#3 Начинаем программировать

В этом задании разомнемся в использовании **ES2015** и вспомним как программировать: мы напишем код, который найдет на странице все шаблоны, которые описывают экраны приложения и сделаем так, чтобы при использовании клавиатурной комбинации `Alt + стрелка` 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 и npm
- Установлен плагин editorconfig
- Установлен 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. Создайте файл `js/main.js` и подключите его с помощью тега `<script/>` в конце файла `index.html`
2. Соберите ссылки на все возможные DOM-элементы экранов приложения в массив

   Все экраны записаны в шаблон `<template />` в `index.html`. Обратите внимание на то, что некоторые блоки, находящиеся в шаблоне, экранами не являются.

   Обратите внимание

   Экраны в шаблонах 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

   Содержимое шаблона, описывающего экран приложения, должно отрисовываться в блок `section.main`. В будущем, при переключении игровых экранов вы будете повторно использовать блок с таймером, но пока это не нужно и достаточно просто переключать содержимое всего блока

   ---

4. Покажите первый экран приложения. Какой из экранов приложения является приветственным, написано в техническом задании в файле `spec.md`
5. Добавьте обработчик клавиатурных событий на `document`, который будет по нажатию на комбинации `Alt + ←` и `Alt + →` переключать экраны на следующий и предыдущий 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

#4 Тесные связи

В этом разделе мы перенесем 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: `npm i gulp-sourcemaps -DE`
    3. Подключите **Rollup** как зависимость в своем `gulpfile.js`: `const rollup = require('gulp-better-rollup');`
    4. Подключите **sourcemaps** как зависимость в своем `gulpfile.js`: `const sourcemaps = require('gulp-sourcemaps');`
    5. В `gulpfile.js` доработайте задачу **scripts** следующим образом:

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

2. 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>`);
```

Модуль может называться произвольно, использовать название из примера необязательно.

3.Разбейте на модули шаблоны из `index.html` используя строки-шаблоны (Template strings). Перенесите код блоков, описанных в `template` в отдельные модули и экспортируйте из этих модулей DOM-элементы, созданные с помощью функции из второго 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` создает DOM-элемент на основе переданной ей разметки. В итоге у вас должны получиться следующие модули:

1. Главный экран, на основе блока `.main--welcome`.
2. Экран первой игры, блок `.main--level-artist`.
3. Экран второй игры, блок `.main--level-genre`.
4. Экран результатов, блок `.main--result`.
5. Экран поражения, еще один блок `.main--result`.

---

4. Доработайте модули так, чтобы переключение между разделами происходило не с помощью нажатия на клавиатурные стрелки в Угадаймелодии или с помощью блока со стрелками в Пиксельхантере, а из самих модулей.

1. Вынесите из `main.js` в отдельный модуль функцию, отрисовывающую переданный блок на страницу.
2. Уберите из модуля `main.js` всю логику переключения между экранами.
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:

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

Экран первой игры `.main--level-artist` должен показываться по нажатию на кнопку Play — блок `.main-play` на главном экране.

Экран второй игры `.main--level-genre` должен показываться по нажатию на любой из вариантов ответа — блок `.main-answer` на первом игровом экране.

Экран результатов `.main--result` должен показываться по нажатию на кнопку «Ответить», блок `.genre-answer-send`. Кнопка «Ответить» должна быть отключена, `disabled`, пока не выбран ни один из возможных вариантов ответа.

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

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

#5 Структурируй и шаблонизируй!

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, созданные на прошлом разделе так, чтобы данные находящиеся в наших шаблонах («рыбе») были отделены от верстки

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 статический контент, на функцию, которая принимает данные, отрисовывает их, навешивает обработчики и возвращает отрисованый DOM

Пример того как может выглядеть модуль с данными:

```
export default {
  greeting: 'Привет!',
  content: {
    title: 'Приготовься!',
    text: 'Тебе предстоит сразиться с тремя соперниками !'
  },
  footer: 'Сделано в <a>HTML Academy</a>'
```

```
};
```

Пример того как может выглядеть модуль экрана:

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

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

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

  return getElementFromTemplate(article);
}
```

2. Передайте нашим новым модулям соответствующие данные. Модули, которые отрисовывают вопросы, должны получить объект вопроса и отрисовать его.
3. Обобщите повторяющиеся модули так, чтобы они работали с уникальным набором данных и верстка в каждом модуле не дублировалась.

Для тестирования игры вы можете скачать несколько бесплатных мелодий с сервиса YouTube

Используйте треки помеченные как **Attribution not required**, тогда вам не придется добавлять специальную ссылку на автора трека в ваши шаблоны

Скачаные треки вы можете добавлять или не добавлять в репозиторий на ваше усмотрение

Для упрощения работы с треками у вас в репозитории лежит файл `player.js`, который предоставляет **API** для работы с аудиоконтентом на вашей странице
