Техническое задание

1. Описание функционала

1.1. Описание элементов приложения

Приложение состоит из двух основных экранов: «Треки» и «Плейлисты». Меню с левой стороны, шапка, форма поиска, профиль пользователя, а также плеер музыки общие. Переключение экранов происходит в общем контейнере посередине экрана.

2. Общий контейнер

2.1. Шапка

Шапка располагается в верхней части страницы.

В правом верхнем углу шапки отображается имя и аватар пользователя. Данный элемент не активен и не реагирует на взаимодействие со стороны пользователя. Информация о профиле загружается в момент старта приложения и не может меняться во время его использования.

В центральной части шапки отображается форма поиска. Данный элемент активен. При вводе пользователем информации в поле поиска текущие плейлисты (см. экран «Плейлисты») или треки (см. экран «Треки») фильтруются по значению, введённому в поле. Поля, по которым осуществляется фильтрация, указаны в соответствующих пунктах ТЗ.

В левой части шапки отображается логотип приложения. Данный элемент активен. При клике на логотип пользователя переносит на начальную страницу приложения.

2.2. Меню

Меню располагается в левой части страницы.

Меню представляет собой список пунктов. Первые три пункта меню всегда статичны и не меняются. Это пункты «Плейлисты», «Треки» и «Избранное». По клику на эти пункты мы переходим на соответствующий экран. В случае пункта «Плейлисты» открывается список плейлистов. В случае пункта «Треки» открывается общая подборка треков на основании предпочтений пользователя. А в случае пункта «Избранное» открывается список треков, которые имеют соответствующий флаг.

Заметьте, что в случае клика по пункту «Треки» отображаемые треки не привязаны к одному общему плейлисту.

Остальные пункты меню динамические и загружаются единожды при старте приложения. Каждый из этих пунктов носит название одного из плейлистов и при клике переносит пользователя на экран «Треки», где находятся все треки, которые относятся к данному плейлисту.

2.3. Блок управления музыкой (плеер)

Плеер располагается в нижней части страницы.

В плеере отображается информация о текущем воспроизводимом треке. Если такого трека нет, то отображается информация о случайном треке.

Информация о треке, которая отображается в плеере:

- обложка альбома;
- наименование трека;
- исполнитель трека;
- значок «Избранное».

Помимо этого, плеер предоставляет интерфейс для управления воспроизводимым треком. Для этого в плеере есть кнопки управления и полоса воспроизведения.

В плеере присутствуют следующие кнопки управления:

- **Перемешать**

При клике на данную кнопку все треки, следующие за текущим, перемешиваются в случайном порядке.

- **Назад**

При клике на данную кнопку текущий воспроизводимый трек меняется на предшествующий в плейлисте. Если текущий трек является первым в плейлисте — кнопка дизейблится.

- **Воспроизвести/Пауза**

Кнопка имеет два состояния: «Воспроизведение» и «Пауза». В первом состоянии иконка кнопки — треугольник со скошенными углами, во втором состоянии — иконка паузы (две вертикальные толстые черты). Кнопка ведёт себя как тоггл — при клике переходит с текущего состояния на другое. При этом, соответственно, в состоянии «Воспроизведение» трек должен воспроизводиться, а в состоянии «Пауза» — нет.

- **Вперёд**

При клике на данную кнопку текущий воспроизводимый трек меняется на следующий в плейлисте. Если текущий трек является последним в плейлисте — кнопка дизейблится.

- **На повтор**

При клике на данную кнопку после окончания воспроизведения текущего трека снова воспроизводится текущий. Кнопка ведёт себя как тоггл и имеет два состояния: «Включено» и «Выключено». Состояние определяет цвет кнопки. В первом случае — оранжевый, во втором — серый.

Под кнопками управления присутствует полоса проигрывания трека. Длина полосы не меняется в зависимости от длительности текущего трека. Полоса отображает текущее место воспроизведения трека, причём то, что уже воспроизведено, отображается оранжевым цветом, а то, что ещё не воспроизведено, — серым. Слева от полосы находится текущий таймкод воспроизведения в формате MM:CC, а справа — длительность трека в аналогичном формате.

В правой части плеера находится ползунок управления громкостью и иконка динамика, которые управляют текущей громкостью воспроизведения по шкале от 0 до 100. Клик по динамику в случае, если текущая громкость ненулевая, выставляет значение 0. Если текущая громкость нулевая — выставляет значение, которое предшествовало нулю.

3. Экраны

3.1. Экран «Треки»

Экран открывается в двух случаях. В случае выбора пункта «Треки» в меню и в случае выбора плейлиста в левом меню или на экране «Плейлисты». Экран представляет собой список треков. Для каждого трека выводится такая информация, как:

- номер в плейлисте;
- обложка альбома;
- наименование;
- исполнитель;
- название альбома;
- дата добавления;
- значок «Избранное»;
- длительность.

В правой части отображения каждого трека находится иконка многоточия. При клике по ней отображается модальное окно с двумя возможными контекстными действиями: «Добавить в плейлист» и «Удалить из плейлиста». Единовременно отображается лишь одно действие, в зависимости от того, находится ли текущий трек в плейлисте.

В качестве основного задания необходимо реализовать только удаление из текущего плейлиста. Добавление трека в плейлист является дополнительным заданием, и для этого необходимо реализовать модальное окно с выбором плейлиста, в который необходимо добавить трек.

