16. Учебный проект: большие перемены (Часть 1)

Задача

В этом задании мы добавим в наши компоненты работу с данными и сделаем так, чтобы при изменении информации внутри компонента, изменялись и моковые данные из которых эти компоненты были созданы.

Сперва разберемся с кодом, который скопился у нас в `BoardController` и отвечает за смену задачи на форму редактирования задачи. Для этого создадим `TaskController`:

1. В нем нужно описать конструктор и метод `render`.

2. Конструктор должен принимать `container` — элемент, к которому он будет аппендить задачу и форму редактирования.

3. Метод `render` должен принимать данные одной задачи. Так же в него должен переехать код, который отвечает за отрисовку задачи, ее замену на форму редактирования и наоборот, а также установка связанных с этим обработчиков событий.

Обновим компонент на основе измененных данных

Реализуем обработку кликов на кнопках «Favorites», «Archive» у карточки задачи. Обработчики должны изменять данные задачи — добавлять/удалять из избранного и помечать задачу выполненной/невыполненной. И на основе измененных данных перерендеривать компоненты. Для этого:

1. В компоненте задачи добавьте методы для установки обработчиков клика для каждой кнопки.

2. В конструкторе `TaskController` опишите новый входной параметр — функцию `onDataChange`. Эта функция должна вызываться в обработчике клика и получать на вход старую задачу и измененную задачу. Саму функцию мы опишем на следующем шаге.

3. В `BoardController` опишите метод `_onDataChange`, который должен принимать старую задачу и новые данные. А на основе этой информации обновлять моки

и вызывать метод `render` у конкретного экземпляра `TaskController`, передав в него новую задачу.

4. Чтобы все заработало передайте метод `_onDataChange` в `TaskController` при создании его экземпляра.

Обновим компонент на основе действий пользователя

Форма редактирования довольно сложный интерактивный компонент. Но это поведение — не часть бизнес-логики приложения. Это бизнес-логика самого компонента. Поэтому для реализации этой логики введем понятие `SmartComponent` — компонент, который может себя перерендеривать.

1. Создайте абстрактный класс `AbstractSmartComponent` (унаследовав его то `AbstractComponent`) с двумя методами:

1. Абстрактный метод `recoveryListeners` — его нужно будет реализовать в наследнике. Его задача — восстанавливать обработчики событий после перерендеринга.

2. Обычный метод `rerender` в котором нужно:

- удалить старый DOM-элемент компонента;
- создать новый DOM-элемент;
- поместить новый элемент вместо старого;
- восстановить обработчики событий, вызвав `recoveryListeners`.

2. Унаследуйте компонент формы редактирования от `AbstractSmartComponent` используя конструкцию языка `extends` и добавте в него метод `recoveryListeners`. Реализацию пока писать не нужно, оставьте тело метода пустым.

3. Теперь нужно реализовать перерендеривание формы редактирования после взаимодействие с пользователем:

- показ/скрытие инпута для ввода даты исполнения;
- показ/скрытие дней повторения задачи;

- блокировка кнопки Save;

4. При перерендеривании компонента все обработчики событий будут потеряны. Поэтому в методе `recoveryListeners` их нужно восстановить.

Отобразим только 1 форму редактирования

Мы научились создавать интерактивные компоненты. Мы научились обновлять данные. Осталось реализовать малое — запретить открывать несколько форм редактирования одновременно. Мы реализуем это простым принципом — перед тем, как `TaskController` будет переводить задачу в форму редактирования, мы пошлем приказ всем экземплярам `TaskController` отобразить дефолтное состояние. Итак:

1. Добавте метод `setDefaultView` в `TaskController` для отображения задачи вместо формы редактирования.

2. Добавте в конструктор `TaskController` новый параметр — функцию `onViewChange` и вызывайте ее перед тем, как произойдет смена карточки задачи на форму редактирования.

3. В `BoardController` опишите метод `_onViewChange`. В нем вызывите у всех экземпляров `TaskController` метод `setDefaultView`.

