22. Учебный проект: меняй-удаляй (часть 1)

Рабочая ветка **module7-task1**

Задача

Пришло время расширить функциональность нашего приложения. Сегодня нам предстоит решить сразу несколько задач: реализовать фильтрацию и разобраться, как создавать и удалять задачи.

Модель данных

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

1. Создайте директорию `/src/models` с новым файлом `tasks.js`, в котором опишите класс `Tasks`.

2. Добавьте в класс 2 метода: один для получения задач, другой для их записи.

3. Добавьте ещё один метод для обновления конкретной задачи. Он должен принимать два параметра: `id` обновляемой задачи (если для этого понадобится добавить поле `id` в моковые данные, то сделайте это) и обновлённую задачу. Реализацию метода

заберите из `BoardController`.

4. В `main.js` создайте экземпляр модели и передайте в неё, с помощью созданного на втором шаге метода записи, моковые данные.

5. В `main.js` передайте в конструктор `BoardController` модель, а передачу моковых данных в метод `render` — удалите.

6. В `BoardController` замените работу с массивом задач на работу с моделью: для получения и обновления данных используйте соответствующие методы модели.

Фильтрация

В этой части задания мы запрограммируем фильтры.

1. На первом шаге классика — компонент фильтров у нас уже есть, нужен контроллер для него. С конструктором и методом `render`. Конструктор принимает контейнер и модель данных. После создания экземпляра контроллера в `main.js`, вызовите у него метод `render` для отрисовки.

2. Для реализации фильтрации модернизируйте модель: добавьте в неё метод для активации фильтра. Измените метод модели для получения данных, чтобы он учитывал

активный фильтр. Активируйте фильтр в модели при взаимодействии пользователя с интерфейсом.

3. Но как `BoardController` будет узнавать об изменении активного фильтра? Подход уже вам известен:

   - в модель добавьте метод для установки обработчика изменения активного фильтра;

   - при активации фильтра в модели вызовите установленный обработчик;

   - в `BoardController` передайте ей обработчик, который будет получать данные от модели и вызывать перерисовку.

4. Осталось реализовать обновление счётчика у фильтров при изменении, а в будущем добавление и удаление, данных:

   - в модель добавьте метод для установки обработчика изменения данных;

   - в контроллере фильтров передайте в модель обработчик, пересчитывающий количество задач по каждому фильтру и вызывающий перерисовку

компонента фильтров.

Обратите внимание, при переключении фильтров должны сбрасываться сортировка и счётчик показанных задач («Load more»). Подробности в техническом задании.

Удаление и добавление данных

Удаление и добавление задач можно реализовать разными способами. Предлагаем один из самых простых. Наша доска уже умеет перерисовываться при изменении данных. Значит для того, чтобы удалить задачу с доски, достаточно изменить данные (удалить из них конкретную задачу).

1. Для удаления научим функцию `onDataChange` принимать в качестве обновлённых данных `null`. Логика следующая: если вместо новых данных пришёл `null`, то старые данные нужно удалить.

2. Дальше добавим в компонент формы редактирования обработчик события `click` по кнопке удаления, где вызовем функцию `onDataChange` и передадим ей в качестве новых данных `null`.

3. Теперь провернём всё то же самое, только для добавления — то есть `null` должен приходить в `onDataChange` вместо старых данных, а новые данные должны добавляться в модель (реализуйте в модели

соответствующие методы).

4. Затем нам нужно добавить обработчик на кнопку «Add new task», по клику на которую нужно показать форму добавления новой задачи. В качестве формы добавления нужно использовать форму редактирования, поэтому должно сохраниться всё поведение контролов и валидации. Кроме того, добавьте условие, чтобы можно было открыть только одну форму добавления.

Обратите внимание, при попытке пользователя добавить новую задачу, должны сбрасываться фильтры, сортировка и счётчик показанных задач («Load more»); скрываться без сохранения любая показанная форма редактирования. Подробности в техническом задании.

5. В конце останется лишь обработать отправку формы.

Обратите внимание, при закрытии формы добавления (любым способом) введённая информация не сохраняется.

Безопасность превыше всего

Мы отлично поработали! Данные в нашем приложении создаются, удаляются, изменяются... пользователями. А где есть пользовательский ввод, там потенциальная дыра в безопасности. Поэтому пора подумать о том, как обезопасить приложение и добропорядочных пользователей от «хакеров».