Если в списке больше треков, чем умещается на экране, то в списке появляется вертикальный скролл, при этом шапка списка остаётся зафиксированной в верхней части экрана.

В случае клика по треку из списка данный трек немедленно заменяет текущий воспроизводящийся и начинает воспроизводиться. Соответственно, информация в плеере снизу так же заменяется на выбранный трек. Плейлист, который воспроизводится, — общая подборка. Если в плеере активен пункт «Случайно», то выбранный трек так же начинает воспроизводиться, а последующие идут в случайном порядке.

В случае, если в выбранном плейлисте нет ни одного трека, вместо списка отображается надпись «В текущем плейлисте нет ни одного трека».

3.2. Экран «Плейлисты»

Экран открывается в случае выбора пункта «Плейлисты» в меню. Экран представляет собой мозаику с карточками плейлистов. Список плейлистов загружается единожды в момент старта приложения.

Для каждого плейлиста отображается следующая информация:

- название плейлиста;
- количество треков;
- обложка плейлиста.

При клике на один из плейлистов он начинает воспроизводиться с первого трека. Если в плеере выбран пункт «Случайно», то плейлист начинает воспроизводиться в случайном порядке.

В случае, когда в списке больше плейлистов, чем умещается на экране, появляется вертикальный скролл.

4. Разное

В зависимости от состояния страницы некоторые элементы могут принимать активные состояния (например, кнопки плеера, кнопка «Избранное»). Примеры вы найдёте в директории с вёрсткой.

Список шагов

Мы подготовили для вас поэтапный алгоритм выполнения проекта. Это поможет вам структурировать свою работу и постепенно переходить от одного шага к другому. Рекомендуем вам сдавать проект куратору так же по этапам: после выполнения одного или нескольких шагов сдать на проверку, получить обратную связь и переходить к следующему шагу.

Этот алгоритм предложен авторами курса как оптимальный вариант работы над проектом. Если вам кажется, что задачу лучше решить другим способом, вы можете это сделать в своём проекте.

1. Настройка окружения

Настройка сборки

1. Настройте сборку проекта так, чтобы зависимости в TS собирались при помощи сборщика модулей Webpack. Для этого выполните следующие шаги:
    1. Установите из npm пакеты webpack и webpack-cli.
    2. Выполните в терминале команду npx webpack-cli init, следуйте инструкции и опишите конфигурацию сборки:
        - задайте src/index.ts точкой входа;
        - в качестве директории для сборки укажите папку public. Помните, что путь должен быть абсолютным. Используйте path.resolve;
        - файл сборки (бандл) назовите bundle.js. Для этого заведите секцию output в webpack.config.js;
        - активируйте генерацию source-maps.
    3. Добавьте в package.json скрипт с именем build:dev со значением webpack --mode development для сборки кода с помощью **Webpack** в режиме сборки для разработки. Когда наш проект будет готов, мы перенастроим этот скрипт на сборку для публикации.
2. Собираемые с помощью **Webpack** файлы (bundle.js, а не вся директория public) не должны попасть в репозиторий. Поэтому добавьте их в .gitignore.
3. Установите из npm пакеты ts-loader и typescript.
4. Добавьте в файл webpack.config.js правило для обработки файлов с расширением, которое подходит под регулярное выражение /\\.tsx?$/. Правило должно обрабатывать все файлы, кроме директории node_modules, и использовать ts-loader в качестве загрузчика.

Настройка сервера для разработки

Чтобы на каждое изменение кодовой базы не собирать бандл и не открывать public/index.html в браузере руками, настроим сервер для разработки.

1. Установите из npm пакет webpack-dev-server.
2. Опишите настройки сервера в webpack.config.js:
    - укажите абсолютный путь до директории со сборкой;
    - добавьте флаг слежки за изменением файлов, чтобы не обновлять страницу руками.
3. Осталось добавить npm-скрипт, и можно пользоваться. Назовём его serve со значением webpackserve.

Теперь достаточно выполнить команду npm serve. Браузер с нужным адресом откроется автоматически. Если этого не произойдёт, откройте адрес вручную. Адрес будет указан в терминале после запуска команды.

После этого займёмся дополнительной настройкой нашего репозитория.

- Установить пакет eslint. Инициализировать конфиг eslint — npm init @eslint/config.
- При инициализации следовать инструкции в терминале и выбрать поддержку typescript.

Добавить настройку строгой типизации согласно модулю 2, тема 2.

Промежуточный контроль

На данном этапе готов лишь репозиторий. Важно проверить, что при выполнении команды npm serve проект собирается, а также становится доступным локально. Также при любом изменении файлов проект должен пересобираться.

2. Нарезка вёрстки

Первая ваша задача будет заключаться в «нарезке» вёрстки на отдельные компоненты (элементы интерфейса). Компоненты мы будем отрисовывать динамически — с помощью TypeScript. Для этого мы подготовим функции, возвращающие вёрстку отдельных элементов интерфейса.

1. Ознакомьтесь с разметкой в директории markup.
2. Откройте файл public/index.html. В этом файле приведена общая разметка страницы (layout) и присутствуют комментарии с описанием элементов, которые должны быть добавлены в данную область. Перед закрывающим тегом body с помощью тега script подключите bundle.js.
3. В файле src/index.ts опишите функцию для отрисовки (вставки в DOM) компонентов. Спецификация функции проста: она принимает контейнер, вёрстку и конкретное место в контейнере, куда требуется отрисовать эту вёрстку. Для реализации пригодится такая функция, как insertAdjacentHTML.
4. Создайте модули (отдельные файлы) и опишите в них функции для генерации DOM-элементов. Эти модули — наши будущие компоненты, view в рамках MVP. Мы будем объединять файлы по папкам по их принадлежности к функционалу.