4. В `BoardController` при создании экземпляров `TaskController` передайте в их конструктор метод `_onViewChange`.

Выполненные пункты ТЗ

Зачёркнутые пункты мы разберём на следующей лекции.

- В карточке пользователю доступна кнопка «Favorite». При нажатии на кнопку, задача добавляется в избранное. Повторное нажатие приводит к удалению из избранного. ~~При этом сразу же происходит пересчет количества задач в избранном (пункт ТЗ 1.4 «Фильтры»).~~

- В карточке пользователю доступна кнопка «Archive». При нажатии на кнопку, задача помечается выполненной. ~~При этом к ней сразу же применяется действие фильтров (пункт ТЗ 1.4 «Фильтры»).~~

- Кнопки «Favorite» и «Archive» отражает состояние задачи с помощью установки/удаления css-класса `card__btn--disabled`.

- (Рассматриваем этот пункт в контексте редактировария) Менеджер задач позволяет создавать задачи двух видов:

  - задачи без повторения и регулярные задачи — повторяемые по определённым дням недели;

  - для регулярных задач обязательно должен быть выбран день (дни) недели для повторения и скрыто поле «Дата исполнения»;

  - в случае несоблюдения условий задача не может быть сохранена — кнопка «Save» заблокирована.

- В форме создания пользователь заполняет поля:

  - Дни недели повтора задачи;

  - Флаг повторения.

- Если у пользователя открыта задача для редактирования/создания и пользователь открывает другую задачу для редактирования/создания, то первая задача переходит в режим просмотра (пропадает в случае создания). Несохраненные данные при этом теряются.

Задание можно отправить на проверку только после привязки к нему пулреквеста, отправленного из ветки `module6-task1`.

17. Личный проект: большие перемены (Часть 1)

Задание

В этом задании мы добавим в наши компоненты работу с данными и сделаем так, чтобы при изменении информации внутри компонента, изменялись и моковые данные из которых эти компоненты были созданы.

Большое путешествие

Сперва разберемся с кодом, который скопился у нас в `TripController` и отвечает за смену точки маршрута на форму редактирования. Для этого создадим `PointController`:

1. В нем нужно описать конструктор и метод `render`.

2. Конструктор должен принимать `container` — элемент, к которому он будет аппендить точку маршрута и форму редактирования.

3. Метод `render` должен принимать данные одной точки маршрута. Так же в него должен переехать код, который отвечает за отрисовку точки маршрута, ее замену на форму редактирования и наоборот, а также установка связанных с этим обработчиков событий.

Обновим компонент на основе измененных данных

Реализуем обработку кликов на кнопке «Favorite» (отображается в виде звёздочки) у формы редактирования. Обработчики должны изменять данные точки маршрута — добавлять/удалять из избранного. И на основе измененных данных перерендеривать компоненты. Для этого:

1. В компоненте формы редактирования добавьте метод для установки обработчика клика для кнопки.

2. В конструкторе `PointController` опишите новый входной параметр — функцию `onDataChange`. Эта функция должна вызываться в обработчике клика и получать на вход старую точку маршрута и измененную точку маршрута. Саму функцию мы опишем на следующем шаге.

3. В `TripController` опишите метод `_onDataChange`, который должен принимать старую точку маршрута и новые данные. А на основе этой информации обновлять моки и вызывать метод `render` у конкретного экземпляра `PointController`, передав в него новую точку маршрута.

4. Чтобы все заработало передайте метод `_onDataChange` в `PointController` при создании его экземпляра.

Обновим компонент на основе действий пользователя

Форма редактирования довольно сложный интерактивный компонент. Но это поведение — не часть бизнес-логики приложения. Это бизнес-логика самого

компонента. Поэтому для реализации этой логики введем понятие `SmartComponent` — компонент, который может себя перерендеривать.

1. Создайте абстрактный класс `AbstractSmartComponent` (унаследовав его то `AbstractComponent`) с двумя методами:

1. Абстрактный метод `recoveryListeners` — его нужно будет реализовать в наследнике. Его задача — восстанавливать обработчики событий после перерендеринга.

