**Лекция 7**

**Учебный проект:**

Пришло время избавиться от мок и получить информацию для игры с реального сервера. Начнём с получения списка вопросов и заодно познакомимся с новыми инструментами: axios, recompose и redux-thunk.

Задача

Установите пакеты axios, recompose и redux-thunk с помощью команды: npm i axios recompose redux-thunk -E.

Отключите и удалите файл с моками вопросов.

Создайте новый модуль api.js. Подключите в нём axios и опишите функцию, принимающую dispatch в качестве параметра и возвращающую сконфигурированный инстанс axios:

baseURL — <https://htmlacademy-react-2.appspot.com/guess-melody>,

timeout — 5000,

withCredentials — true.

Откройте файл src/index.js. Подключите в нём redux-thunk из одноимённого пакета и compose из recompose. Также нам потребуется импортировать applyMiddleware из пакета redux и созданную в предыдущем шаге функцию для получения сконфигурированного экземпляра axios.

В этом же файле (src/index.js) создайте новую переменную и сохраните в неё экземпляр настроенного axios. Например:

// configreAPI - функция из модуля api.js,

// созданная в третьем шаге.

const api = configureAPI((...args) => store.dispatch(...args));

Обновите код инициализации хранилища (мы по-прежнему в файле src/index.js). Добавьте подключение thunk:

const store = createStore(

reducer,

compose(

applyMiddleware(thunk.withExtraArgument(api)),

window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ ? window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_() : (f) => f

)

);

Обратите внимание, чтобы не передавать в createStore пустую функцию, если React DevTools не установлены, можно воспользоваться продвинутым способом настройки стора из документации.

Напишите код для загрузки списка вопросов с сервера. Для получения списка вопросов с сервера, выполните GET запрос к ресурсу /questions. Загруженный список вопросов поместите в store. Создайте для этого соответствующий action и action creator.

Не забудьте протестировать новую функциональность. Чтобы замокать axios установите дополнительный пакет axios-mock-adapter с помощью команды npm i axios-mock-adapter -DE.

Немного рефакторинга

Пришло время немного порефакторить код. Для начала разбейте редьюсер, разделив зоны ответственности: состояние игры, загружаемые данные, состояние пользователя.

Следующим шагом пересмотрите получение данных из store, воспользовавшись пакетом reselect, предварительно установив его с помощью команды: npm i reselect -E.

В директории /src/components создайте новый компонент AuthorizationScreen. Разметку для компонента возьмите из шаблона result-login в файле index.html.

Обновите объект начального состояния. Добавьте поле isAuthorizationRequired со значением false. На основании значения этого поля будем принимать решение о необходимости рендеринга компонента, созданного в предыдущем шаге. Это временное решение, и мы его обязательно улучшим в следующем модуле.

Добавьте новый action и action creator, которые будут отвечать за обновление поля isAuthorizationRequired. Обновите код функции-редьюсера: добавьте код обновления поля isAuthorizationRequired.

Измените логику рендера основного компонента: если значение поля isAuthorizationRequired равно true, значит следует рендерить компонент AuthorizationScreen, созданный в первом шаге. В случае значения false выполняется рендеринг компонент с вопросами.

Напишите код для выполнения аутентификации на сервере. Для выполнения аутентификации вам требуется выполнить POST запрос к ресурсу /login. В теле запроса следует передать объект с двумя полями: email и password, которые пользователь вводит в компоненте AuthorizationScreen. Убедитесь, что пользователь заполнил оба поля.

В случае успешной аутентификации сервер вернёт код 200, запишет в куки авторизационный токен, а в теле запроса вернёт объект вида:

{

id: 1,

email: "Oliver.conner@gmail.com",

name: "Oliver.conner"

}

Сохраните полученную информацию в store.

Обратите внимание. Сервер может вернуть код 400 (Bad request), который означает, что были переданы не все данные (например, только email). Более подробную информацию об ошибке вы можете узнать из ответа сервера.

Обратите внимание. После авторизации в случае долгого простоя или по иным причинам сервер может разлогинить пользователя, из-за чего на дальнейшие запросы сервер может возвращать код 401 (Unauthorized). Это нормально. Мы обработаем эту ситуацию в следующем модуле.

В приложении отсутствует регистрация, поэтому для тестирования вы можете вводить любые комбинации email/пароль.

Не забудьте написать все необходимые тесты для новой функциональности.