Например, всё, что относится к плееру внизу страницы, может находиться в папке src/footer-player. Поэтому:
    - для очередного функционала нужно завести отдельную директорию в папке src/;
    - именуйте файлы как существительные;
    - для экспорта функций используйте именованный экспорт.
5. На сколько компонентов и на какие именно компоненты разбить страницу, решите самостоятельно или воспользуйтесь предложенным списком:
    - сайдбар с плейлистами;
    - список треков;
    - страница списка плейлистов;
    - компонент плеера (находится внизу страницы);
    - компонент профиля пользователя.
6. Импортируйте эти модули в index.ts. Отрисуйте компоненты на страницу:
    - все компоненты по одному;
    - компонент «Трека» отрисовывается в списке 3 раза.
7. Откройте в браузере файл public/index.html. Все компоненты должны быть корректно отрисованы. Для проверки сравните полученный результат с разметкой в директории markup.

Промежуточный контроль

На данном этапе должна присутствовать визуальная часть приложения, но все элементы должны отрисовываться динамически в момент старта приложения. Интерактивность на данном этапе отсутствует.

3. Моканье данных

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

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, изменять, отправлять на сервер обратно.

1. Для начала нам потребуется структура данных, которая опишет трек. Это будет объект с полями:

    - наименование трека;
    - исполнитель трека;
    - альбом трека;

    - длительность трека (согласовать формат);
    - остальные данные ограничьте самостоятельно. Что ещё должно быть в структуре, можно узнать из технического задания (которого ещё нет);
    - (дополнить, согласовав с бэком).
2. Обратите внимание, это именно описание структуры данных, а не готовый к использованию объект. Ваша задача — написать функцию, которая как раз будет по этой структуре данных создавать и возвращать готовые объекты.

3. Напишите функцию, которая будет возвращать готовые объекты по структуре из предыдущего пункта. С помощью этой функции в index.ts сгенерируйте временные данные (моки). Побольше… 5–10 объектов. Для удобства их лучше сложить в массив. (дописать о том, что это треки)

4. Теперь давайте перепишем функцию по созданию шаблона трека, чтобы она на вход принимала данные — объект определённой структуры.

5. Разработайте структуры для остальных компонентов и вычислите для них данные на основе моков, полученных в пункте 2.

6. Для оставшихся компонентов так же перепишите функцию по созданию шаблона, чтобы она на вход принимала данные.

7. Далее перепишем код в index.ts для работы с моковыми данными. Вместо всех компонентов, которые вы создали во втором задании, теперь необходимо отрендерить их динамически созданные аналоги. (не нравится формулировка)

Промежуточный контроль

На данном этапе приложение должно выглядеть и функционировать, как в предыдущем этапе. С единственной оговоркой — компоненты трека теперь должны отрисовываться по случайно сгенерированным данным.

4. Начинаем юзать объекты

В этом задании мы начнём использовать объекты для описания компонентов приложения. А объекты будем строить с помощью классов.

Представим все наши компоненты в виде классов:

1. Функции, которые мы использовали для получения шаблона разметки, превратим в метод getTemplate класса. Этот метод по-прежнему должен возвращать разметку.
2. Перепишем передачу данных в разметку. Ранее мы брали данные из аргументов функции, теперь же мы будем брать данные из свойств класса,

обращаясь через this. Но прежде их нужно туда записать. Передавать данные мы будем через параметры конструктора (при вызове с new), поэтому в описании метода constructor возьмём данные из аргументов и запишем их как **приватные** свойства класса. Используйте возможность TS'а автоматически создавать свойства из переданных аргументов.

3. Добавим метод getElement, который будет создавать DOM-элемент на основе шаблона, записывать его в приватное свойство класса element и возвращать созданный DOM-элемент. Можно, например, воспользоваться современным оператором ??=. Для этого потребуется описать вспомогательную функцию, например, createElement. Лучше завести под такие функции отдельный файл, например, /src/utils.ts. А также нужно будет позаботиться о том, чтобы DOM-элемент создавался только в случае, когда он ещё не был создан.
4. И сразу же создадим метод removeElement. Он нам понадобится для очищения ресурсов. В нём мы должны удалить ссылку на созданный DOM-элемент. Для этого достаточно записать null в свойство класса element.
5. Прежде чем править наш код в index.ts, нужно изменить функцию для отрисовки (вставки в DOM), которую мы написали в самом начале. Раньше это была функция-обёртка над insertAdjacentHTML, которая принимала контейнер, шаблон и позицию отрисовки. Теперь вместо шаблона мы будем передавать DOM-элемент, поэтому insertAdjacentHTML нужно заменить на другую стандартную функцию, которая умеет вставлять DOM-элементы.
6. Теперь, когда подготовительные работы закончены, используйте в index.ts для создания компонентов не функции, а классы.

Промежуточный контроль

Аналогично прошлому шагу, но теперь каждый компонент должен быть обёрнут в свой класс.

5. Используем абстракции

В этом задании мы попрактикуемся в наследовании. Нам нужно выделить общие части компонентов в абстрактный класс.