2. Обычный метод `rerender` в котором нужно:

- удалить старый DOM-элемент компонента;
- создать новый DOM-элемент;
- поместить новый элемент вместо старого;
- восстановить обработчики событий, вызвав `recoveryListeners`.

2. Унаследуйте компонент формы редактирования от `AbstractSmartComponent` используя конструкцию языка `extends` и добавьте в него метод `recoveryListeners`. Реализацию пока писать не нужно, оставьте тело метода пустым.

3. Теперь нужно реализовать перерендеривание формы редактирования после взаимодействие с пользователем:

- при смене типа поездки нужно менять плейсхолдер у пункта назначения и менять набор дополнительных опцией.
- при выборе пункта назначения нужно менять описание.

4. При перерендеривании компонента все обработчики событий будут потеряны. Поэтому в методе `recoveryListeners` их нужно восстановить.

Отобразим только 1 форму редактирования

Мы научились создавать интерактивные компоненты. Мы научились обновлять данные. Осталось реализовать малое — запретить открывать несколько форм редактирования одновременно. Мы реализуем это простым принципом — перед

тем, как `PointController` будет переводить точку маршрута в форму редактирования, мы пошлем приказ всем экземплярам `PointController` отобразить дефолтное состояние. Итак:

1. Добавте метод `setDefaultView` в `PointController` для отображения точки маршрута вместо формы редактирования.

2. Добавьте в конструктор `PointController` новый параметр — функцию `onViewChange` и вызывайте ее перед тем, как произойдет смена точки маршрута на форму редактирования.

3. В `TripController` опишите метод `_onViewChange`. В нем вызовите у всех экземпляров `PointController` метод `setDefaultView`.

4. В `TripController` при создании экземпляров `PointController` передайте в их конструктор метод `_onViewChange`.

Выполненные пункты [Т3]

- Для создания новой точки маршрута пользователь заполняет:

  - Тип точки маршрута (один из: `Taxi`, `Bus`, `Train`, `Ship`, `Transport`, `Drive`, `Flight`, `Check`, `Sightseeng`, `Restaurant`).

  - Пункт назначения. Выбирается из списка предложенных значений, полученных с сервера. Пользователь не может ввести свой вариант для пункта назначения.

  - Кнопка «Favorite» (отображается в виде звёздочки). Добавляет точку маршрута в избранное. После добавления в избранное, кнопка меняет состояние — закрашенная звезда. Повторный клик по кнопке удаляет точку маршрута из избранного и возвращает кнопку в исходное состояние. Добавление/удаление в избранное происходит сразу и не зависит от нажатия кнопок «Save».

- Одновременно может быть открыта только одна форма создания/редактирования.

- Если пользователь внёс изменения в точку маршрута, не выполнил сохранение и пытается перейти к редактированию другой точки маршрута либо создать новую, то в этом случае первая форма редактирования закрывается без сохранения изменений. Открывается форма создания/редактирования второй точки маршрута.

Задание можно отправить на проверку только после привязки к нему пулреквеста, отправленного из ветки `module6-task1`.

Киноман

Сперва разберемся с кодом, который скопился у нас в `PageController` и отвечает за показ попапа с подробной информацией о фильме и его закрытие. Для этого создадим `MovieController`:

1. В нем нужно описать конструктор и метод `render`.

2. Конструктор должен принимать `container` — элемент, к которому он будет аппендить карточку фильма и попап с подробной информацией.

3. Метод `render` должен принимать данные одной карточки фильма. Также в него должен переехать код, который отвечает за отрисовку карточки фильма, показ попапа с подробной информацией о фильме и его закрытие, а также установка связанных с этим обработчиков событий.

Обновим компонент на основе измененных данных

Реализуем обработку кликов на кнопках «Add to watchlist», «Already watched», «Add to favorites» у карточки фильма и у попапа подробной информации о фильме. Обработчики должны изменять данные фильма — добавлять/удалять из избранного и списка к просмотру, а так же помечать его просмотренным. И на основе измененных данных перерендеривать компоненты. Для этого:

1. В компоненте фильма и попапе добавьте методы для установки обработчиков клика для каждой кнопки.

