      "type": "photo"
    }
  ]
},
{
  "type": "one-of-three",
  "question": "Найдите фото среди изображений",
  "answers": [
    {
      "image": {
        "url": "http://placehold.it/304x455",
        "width": 304,
        "height": 455
      },
      "type": "painting"
    },
    {
      "image": {
        "url": "http://placehold.it/304x455",
        "width": 304,
        "height": 455
```

```
        },
        "type": "painting"
      },
      {
        "image": {
          "url": "http://placehold.it/304x455",
          "width": 304,
          "height": 455
        },
        "type": "photo"
      }
    ]
  }
]
```

Возможные типы вопросов:

```
const QuestionType = {
  TWO_OF_TWO: 'two-of-two',
  TINDER_LIKE: 'tinder-like',
  ONE_OF_THREE: 'one-of-three'
};
```

Возможные типы ответов:

```
const AnswerType = {
  PAINTING: 'painting',
  PHOTO: 'photo'
};
```

Для Угадай

```
[
  {
    "type": "genre",
    "question": "Выберите все песни в жанре R'n'B",
    "genre": "rnb",
    "answers": [
      {
        "src": "/path/to/file.mp3",
        "genre": "rnb"
      },
      {
        "src": "/path/to/file.mp3",
        "genre": "blues"
      },
      {
        "src": "/path/to/file.mp3",
        "genre": "rock"
      },
      {
        "src": "/path/to/file.mp3",
        "genre": "rnb"
```

```
      }
    ]
  },
  {
    "type": "genre",
    "question": "Выберите все блюзовые песни",
    "genre": "blues",
    "answers": [
      {
        "src": "/path/to/file.mp3",
        "genre": "blues"
      },
      {
        "src": "/path/to/file.mp3",
        "genre": "pop"
      },
      {
        "src": "/path/to/file.mp3",
        "genre": "rock"
      },
      {
        "src": "/path/to/file.mp3",
        "genre": "rnb"
      }
    ]
  },
  {
    "type": "artist",
    "question": "Кто исполняет эту песню?",
    "src": "path/to/file.mp3",
    "answers": [
      {
        "image": {
          "url": "http://placehold.it/705x455",
          "width": 300,
          "height": 300
        },
        "title": "Пелагея",
        "isCorrect": false
      },
      {
        "image": {
          "url": "http://placehold.it/705x455",
          "width": 300,
          "height": 300
        },
        "title": "Краснознамённая дивизия имени моей Бабушки",
        "isCorrect": false
      },
      {
        "image": {
          "url": "http://placehold.it/705x455",
          "width": 300,
          "height": 300
        },
        "title": "Кровосток",
```

```
        "isCorrect": true
      }
    ]
  }
]
```

Возможные типы вопросов:

```
const QuestionType = {
 GENRE: 'genre',
 ARTIST: 'artist'
};
```

Возможные музыкальные жанры:

```
const Genre = {
 COUNTRY: 'country',
 BLUES: 'blues',
 FOLK: 'folk',
 CLASSICAL: 'classical',
 ELECTRONIC: 'electronic',
 HIP_HOP: 'hip-hop',
 JAZZ: 'jazz',
 POP: 'pop',
 ROCK: 'rock'
};
```

7,2

В этом задании мы запишем данные от наших игр на сервер

Задание

По завершении игры нужно отправить результат пользователя на сервер:

Для пикселя

Сервер принимает статистику по адресу: `https://es.dump.academy/pixel-hunter/stats/:appId-:username`

Метод создания новых данных `POST`, где:

- `:appId` — это уникальный статический **id** вашего приложения (например, можете составить его из случайных цифр — **22101985**),
- `:username` — это имя игрока, которому принадлежит статистика

Формат данных может быть произвольный, например:

```
{
  stats: ['correct', 'wrong', 'fast', 'slow', 'correct', 'wrong', 'fast', 'slow', 'correct', 'wrong'], // Статистика ответа пользователя
  lives: 0 // Кол-во оставшихся жизней
```