1. Изучите структуру существующих компонентов, чтобы понять, какая логика и какие данные повторяются. Не всегда ответ на этот вопрос очевиден, и порой придётся подумать, порисовать, порефакторить. Но если вы описали методы вроде getElement и removeElement и свойства вроде element, как мы просили в предыдущих заданиях, то общими будут именно они.

2. Опишите абстрактный класс. Это точно такой же класс, как все остальные, только создавать объекты напрямую из него нельзя. От него можно только наследоваться. TS проведет нужные проверки за нас.

3. Далее объявите в абстрактном классе общие свойства и методы, пока что пустые.

4. У всех наших компонентов-наследников обязательно должен быть реализован метод getTemplate. Абстрактный класс поможет нам об этом не забыть. Теперь, если вы забудете в своём компоненте реализовать метод getTemplate, об этом вам напомнит ошибка при сборке.

5. А теперь унаследуем все наши компоненты от абстрактного класса с помощью языковой конструкции extends.

6. В заключение перенесём реализацию общих методов из потомков (наших компонентов) в родителя (абстрактный класс). Теперь, благодаря ООП и наследованию, у всех наших потомков будут методы, объявленные в родителе, а не дублированные в каждом компоненте.

   Не забудьте, что метод getTemplate должен остаться в компонентах, как и прочие частные методы.

Промежуточный контроль

Чтобы убедиться, что вы всё сделали правильно, запустите проект локально. Он должен сохранить свою полную работоспособность после всех ваших манипуляций.

6. Усиливаем абстракции

На этом шаге мы максимально абстрагируемся от работы с DOM напрямую в пользу работы с нашими компонентами.

1. У нас появились вспомогательные функции по работе с DOM (вроде render). Вынесите их в отдельный модуль, например, core/render.ts, чтобы не мешать их с другими вспомогательными функциями. А также измените реализацию этих функций, чтобы в них можно было передавать компоненты вместо DOM-элементов.
2. У нас осталось место, где мы работаем с DOM напрямую, — это подписка на события. Откажемся от прямого использования addEventListener в index.ts, оставив его только в компонентах. Для этого добавьте в компоненты методы для установки обработчиков событий, а в тело этих методов перенесите использование addEventListener. Глобальные обработчики — на document и window — остаются как есть. (пока обработчики не описаны до этого момента, но надо что-то придумать)

Теперь мы ничего не знаем о внутреннем устройстве компонентов и их реальных событиях. Важным для нас остаётся лишь их интерфейс — набор методов, которыми они обладают. Мы можем менять разметку и реальные события отдельных

компонентов безболезненно для всего приложения. Главное — сохранять интерфейс компонентов.

Промежуточный контроль

На данном этапе файлы нашего проекта приведены в более качественный вид. Мы разделили вспомогательные функции, вынесли общую логику в абстрактный класс, а также подготовили хороший плацдарм для работы с событиями.

7. Начинаем писать логику

Презентер

Пришло время разгрузить index.ts и вынести часть связанной логики в отдельную сущность — презентер. Задача презентера — создавать компоненты, добавлять их на страницу, навешивать обработчики. То есть реализовывать бизнес-логику и поведение приложения.

1. Создайте класс для презентера трека и класс для презентера списка треков (например, Track и TrackList). Либо вы можете создать единый компонент, который будет имплементировать логику трека и списка треков. Все зависит от вас.
2. Конструктор презентера должен принимать необходимые данные.
3. Разнесите по презентерам всю логику по отрисовке треков, а также по навешиванию на них обработчиков из index.ts.
4. В index.ts создайте экземпляры презентеров, передайте данные и инициализируйте их.

Частичный датабиндинг

Реализуем обработку кликов на кнопке «Like» (сердечко). Обработчики должны изменять данные трека — добавлять или удалять из избранного — и на основе изменённых данных перерисовывать компонент.

Для этого:

1. В компоненте трека добавьте метод для установки обработчика клика для сердечка.
2. В презентере списка треков опишите метод изменения данных. Задача метода — обновить моки и вызвать перерисовку конкретного трека.
3. В презентер трека передайте эту функцию изменения данных. Функция должна вызываться в обработчике клика и получать на вход обновлённые данные.
4. После изменения данных презентер должен перерендерить все компоненты.

> Можно реализовать более хитрый механизм перерендеривания, при котором будут перерисовываться только те компоненты, которые изменились. Этот вариант сложнее и не обязателен, но, если есть желание, можно поэкспериментировать :)

Промежуточный контроль

На данном этапе у нас появляется новая интерактивность в приложении. Это обработка кнопки лайка с последующим изменением данных и перерендериванием всех/изменившихся компонентов (зависит от вашего подхода).

8. Добавление/Удаление из плейлиста

Займёмся реализацией модального окна (pop-up) с дополнительными действиями. Наша задача — отображать модальное окно каждый раз, когда пользователь кликает на иконку многоточия у элемента трека. Есть несколько вариантов реализации подобного поведения, но у них будет общий кусок логики.

Для начала нужно отлавливать клик по многоточию трека. Делать это следует в презентере, а соответствующий EventListener навешивать во View элемента трека. Так как у каждого презентера трека будет свой отдельный EventListener, мы сразу же после клика знаем, по какому именно треку кликнул пользователь. Следующим шагом необходимо отрисовать модальное окно, и тут уже можно подойти двумя разными подходами.