2. В конструкторе `MovieController` опишите новый входной параметр — функцию `onDataChange`. Эта функция должна вызываться в обработчике клика и получать на вход старую карточку фильма и измененную карточку фильма. Саму функцию мы опишем на следующем шаге.

3. В `PageController` опишите метод `_onDataChange`, который должен принимать старую карточку фильма и новые данные. А на основе этой информации обновлять моки и вызывать метод `render` у конкретного экземпляра `MovieController`, передав в него новую карточку фильма.

4. Чтобы все заработало передайте метод `_onDataChange` в `MovieController` при создании его экземпляра.

Обновим компонент на основе действий пользователя

Попап подробной информации довольно сложный интерактивный компонент. Но это поведение — не часть бизнес-логики приложения. Это бизнес-логика самого компонента. Поэтому для реализации этой логики введем понятие `SmartComponent` — компонент, который может себя перерендеривать.

1. Создайте абстрактный класс `AbstractSmartComponent` (унаследовав его то `AbstractComponent`) с двумя методами:

1. Абстрактный метод `recoveryListeners` — его нужно будет реализовать в наследнике. Его задача — восстанавливать обработчики событий после перерендеринга;

2. Обычный метод `rerender` в котором нужно:

- удалить старый DOM-элемент компонента;
- создать новый DOM-элемент;
- поместить новый элемент вместо старого;
- восстановить обработчики событий, вызвав `recoveryListeners`.

2. Унаследуйте компонент попапа подробной информации от `AbstractSmartComponent` используя конструкцию языка `extends` и добавьте в него метод `recoveryListeners`. Реализацию пока писать не нужно, оставьте тело метода пустым.

3. Теперь нужно реализовать перерендеривание попапа после взаимодействие с пользователем:

- при пометке фильма как просмотренного будет показывать блок с предложением оценить фильм (рейтинг). Это должно работать и в обратном порядке, если пользователь удалил фильм из просмотренных, то блок с рейтингом должен скрыться, а оценка пользователя удалиться.
- при клике на эмоцию будет подставлять её в соответствующий блок.

4. При перерендеривании компонента все обработчики событий будут потеряны. Поэтому в методе `recoveryListeners` их нужно восстановить.

Отобразим только 1 попап

Мы научились создавать интерактивные компоненты. Мы научились обновлять данные. Осталось реализовать малое — запретить открывать несколько попапов одновременно. Мы реализуем это простым принципом — перед тем, как `MovieController` будет переводить карточку фильма в попап, мы пошлем приказ всем экземплярам `MovieController` отобразить дефолтное состояние. Итак:

1. Добавьте метод `setDefaultView` в `MovieController` для отображения карточки фильма вместо попапа подробной информации.

2. Добавьте в конструктор `MovieController` новый параметр — функцию `onViewChange` и вызывайте ее перед тем, как произойдет смена карточки фильма на попап подробной информации.

3. В `PageController` опишите метод `_onViewChange`. В нем вызовите у всех экземпляров `MovieController` метод `setDefaultView`.

4. В `PageController` при создании экземпляров `MovieController` передайте в их конструктор метод `_onViewChange`.

Выполненные пункты ТЗ

- При наведении курсора мыши на блок карточки фильма появляются дополнительные кнопки управления:
  - «Add to watchlist» — добавляет фильм в список к просмотру;
  - «Already watched» — помечает фильм как просмотренный;
  - «Add to favorites» — добавляет/удаляет фильм в избранное.
- Фильм может относится к нескольким жанрам. Если фильм относится к нескольким жанрам, выводите «Genres», иначе «Genre».
- Одновременно может быть открыт только один попап. При открытии нового попапа, прежний закрывается. Несохранённые изменения (неотправленный комментарий) пропадают.