```
}
```

Обратите внимание

Сохранять результат пользователя нужно независимо от успеха

Сервер получает статистику по адресу: `https://es.dump.academy/pixel-hunter/stats/:appId-:username`

Метод получения статистики `GET` для пользователя `:username` в приложении с кодом `:appId`. Формат данных:

```
[
  {
    date: 1234567567898, // Дата создания статистики в ms
    stats: ['correct', 'wrong', 'fast', 'wrong', 'correct', 'wrong', 'wrong']
, // Статистика ответа пользователя
    lives: 0 // Кол-во оставшихся жизней
  },
  {
    date: 1234567567898, // Дата создания статистики в ms
    stats: ['correct', 'wrong', 'fast', 'slow', 'correct', 'wrong', 'fast', '
slow', 'correct', 'wrong'], // Статистика ответа пользователя
    lives: 0 // Кол-во оставшихся жизней
  }
]
```

Пример типов ответов:

```
const Result = {
  CORRECT: 'correct',
  WRONG: 'wrong',
  FAST: 'fast',
  SLOW: 'slow'
};
```

Экран статистики должен выводить всю историю прохождения игры в соответствии с техническим заданием.

Обратите внимание

Формат данных, который принимает сервер, — **application/json**

Для Угадай

Сервер принимает статистику по адресу: `https://es.dump.academy/guess-melody/stats/:appId`

Метод создания новых данных `POST`, где `:appId` — это уникальный статический **id** вашего приложения (например, можете составить его из случайных цифр — **22101985**).

Формат данных может быть произвольный, например:

```
{
  time: 230, // Общее время потраченное на игру
```

```
  answers: [15, 45, 10, 6, -1, 20, 31, -1, -1, 15]  // Статистика ответов пол
ьзователя
}
```

Обратите внимание

Сохранять результат нужно только в случае успешного прохождения игры

Сервер отдаёт статистику по адресу: `https://es.dump.academy/guess-melody/stats/:appId`

Метод получения статистики `GET`, где `:appId` — это уникальный статический **id** вашего приложения:

```
[
  {
    date: 1234567567898, // Дата создания статистики в ms
    time: 230, // Время за сколько ответил пользователь
    answers: [15, 45, 10, 6, -1, 20, 31, -1, -1, 15]  // Статистика ответов п
ользователя
  },
  {
    date: 1234567567898, // Дата создания статистики в ms
    time: 228, // Время за сколько ответил пользователь
    answers: [-1, 45, 10, 6, -1, -1, 31, 40, 15, 15]  // Статистика ответов п
ользователя
  }
]
```

Экран статистики должен выводить результат игрока в соответствии с техническим заданием.

Обратите внимание

Формат данных, который принимает сервер, — **application/json**

8 лекция

8,1

Это необязательное задание, и его выполнение никак не влияет на итоговую оценку по результатам интенсива, поэтому советуем приступить к нему только после выполнения всех остальных заданий.

В этом задании мы подключим транспайлер (transpiler) Babel в сборщик зависимостей Rollup.js и несколько расширений для того, чтобы добавить поддержку новых возможностей, доступных в ECMAScript версии 2016 и выше, таких как `async/await`.

1. Установите с помощью `npm` плагин Babel для Rollup.js `rollup-plugin-babel@latest`
2. Установите с помощью `npm` дополнительные пакеты Babel `@babel/core` и `@babel/preset-env`
3. Установите необходимые в коде дополнения ([полифил](https://babeljs.io/docs/en/babel-polyfill)): `@babel/polyfill`
4. В `gulpfile.js` допишите задачу (task) `scripts` так, чтобы Rollup.js при сборке использовал Babel. Для этого в вызов метода `rollup` передайте объект с настройками, где в массив `plugins` будет записан вызов `babel` с нужными параметрами. В итоге вызов будет выглядеть примерно так:

```
rollup({
  plugins: [
    babel({
      babelrc: false,
      exclude: `node_modules/**`,
       presets: [`@babel/env`]
    })
  ]
}, `iife`)
```

13. Установите через npm плагины для загрузки CommonJS-модулей в Rollup `rollup-plugin-node-resolve` и`rollup-plugin-commonjs`
14. В `gulpfile.js` в вызове `rollup` передайте вызов установленных модулей как плагинов в массиве `plugins` перед вызовом `babel`
15. Подключите во входной точке вашего приложения (например `main.js`) `@babel/polyfill` как зависимость
16. По желанию установите из `npm` минификатор JS-кода **UglifyJS**`gulp-uglify` и подключите его в`gulpfile.js` после обработки JS-файлов `rollup`
17. Измените конфигурацию `.eslintrc` так, чтобы ESLint допускал использование в коде `async/await` и других конструкций из поздних версий ES. Для этого в `parserOptions` добавьте запись `"ecmaVersion": 2018`

Обратите внимание

Если при запуске приложения у вас возникают проблемы с горячим обновлением (**hot reload**) приложения — не обновляются скрипты, зависает сборка и т.д. В этом случае отключите кэширование у `rollup`

```
rollup({
  cache: false,
  plugins: [
    ...
  ]
```

```
}, `iife`)
```

8,2

Это необязательное задание, и его выполнение никак не влияет на итоговую оценку по результатам интенсива, поэтому советуем приступить к нему только после выполнения всех остальных заданий.

Перепишите все случаи использования `Promise` в вашем коде на `async/await`.

Вот как может выглядеть работа с загрузкой данных по сети до и после рефакторинга:

```
// до рефакторинга
const load = () => {
  fetch(`http://localhost:8080/api/request`)
    .then((resp) => resp.json())
    .then((data) => console.log(data))
};

// после рефакторинга
async function load() {
  const response = await fetch(`http://localhost:8080/api/request`);
  const responseData = await response.json();
  console.log(responseData);
};
```

Задание можно отправить на проверк

9 лекция

Это необязательное задание и его выполнение никак не влияет на итоговую оценку по результатам интенсива, поэтому советуем приступить к нему только после выполнения всех остальных заданий.

В этом задании мы перепишем наш проект на язык TypeScript.

1. Установите с помощью `npm` язык и инструменты языка **TypeScript**`typescript`
2. Создайте в корне проекта конфигурационный файл компилятора **TypeScript**`tsconfig.json`:

```
{
  "files": [
    "./js/main.ts"
  ],
  "compilerOptions": {
    "outDir": "./build",
    "allowJs": false,
     "target": "es2015",
     "removeComments": true,
     "preserveConstEnums": true,
     "sourceMap": true,
     "noImplicitAny": false,
     "strictNullChecks": true
   }
}
```

17. Установите с помощью `npm` пакеты необходимые для сборки **TypeScript** проекта через **Gulp.JS**`tsify`, `vinyl-source-stream` и `browserify`
18. В `gulpfile.js` перепишите задачу (task) `scripts` так, чтобы Rollup.js при сборке использовал **TypeScript**:

```
const browserify = require(`browserify`);
const source = require(`vinyl-source-stream`);
const tsify = require(`tsify`);
gulp.task(`scripts`, function () {
  return browserify({
    basedir: `.`,
    debug: true,
    entries: [`js/main.ts`],
    cache: {},
    packageCache: {}
  })
  .plugin(tsify)
  .bundle()
  .pipe(source(`main.js`))
  .pipe(gulp.dest(`build/js`));
});
```

34. Переименуйте все `js`-файлы в `ts`-файлы
35. Попробуйте собрать проект при помощи команды `npm run build`, если проект не собирается — исправьте ошибки компиляции и попробуйте собрать проект снова до тех пор пока он не соберётся