Подход №1: просто и быстро

Пусть у каждого View трека уже будет находиться в шаблоне нужное нам модальное окно, но с одной особенностью — оно будет скрыто по умолчанию. При клике на многоточие достаточно будет просто сделать это модальное окно видимым (например: opacity: 0 → opacity: 1, в таком случае будет очень легко воспользоваться анимацией через transition). Позиционировать попап можно через position: relative у контейнера-родителя и position: absolute у самого попапа.

Останется в рамках этого же View добавить два метода для навешивания слушателей на удаление и добавление трека из плейлиста, а в презентере навесить соответствующие функции.

Проблему, когда на странице могут быть открыты сразу несколько модальных окон, можно решить через TrackListPresenter, в котором обрабатывать открытие модального окна, закрывая (если открыты) модальные окна у всех остальных треков.

Напомним, что в ТЗ четко описано, какие пункты должны отображаться в этом модальном окне. Перерисовывать «скрытое» модальное окно, соответственно, нужно при обновлении трека.

Подход №2: сложнее, но правильнее

В первом подходе есть небольшая проблема — одновременно в DOM’е существует большое количество экземпляров модального окна, что теоретически может вызывать подвисания при большом количестве треков на странице, при использовании приложения на слабых устройствах. Чтобы избежать подобной проблемы, можно пойти более сложным путём.

Создадим специальный SIngleton-сервис, который будет реализовывать в себе методы отрисовки попапа для конкретного трека. «Отрисовать попап» — это означает просто вставить вёрстку попапа в определённое место на странице. Правила те же, можно заранее доработать шаблон компонента трека, чтобы в него можно было вставить вёрстку модального окна с таким же позиционированием, как и в подходе №1. Единственной проблемой будет определение конкретного места, куда вставлять вёрстку попапа. Для этого можно сделать «хак» и доработать шаблон компонента трека идентификатором. Можно привязать его, например, к айди структуры трека. Таким образом, алгоритм будет выглядеть следующим образом:

- обрабатываем клик по многоточию по цепочке, передавая Id трека: TrackView → TrackPresenter → TrackListPresenter;
- из TrackListPresenter’а вызываем метод нашего singleton-сервиса, куда передаём Id трека;
- Singleton-сервис ищет точное место, куда нужно вставить попап (лучше воспользоваться сначала селектором по id’у для выбора конкретного трека на странице, а потом — по классу для выбора места конкретного трека для вставки шаблона попапа);
- Singleton-сервис запоминает, у какого предыдущего трека открывал попап (чтобы удалить его, если пользователь захочет открыть попап у другого трека).

Промежуточный контроль

При попытке открыть следующий попап алгоритм будет повторяться.

Функционал закрытия попапа по клику на свободное место вы сможете реализовать по желанию, самостоятельно.

9. Модель + удаление

Пришло время расширить функциональность нашего приложения. Сегодня нам предстоит решить сразу несколько задач: разобраться, как добавлять и удалять треки из плейлиста.

Модель данных

Прежде чем мы начнём реализовывать основную функциональность, нам нужно ввести в наше приложение модель данных для синхронизации треков между различными частями приложения.

1. Создайте директорию /src/model с новым файлом tracks.ts, в котором опишите класс с именем Tracks или любым другим именем на ваше усмотрение.
2. Добавьте в класс 2 метода: один для получения трека, другой для их записи.
3. Добавьте ещё один метод для обновления конкретного трека.
4. В index.ts создайте экземпляр модели и передайте в неё с помощью созданного на втором шаге метода записи моковые данные.
5. В index.ts передайте модель в конструктор презентера, а передачу моковых данных в метод init удалите.
6. В презентере замените прямую работу с моковым массивом треков на работу с моделью: для получения и обновления данных используйте соответствующие методы модели.

Удаление и добавление данных

Удаление и добавление треков можно реализовать разными способами. Предлагаем один из самых простых. Список треков уже умеет перерисовываться при изменении данных. Значит, для того, чтобы удалить трек, достаточно изменить данные (удалить из них конкретный трек).

1. Для удаления научим обработчик пользовательских действий принимать тип действия «Удаление». Логика следующая: заводим Enum со всеми возможными действиями пользователя, в обработчике пользовательских действий описываем switch по типу действия, где вызываем соответствующие методы модели.
2. Дальше добавим в компонент попапа обработчик события click по кнопке удаления, где вызовем функцию обновления данных с нужным типом Enum’a.

Промежуточный контроль

На данном этапе мы сделали сразу две важные вещи. Первая — данные теперь не находятся в непонятном состоянии. У нас появилась модель, которая хранит и следит за ними. Помимо этого, мы научили наше приложение удалять треки. А это ещё один шаг.

11. Окошко поиска

Реализуем фильтры отображаемых на экране треков и плейлистов. Мы будем реализовывать упрощенную версию, так как полноценное окно поиска с автокомплитом и модальным окном для выбора результата было бы слишком комплексным для нашего итогового проекта.

Для начала займёмся функционалом самого окошка. Окошко поиска уже должно находиться в вёрстке на странице. Нам остаётся лишь создать презентер и наполнить его необходимой логикой.