- (Реализовано только отображение блока с оценкой) Выставить оценку фильму можно в попапе:

  - Если пользователь отметил фильм как просмотренный («Already watched»), то между блоком с описанием фильма и блоком с комментариями отображается дополнительный блок «How you feel it», где пользователь может выставить оценку фильму.

Задание можно отправить на проверку только после привязки к нему пулреквеста, отправленного из ветки `module6-task1`.

18. Учебный проект: большие перемены (Часть 2)

Задача

В этом задании мы подключим библиотеки, которые помогут нам эффективно изменять и отображать поля с датой.

1. Установите из npm и подключите библиотеку `flatpickr`.

2. Импортируйте в компонент с формой редактирования стили для `flatpickr` из `node_modules`.

3. Уставновите из npm пакеты `style-loader` и `css-loader`. Они потребуются для сборки стилей. Допишите в конфиг Webpack конструкцию, которая поможет собрать стили `flatpickr` в проект:

```
module: {
  rules: [
    {
      test: /\.css$/i,
      use: ['style-loader', 'css-loader'],
    },
```

```
    ],

},
```

4. Настройте `flatpickr` так, чтобы выбор даты в форме редактирования осуществлялся с его помощью.

5. Установите из npm и подключите библиотеку

Выполненные пункты ТЗ

- Дата исполнения:

  - выбор времени и даты осуществляется с помощью библиотеки `flatpickr.js`

Задание можно отправить на проверку только после привязки к нему пулреквеста, отправленного из ветки `module6-task2`.

19. Личный проект: большие перемены (Часть 2)

Задача

В этом задании мы подключим библиотеки, которые помогут нам эффективно работать с датой.

Большое путешествие

1. Установите из npm и подключите библиотеку `flatpickr`.

2. Импортируйте в компонент с формой редактирования стили для `flatpickr` из `node_modules`.

3. Установите из npm пакеты `style-loader` и `css-loader`. Они потребуются для сборки стилей. Допишите в конфиг Webpack конструкцию, которая поможет собрать стили `flatpickr` в проект:

```
module: {
  rules: [
    {
      test: /\.css$/i,
      use: ['style-loader', 'css-loader'],
    },
  ],
},
```

4. Настройте `flatpickr` так, чтобы выбор даты в форме редактирования осуществлялся с его помощью.

5. Установите из npm и подключите библиотеку `moment.js`. С её помощью посчитайте длительность каждой точки маршрута и выведите даты в проекте.

Выполненные пункты ТЗ

- Дата и время начала события. Выбор времени и даты осуществляется с помощью библиотеки `flatpickr.js`. Выбранная дата и время отображаются в поле в формате: `DD/MM/YYYY HH:mm`. Например: `25/12/2019 16:00`.

- Дата и время окончания события. Выбор времени и даты осуществляется с помощью библиотеки `flatpickr.js`. Выбранная дата и время отображаются в поле в формате: `DD/MM/YYYY HH:mm`. Например: `29/12/2019 16:55`. Дата окончания не может быть меньше даты начала события.

- В колонке «Time» отображается время и продолжительность нахождения в точке маршрута (разность между окончанием и началом события). Время маршрута отображается в формате «Начало» — «Окончание» (10:30 — 11:00). Формат продолжительности нахождения в точке маршрута зависит от длительности:

  - Менее часа: 00M (23M);

- Менее суток: 00H 00M (02H 44M);
- Более суток: 00D 00H 00M (01D 02H 30M).

Задание можно отправить на проверку только после привязки к нему пулреквеста, отправленного из ветки `module6-task2`.

Киноман

1. Установите из npm и подключите библиотеку `moment.js`. Выведите с её помощью даты релиза фильма и даты комментариев.

Выполненные пункты ТЗ

- Попап содержит расширенную информацию о фильме:
  - Дата и год релиза в формате «DD MMMM YYYY» (например: 01 April 1995);
  - Продолжительность (в формате «1h 36m»).

Задание можно отправить на проверку только после привязки к нему пулреквеста, отправленного из ветки `module6-task2`.