Установите из npm и подключите в проект библиотеку для превращения в строку возможного HTML или JavaScript кода в пользовательском вводе. Мы рекомендуем библиотеку `he`.

Обработайте с помощью этой библиотеки описание задачи.

24. Личный проект: меняй-удаляй (часть 1)

Задача

Киноман

Пришло время расширить функциональность нашего приложения. Сегодня нам предстоит решить сразу несколько задач: реализовать фильтрацию и разобраться, как удалять комментарии.

Модель данных

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

1. Создайте директорию `/src/models` с новым файлом `movies.js`, в котором опишите класс `Movies`.

Обратите внимание, что если вы следовали нашим рекомендациям и выделили комментарии в отдельную структуру, для них нужно завести отдельную модель и провести похожие манипуляции.

2. Добавьте в класс 2 метода: один для получения фильмов, другой для их записи.

3. Добавьте ещё один метод для обновления конкретного фильма. Он должен принимать два параметра: `id` обновляемого фильма (если для этого понадобится добавить поле `id` в моковые данные, то сделайте это) и обновлённый фильм. Реализацию метода заберите из `PageController`.

4. В `main.js` создайте экземпляр модели и передайте в неё, с помощью созданного на втором шаге метода записи, моковые данные.

5. В `main.js` передайте в конструктор `PageController` модель, а передачу моковых данных в метод `render` — удалите.

6. В `PageController` замените работу с массивом фильмов на работу с моделью: для получения и обновления данных используйте соответствующие методы модели.

Фильтрация

В этой части задания мы запрограммируем фильтры.

1. На первом шаге классика — компонент фильтров у нас уже есть, нужен контроллер для него. С конструктором и методом `render`. Конструктор принимает контейнер и модель данных. После создания экземпляра контроллера

в `main.js`, вызовите у него метод `render` для отрисовки.

2. Для реализации фильтрации модернизируйте модель: добавьте в неё метод для активации фильтра. Измените метод модели для получения данных, чтобы он учитывал активный фильтр. Активируйте фильтр в модели при взаимодействии пользователя с интерфейсом.

3. Но как `PageController` будет узнавать об изменении активного фильтра? Подход уже вам известен:

   - в модель добавьте метод для установки обработчика изменения активного фильтра;

   - при активации фильтра в модели вызовите установленный обработчик;

   - в `PageController` передайте ей обработчик, который будет получать данные от модели и вызывать перерисовку.

4. Осталось реализовать обновление счётчика у фильтров при изменении, а в будущем добавление и удаление, данных:

- в модель добавьте метод для установки обработчика изменения данных;

- в контроллере фильтров передайте в модель обработчик, пересчитывающий количество фильмов по каждому фильтру и вызывающий перерисовку компонента фильтров.

Обратите внимание, при переключении фильтров должны сбрасываться сортировка и счётчик показанных фильмов («Show more»). Подробности в техническом задании.

Удаление и добавление данных

Удаление и добавление комментариев можно реализовать разными способами. Предлагаем один из самых простых. Попап с подробной информацией о фильме уже умеет перерисовываться при изменении данных. Значит для того, чтобы удалить комментарий, достаточно изменить данные (удалить из них конкретный комментарий).

1. Для удаления научим функцию `onDataChange` принимать в качестве обновлённых данных `null`. Логика следующая: если вместо новых данных пришёл `null`, то старые данные нужно удалить.

Обратите внимание, что если вы следовали нашим рекомендациям и выделили комментарии в отдельную структуру, для них стоит завести отдельную функцию по типу `onDataChange`, чтобы не городить условия в одной функции.

2. Дальше добавим в попап обработчик события `click` по кнопке удаления комментария, где вызовем функцию обновления данных и передадим ей в качестве новых данных `null`.

Обратите внимание, пользователь может удалять любые комментарии к фильму.

3. Теперь провернём всё то же самое, только для добавления — то есть `null` должен приходить в метод обновления данных вместо старых данных, а новые данные должны добавляться в модель (реализуйте в модели соответствующие методы).

4. В конце останется лишь обработать отправку формы.

Обратите внимание, при закрытии попапа (любым способом) введённая информация не сохраняется.

Безопасность превыше всего

Мы отлично поработали! Данные в нашем приложении создаются, удаляются, изменяются... пользователями. А где есть пользовательский ввод, там потенциальная дыра в безопасности. Поэтому пора подумать о том, как обезопасить приложение и добропорядочных пользователей от «хакеров».