- Заведите в папке под функционал окошка поиска новый презентер. Добавьте в нём логику.
- Разберёмся с тем, когда инициализировать презентер. Окошко поиска всегда на странице, поэтому нет необходимости несколько раз заводить и удалять презентер динамически. Подумайте и примите решение: где необходимо инициализировать данный презентер?
- Логика фильтрации треков простая — для начала нужно передать наш фильтр в модель. Заведём в TracksModel в метод getTracks условие, которое будет фильтровать треки по строковому полю перед тем, как отдавать их из метода. Треки необходимо фильтровать по названию, исполнителю и альбому. Для плейлистов достаточно фильтровать по названию.
- Добавим в презентер поиска метод для установки коллбэка при изменении поля поиска. Теперь после инициализации презентера можно передать коллбэк, который будет синхронизировать поля для фильтра в моделях с тем, что пользователь вводит в строку поиска.
- Осталось реализовать внутри коллбэка связь с презентером треков/плейлистов для того, чтобы перерисовывать по актуальной модели сущности при изменении текстового поля поиска.

12. Смена плейлистов

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

1. Создайте компонент для экрана со списком плейлистов. Разметку вы найдёте в папке markup.

2. Подключите в index.ts и отрисуйте компонент с плейлистами. Пока что список треков и открытый плейлист покажутся друг под другом, сейчас мы с этим разберёмся.

3. Теперь, когда в index.ts есть все необходимые компоненты, реализуйте здесь логику переключения экранов при выборе соответствующего пункта в меню. Для удаления ненужного элемента можете использовать функцию remove. Для этого:

    1. Для начала реализуйте механизм смены экранов прямо в index.ts. Позже механизм можно вынести в отдельный презентер.
    2. Заведите Enum с полями, которые будут отвечать за каждый из экранов. Например, ScreenState.PlaylistList, ScreenState.Tracks, ScreenState.Playlist.
    3. Добавьте для компонента меню обработчики событий, которые позволят отслеживать выбор из плейлистов. После этого настройте компонент меню так, чтобы он рисовался по сгенерированным данным.
    4. Опишите функцию генерации плейлистов, срандомизируйте плейлисты. Штук 5–10.
    5. Отрисуйте меню со сгенерированными плейлистами и убедитесь, что при клике по пунктам меню соответствующие обработчики событий вызываются. Для каждого из эвентов можно завести отдельную сущность, которая будет вызываться.
    6. Теперь научите презентер плейлиста принимать текущий плейлист вместо того, чтобы генерировать его из моковых данных. Удобным способом отлавливайте эвент нажатия на пункты меню.
    7. По клику на плейлист в боковом меню передавайте соответствующую структуру плейлиста в презентер, запросите перерисовку презентера. На этом этапе выбранный в списке меню плейлист должен отображаться на экране при клике по нему.
4. Теперь разберёмся с переходом на экран списка плейлистов.

    1. Обработайте в отдельной функции клик в боковом меню по пунктам «Треки» и «Плейлисты».
    2. При выборе одного из этих пунктов отрисовывайте соответствующий экран и изменяйте состояние текущего экрана приложения.
5. Переместите логику управления и перерисовки экранов в отдельный презентер. Например, ScreenPresenter. А данные о текущем открытом экране можно хранить в отдельной модели.

Промежуточный контроль

Теперь мы умеем переключать плейлисты между собой. При этом меняется набор треков, который отображается на экране.

13. Добавление/удаление из плейлиста. Часть 2

На предыдущем шаге мы научили наше приложение отображать модальное окно с выбором действия над треком. С удалением из плейлиста всё более-менее понятно без раздумий. При нажатии на кнопку удаления нужно обновить модель (удалить из соответствующего плейлиста трек) и перерисовать весь список треков на странице. А вот с добавлением в плейлист придётся немного напрячься.

Новое модальное окно! Его нужно открывать над затемняющим оверлеем, позиционируя на всю страницу. В этом модальном окне необходимо реализовать выбор одного из плейлистов, а также кнопку «Отмена».

Для реализации подобного модального окна, которое может быть строго одно на странице, понадобится Singleton-сервис. Его задачей будет управление показом и скрытием окна, а также обработка действий пользователя в модальном окне. Алгоритм похож на более сложный подход из шага создания первого модального окна. По клику на плейлист необходимо дополнять модель плейлиста новым треком и, если вы пошли по более простому пути на предыдущем шаге, обновлять элементы треков на странице (в данном случае изменится шаблон «скрытого» модального окна).

14. Работа с API

Перед тем как начать работы по подключению API к приложению, вам необходимо скачать и запустить бекенд у себя. Мы подготовили для вас инструкцию по запуску и настройке.

1. Скачайте код бэкенда из репозитория (здесь будет ссылка на репозиторий, когда опубликуем его на gitlab). Весь бэкенд написан с использованием nestjs. Установите все зависимости с помощью команды **npm i** и запустите с помощью команды **npm start**.
2. Откройте страницу http://localhost:3000/api/, у вас должен открыться swagger с описанием запросов к api. Давайте рассмотрим, что это такое.

Swagger — это фреймворк для спецификации API. Как вы можете заметить, он показывает все возможные запросы к серверу, где указано, какие данные необходимо отправить и какой ответ придёт от сервера. Прямо на этой страничке можно пробовать выполнять запросы.

Прежде чем попробовать выполнить запрос, необходимо указать в swagger токен авторизации. Для этого нажмите кнопку **Authorize** и укажите токен авторизации. Вы можете указать любой, например **eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InN0cmluZyIsImlkIjoxLCJpYXQiOjE2NDgyMzgyOTQsImV4cCI6MTY0ODg0MzA5NH0.j85Ls9RmF7Lne43kESZMkXh5yCQtnggWk9zL4wFVnV0**.

После этого вы можете раскрыть любой из запросов и пробовать их отправлять. Как только вы это сделаете, ниже покажется ответ от сервера.