***Личный проект:***

Пришло время подключить проект к боевому серверу, избавиться от мок и попрактиковаться в работе с новыми инструментами: axios, recompose и redux-thunk. В этом задании мы сконфигурируем проект для взаимодействия с удалённым сервером, а затем загрузим информацию о списке фильмов и предложениях по аренде.

***Что посмотреть:***

Задача

Установите пакеты axios, recompose и redux-thunk с помощью команды: npm i axios recompose redux-thunk -E.

Отключите и удалите файл с моками списка фильмов.

Создайте новый модуль api.js. Подключите в нём axios и опишите функцию, принимающую dispatch в качестве параметра и возвращающую сконфигурированный инстанс axios:

baseURL — <https://htmlacademy-react-2.appspot.com/wtw>,

timeout — 5000,

withCredentials — true.

Откройте файл src/index.js. Подключите в нём thunk из одноимённого пакета и compose из recompose. Также нам потребуется импортировать applyMiddleware из пакета redux и созданную в предыдущем шаге функцию для получения сконфигурированного экземпляра axios.

В этом же файле (src/index.js) создайте новую переменную и сохраните в неё экземпляр настроенного axios. Например:

// configreAPI - функция из модуля api.js,

// созданная в третьем шаге.

const api = configureAPI((...args) => store.dispatch(...args));

Обновите код инициализации хранилища (мы по-прежнему в файле src/index.js). Добавьте подключение thunk:

const store = createStore(

reducer,

compose(

applyMiddleware(thunk.withExtraArgument(api)),

window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ ? window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_() : (f) => f

)

);

Обратите внимание: чтобы не передавать в createStore пустую функцию, если React DevTools не установлены, можно воспользоваться продвинутым способом настройки стора из документации.

Напишите код для загрузки информации о списке фильмов с сервера. Для получения списка фильмов с сервера, выполните GET запрос к ресурсу /films. Загруженный список фильмов должен попадать в store. Создайте для этого соответствующий action и action creator. Список фильмов должен загружаться сразу при старте приложения.

Не забудьте протестировать новую функциональность. Чтобы замокать axios установите дополнительный пакет axios-mock-adapter с помощью команды npm i axios-mock-adapter -DE.

Немного рефакторинга

Пришло время немного порефакторить код. Для начала разбейте редьюсер, разделив зоны ответственности: загружаемые данные, состояние пользователя.

Следующим шагом пересмотрите фильтрацию списка фильмов по жанрам и перепишите с использованием пакета reselect, предварительно установив его.

***Шесть городов:***

Задача

Установите пакеты axios, recompose и redux-thunk с помощью команды: npm i axios recompose redux-thunk -E.

Отключите и удалите файл с моками списка предложений.

Создайте новый модуль api.js. Подключите в нём axios и опишите функцию, принимающую dispatch в качестве параметра и возвращающую сконфигурированный axios:

baseURL — <https://htmlacademy-react-2.appspot.com/six-cities>,

timeout — 5000,

withCredentials — true

Откройте файл src/index.js. Подключите в нём thunk из одноимённого пакета и compose из recompose. Также нам потребуется импортировать applyMiddleware из пакета redux и созданную в предыдущем шаге функцию для получения сконфигурированного экземпляра axios.

В этом же файле (src/index.js) создайте новую константу и сохраните в неё экземпляр настроенного axios. Например:

// configreAPI - функция из модуля api.js,

// созданная в третьем шаге.

const api = configureAPI((...args) => store.dispatch(...args));

Обновите код инициализации хранилища (мы по-прежнему в файле src/index.js). Добавьте подключение thunk:

const store = createStore(

reducer,

compose(

applyMiddleware(thunk.withExtraArgument(api)),

window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_ ? window.\_\_REDUX\_DEVTOOLS\_EXTENSION\_\_() : (f) => f

)

);

Обратите внимание: чтобы не передавать в createStore пустую функцию, если React DevTools не установлены, можно воспользоваться продвинутым способом настройки стора из документации.

Напишите код для загрузки информации о списке предложений с сервера. Для получения списка предложений с сервера, выполните GET запрос к ресурсу /hotels. Загруженный список предложений должен попадать в store. Создайте для этого соответствующий action и action creator. Список предложений должен загружаться сразу при старте приложения.

Не забудьте протестировать новую функциональность. Чтобы замокать axios установите дополнительный пакет axios-mock-adapter с помощью команды npm i axios-mock-adapter -DE.

Немного рефакторинга

Пришло время немного порефакторить код. Для начала разбейте редьюсер, разделив зоны ответственности: загружаемые данные, состояние пользователя.