Установите из npm и подключите в проект библиотеку для превращения в строку возможного HTML или JavaScript кода в пользовательском вводе. Мы рекомендуем библиотеку `he`.

Обработайте с помощью этой библиотеки текст комментария.

Bigtrip

Пришло время расширить функциональность нашего приложения. Сегодня нам предстоит решить сразу несколько задач: реализовать фильтрацию и разобраться, как создавать и удалять точки маршрута.

Модель данных

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

1. Создайте директорию `/src/models` с новым файлом `points.js`, в котором опишите класс `Points`.

2. Добавьте в класс 2 метода: один для получения точек маршрута, другой для их записи.

3. Добавьте ещё один метод для обновления конкретной точки маршрута. Он должен принимать два параметра: `id` обновляемой точки маршрута (если для этого понадобится добавить поле `id` в моковые данные, то сделайте это) и обновлённую точку маршрута. Реализацию метода заберите из `TripController`.

4. В `main.js` создайте экземпляр модели и передайте в неё, с помощью созданного на втором шаге метода записи,

моковые данные.

5. В `main.js` передайте в конструктор `TripController` модель, а передачу моковых данных в метод `render` — удалите.

6. В `TripController` замените работу с массивом точек маршрута на работу с моделью: для получения и обновления данных используйте соответствующие методы модели.

Фильтрация

В этой части задания мы запрограммируем фильтры.

1. На первом шаге классика — компонент фильтров у нас уже есть, нужен контроллер для него. С конструктором и методом `render`. Конструктор принимает контейнер и модель данных. После создания экземпляра контроллера в `main.js`, вызовите у него метод `render` для отрисовки.

2. Для реализации фильтрации модернизируйте модель: добавьте в неё метод для активации фильтра. Измените метод модели для получения данных, чтобы он учитывал активный фильтр. Активируйте фильтр в модели при взаимодействии пользователя с интерфейсом.

3. Но как `TripController` будет узнавать об изменении активного фильтра? Подход уже вам известен:

    - в модель добавьте метод для установки обработчика изменения активного фильтра;
    - при активации фильтра в модели вызовите установленный обработчик;
    - в `TripController` передайте ей обработчик, который будет получать данные от модели и вызывать перерисовку.

Обратите внимание, при переключении фильтров должна сбрасываться сортировка. Подробности в техническом задании.

Удаление и добавление данных

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

1. Для удаления научим функцию `onDataChange` принимать в качестве обновлённых данных `null`. Логика следующая: если вместо новых данных пришёл `null`, то старые

данные нужно удалить.

2. Дальше добавим в компонент формы редактирования обработчик события `click` по кнопке удаления, где вызовем функцию `onDataChange` и передадим ей в качестве новых данных `null`.

3. Теперь провернём всё то же самое, только для добавления — то есть `null` должен приходить в `onDataChange` вместо старых данных, а новые данные должны добавляться в модель (реализуйте в модели соответствующие методы).

4. Затем нам нужно добавить обработчик на кнопку «New Event», по клику на которую нужно показать форму добавления новой точки маршрута. В качестве формы добавления нужно использовать форму редактирования, поэтому должно сохраниться всё поведение контролов и валидации. Кроме того, добавьте условие, чтобы можно было открыть только одну форму добавления.

Обратите внимание, при попытке пользователя добавить новую точку маршрута должны сбрасываться фильтры и сортировка; скрываться без сохранения любая показанная форма редактирования. Подробности в техническом задании.

5. В конце останется лишь обработать отправку формы.

Обратите внимание, при закрытии формы добавления (любым способом) введённая информация не сохраняется.

Безопасность превыше всего

Мы отлично поработали! Данные в нашем приложении создаются, удаляются, изменяются... пользователями. А где есть пользовательский ввод, там потенциальная дыра в безопасности. Поэтому пора подумать о том, как обезопасить приложение и добропорядочных пользователей от «хакеров».

Ограничьте возможность ввода в поле «Пункт назначения» конечным списком городов. Для удобства пользователя в разметке используется `datalist` для упрощения ввода. Конечный список городов строится на основании отдельной структуры данных со списком всех возможных пунктов назначения (сейчас моковой, в дальнейшем — с сервера).

Запретите возможность ввода в поле «Цена» любых значений, кроме числовых.