Это поможет вам пробовать обращаться к API без необходимости что-то программировать.

> Обратите внимание, бекенд принимает любые токены и создаёт для них пользователя. Отправляйте новый access token, если вы хотите создать нового пользователя. Это сделано, чтобы упростить взаимодействие с бэкэндом.

Инициализация

Для инициализации скачайте отдельно папку static, которая содержит статичные файлы БД. Разархивируйте папку так, чтобы она была доступна по пути /streaming_service_api/static. Ссылка на скачивание: https://drive.google.com/file/d/19Pmh_xuuXZYgqLCRpVuWxDXsfPcE0rnK/view?usp=sharing.

Когда папка будет на своём месте, можно выполнять команды по инициализации.

Миграция — процесс наполнения БД данными о музыке, исполнителях, обложках и так далее. Её необходимо сделать единожды при инициализации БД.

$ npm install

$ npm run migration

Запуск приложения

development

$ npm run start

Swagger

Для более удобной ориентации по API в рамках проекта добавлен пакет Swagger, который позволяет потыкать API в удобном графическом интерфейсе. После запуска бэкенда он будет доступен по адресу http://localhost:3000/api/.

Чтобы выполнять запросы, сначала авторизуйтесь. Для этого найдите роут /api/auth/register. В ответе сервера на запрос о регистрации вы получите токен. Скопируйте его и впишите в специальное поле, которое доступно в верхней части страницы. Нажмите кнопку Authorize. После этого вы будете считаться залогиненным пользователем и вам будут доступны все функции API.

Авторизацию в клиентском приложении необходимо делать отдельно. Эта инструкция нужна только для использования Swagger UI в целях знакомства с API.

Теперь, когда вы разобрались с бэкендом, можно приступать к подключению API к приложению.

15. Получение списка плейлистов и музыки

Реализуем отображение на экране списка плейлистов и треков, полученных с бекенда. Сейчас приложение отображает моковые данные. Но мы будем делать запросы на бекенд и отображать результат на страничке.

Для начала займемся функциональностью плейлистов. Отображение плейлистов уже должно находиться в вёрстке на странице. Нам остаётся лишь подключить их к api, а также наполнить его необходимой логикой.

Теперь, когда приложение умеет авторизовываться и получать access token, необходимо доработать презентеры для получения данных приложения.

1. Логика получения данных по api простая. Реализуйте получение списка плейлистов в презентере с помощью обращения к /api/users/playlists, используя axios, и поместите полученные данные в модель. Если вы всё сделали правильно, то у вас должны отобразиться плейлисты с бекенда.
2. В презентере со списком треков реализуйте получения списка треков в зависимости от активного плейлиста. Проверьте, что при переключении плейлиста отображаются нужные треки.
3. Добавьте в презентер треков установку и снятия лайка через api. Подумайте, когда лучше устанавливать или снимать лайк: до выполнения запроса или после выполнения запроса? Обработайте возможные ошибки при установке лайка (подсказка: в случае ошибки выполнения запроса статус лайка в модели необходимо откатить).
4. Добавьте аналогичную лайкам логику для добавления и удаления треков в плейлист. Убедитесь, что треки действительно добавляются и удаляются после перезагрузки страницы.

16. Работа с API музыки (плеер)

Реализуем воспроизведение и переключение треков для плеера. Мы будем реализовывать полноценное воспроизведение аудиофайлов в браузере, и для этого нам потребуется AudioContext.

AudioContext — это такой класс, который позволяет работать с аудио в браузере. Через него можно запускать воспроизведение аудио, останавливать, менять громкость и определять статус воспроизведения.

Сам плеер должен уметь воспроизводить треки, переключать треки, когда они заканчиваются, правильно отображать полосу прокрутки и изменять громкость воспроизведения.

Давайте реализуем всё это по порядку.

Отображение трека в плеере

Прежде чем реализовывать воспроизведение музыки с помощью браузерного API, необходимо хранить состояние о том, какая музыка сейчас должна проигрываться.

1. В папке с компонентом плеера создайте структуру данных, которая будет хранить в себе информацию о текущем треке, статус воспроизведения (проигрывается или не проигрывается) и текущее время воспроизведения в секундах.
2. Создайте модель для хранения и получения данных плеера.
3. В презентере плеера инициализируйте модель моковыми данными и передайте её в компонент при его создании.
4. Отобразите в плеере картинку трека, название текущего трека и исполнителя.
5. Отобразите общее время трека и текущее время воспроизведения. Подключите прогресс воспроизведения к данным из модели.
6. Отобразите кнопку воспроизведения в зависимости от данных в модели.

Воспроизведение треков

Теперь, когда отображение полностью готово, можно заняться воспроизведением треков.

1. Создайте обработчик клика по кнопке воспроизведения в компоненте.
2. Подключите обработчик клика к презентеру. Обработчик должен обновлять состояние в модели и перерисовывать компонент.
3. Создайте аудиоконтекст и звуковую ноду и поместите её в презентер. Убедитесь, что аудиоконтекст не находится в статусе suspended. Добавьте обработку статуса suspended, чтобы воспроизведение работало корректно. https://developer.chrome.com/blog/web-audio-autoplay/#policy-adjustments
4. Загрузите файл трека с api и поместите его в звуковую ноду. Обратите внимание, что вам необходимо использовать responseType arraybuffer, чтобы иметь возможность поместить файл в воспроизведение. Для простоты мы не будем реализовывать подгрузку трека лениво, поэтому трек должен быть загружен в браузер полностью.
5. Добавьте обработку изменения трека в модели: трек должен помещаться в звуковую ноду.
6. Реализуйте в презентере плеера запуск воспроизведение и остановку воспроизведения при клике на кнопку воспроизведения.
7. Добавьте отображение статуса воспроизведения в полосу прокрутки. Для этого необходимо получить текущее время воспроизведения из звуковой ноды. Добавьте обновление модели статуса воспроизведения, когда меняется текущее время воспроизведения. Добавьте перерисовку компонента при изменении текущего времени воспроизведения.
8. Добавьте обработчики для прогресса трека и громкости аудио и подключите их обработку через презентер. Не забудьте сохранять данные в модели.

17. Учим вьюшки перерисовываться

Перерисовка трека при обновлении его данных — уже сложное поведение. Но оно не часть бизнес-логики приложения. Это «бизнес-логика» самого компонента. Поэтому для реализации этой логики заведём Smart-компонент (от англ. smart — умный), который может себя перерисовывать.

1. Создайте абстрактный класс Smart, унаследовав его от AbstractComponent, с несколькими методами:
    - абстрактный метод repairHandlers, его нужно будет реализовать в наследнике. Его задача — восстанавливать обработчики событий после перерисовки;
    - обычный метод updateElement, его задачи:
        - удалить старый DOM-элемент компонента;
        - создать новый DOM-элемент;
        - поместить новый элемент вместо старого;
        - восстановить обработчики событий, вызвав repairHandlers.
    - обычный метод updateData, который будет обновлять данные и, если нужно, вызывать метод updateElement.
2. Унаследуйте компонент трека от Smart с пока пустым методом repairHandlers.
3. Теперь нужно реализовать перерисовку трека после взаимодействия с пользователем.
4. При перерисовке компонента все обработчики событий будут утеряны, поэтому их нужно будет навесить заново.

Инструкции по использованию инфраструктуры

Проект (фронтенд, бекенд и верстка) располагается в учебном гит репозитории.

Ссылка на макет в Figma располагается по ссылке:
https://www.figma.com/file/JIiEqozo3awHCmgn3xO8wU/TS_Graduate?node-id=0%3A1

Формат сдачи материалов и оценивание

Результат проведённой работы опубликуйте на учебный Gitlab (https://gitlab.skillbox.ru/). Также для проверки итогового проекта передайте проверяющим исходный код.

Готовый проект мы будем проверять по следующим критериям:

Основные

- Техническое задание выполнено в полном объёме.

- Приложение не зависает, при использовании приложения не возникает необработанных ошибок.

- Команды сборки и запуска дев-сервера корректно отрабатывают.

- Не используется неявное преобразование переменных. В случае необходимости преобразования оно используется явно. Например, нестрогое сравнение.

- В проекте отсутствуют потенциально нетипизированные места.
 Any, unknown и т. д. отсутствуют в тех местах, которые возможно типизировать.

- Сущности TS правильно именованы:

  - отсутствуют типы данных в названиях переменных, например, TracksArray, StateObject;

  - Enum — название в единственном числе с большой буквы. Его поля — с большой буквы.
 Правильно — AppState.Pause, AppState.Play.
 Неправильно — AppStates.pause, AppStates.play;

  - Type — название в единственном числе с большой буквы. Поля — согласно наименованию переменных;

  - Модификаторы полей — отсутствуют рудименты (_variable, $variable) при именовании переменных.

- Built-in types — отсутствует типизация через прототипы
 const a: number вместо const a: Number.

- Модули не экспортируют по умолчанию и не экспортируют изменяемые данные.

- Переопределённые методы должны содержать ключевое слово override в объявлении в классе-наследнике.

- Все переменные, которые передаются в конструктор класса, объявлены в нём же в качестве полей.

- Вместо **#приватных_полей** используются соответствующие модификаторы доступа.

- Для переменных, которые не должны изменяться вне конструктора, используется модификатор **readonly.**

- Поля класса, которые инициализируются, должны инициализироваться при объявлении (в конструкторе или при объявлении в шапке класса).

- Геттеры и сеттеры являются чистыми функциями. Если для них необходимо внешнее состояние, используется префикс internal или wrapped.

- Не используется конструктор new Array().

- Все переменные, которые не используются за пределами класса, имеют модификатор доступа private.

Дополнительным преимуществом для вашей работы будет соответствие следующим критериям:

- переменные и методы названы согласно следующим правилам:
  - в названии переменных присутствует существительное;
  - в названии методов/функций присутствует и, если возможно, лидирует глагол;
  - переменные названы на английском языке, без транслитерации и орфографических ошибок;
  - переменные не содержат в себе имён собственных;
  - классы именованы в UpperCamelCase;
  - глобальные константы именованы в CONSTANT_CASE.
- в проекте отсутствуют неиспользуемые участки кода;
- версии пакетов жёстко зафиксированы в package.json;
- обработчики событий навешиваются и удаляются по необходимости;
- функции и методы объявлены единообразно. Для обработчиков событий используются стрелочные функции, для методов — обычные;
- работа с массивами единообразна (используется либо только итерация через for...of, либо только методы массивов);
- отсутствует дублирующийся код;
- в коде отсутствуют комментарии (минимум комментариев);
- при объявлении переменных всегда используется Const, когда это возможно;
- если после блока if-else функция прекращает работу, блок else опускается;
- всегда используется строгое сравнение.