Следующим шагом пересмотрите фильтрацию списка предложений по городам и перепишите с использованием пакета reselect, предварительно установив его с помощью команды: npm i reselect -E.

**Личный проект:**

Продолжаем взаимодействовать с сервером и на этом раз решим задачу аутентификации. В каждом проекте предусмотрена страница для входа в приложение с помощью комбинации из логина и пароля. В этом задании мы запрограммируем новый компонент и на практике разбёремся с процессом аутентификации.

***Что посмотреть:***

Задача

В директории /src/components создайте новый компонент «Sign In». Разметку для компонента возьмите из файла sign-in.html.

Обновите объект начального состояния. Добавьте поле isAuthorizationRequired со значением false. На основании значения этого поля будем принимать решение о необходимости рендеринга компонента, созданного в предыдущем шаге. Это временное решение, и мы его обязательно улучшим в следующем модуле.

Добавьте новый action и action creator, которые будут отвечать за обновление поля isAuthorizationRequired. Обновите код функции-редьюсера: добавьте код обновления поля isAuthorizationRequired.

Измените логику рендера компонента «Главная страница»: если значение поля isAuthorizationRequired равно true, значит вместо главной страницы следует рендерить компонент «Sign In», созданный в первом шаге. В случае значения false выполняется рендеринг компонента «Главная страница».

Напишите код для выполнения аутентификации на сервере. Для выполнения аутентификации вам требуется выполнить POST запрос к ресурсу /login. В теле запроса следует передать объект с двумя полями: email и password, которые пользователь вводит в компоненте «Sign In». Убедитесь, что пользователь заполнил оба поля.

В случае успешной аутентификации сервер вернёт код 200, запишет в куки авторизационный токен, а в теле запроса вернёт объект вида:

{

id: 1,

email: "Oliver.conner@gmail.com",

name: "Oliver.conner",

avatar\_url: "img/1.png"

}

Сохраните полученную информацию в store.

Обратите внимание. Сервер может вернуть код 400 (Bad request), который означает, что были переданы не все данные (например, только email). Более подробную информацию об ошибке вы можете узнать из ответа сервера.

Обратите внимание. После авторизации в случае долгого простоя или по иным причинам сервер может разлогинить пользователя, из-за чего на дальнейшие запросы сервер может возвращать код 401 (Unauthorized). Это нормально. Просто обновите страницу. Мы обработаем эту ситуацию в следующем модуле.

В приложении отсутствует регистрация, поэтому для тестирования вы можете вводить любые комбинации email/пароль.

Доработайте компонент формирования главной страницы. Для гостей в правом углу шапки должна отображаться ссылка «Sign In». С ссылкой ничего пока делать не нужно. Для зарегистрированных пользователей (прошедших авторизацию) вместо ссылки «Sign In» отображается аватарка зарегистрированного пользователя.

Не забудьте написать все необходимые тесты для новой функциональности.

***Шесть городов:***

Задача

В директории /src/components создайте новый компонент «Sign In». Разметку для компонента возьмите из файла login.html.

Обновите объект начального состояния. Добавьте поле isAuthorizationRequired со значением false. На основании значения этого поля будем принимать решение о необходимости рендеринга компонента, созданного в предыдущем шаге. Это временное решение, и мы его обязательно улучшим в следующем модуле.

Добавьте новый action и action creator, которые будут отвечать за обновление поля isAuthorizationRequired. Обновите код функции-редьюсера: добавьте код обновления поля isAuthorizationRequired.

Измените логику рендера компонента «Главная страница»: если значение поля isAuthorizationRequired равно true, значит вместо главной страницы следует рендерить компонент «Sign In», созданный в первом шаге. В случае значения false выполняется рендеринг компонента «Главная страница».

Напишите код для выполнения аутентификации на сервере. Для выполнения аутентификации вам требуется выполнить POST запрос к ресурсу /login. В теле запроса следует передать объект с двумя полями: email и password, которые пользователь вводит в компоненте «Sign In». Убедитесь, что пользователь заполнил оба поля.

В случае успешной аутентификации сервер вернёт код 200, запишет в куки авторизационный токен, а в теле запроса вернёт объект вида:

{

id: 1,

email: "Oliver.conner@gmail.com",

name: "Oliver.conner",

avatar\_url: "img/1.png",

is\_pro: false

}

Сохраните полученную информацию в store.

Обратите внимание. Сервер может вернуть код 400 (Bad request), который означает, что были переданы не все данные (например, только email). Более подробную информацию об ошибке вы можете узнать из ответа сервера.

Обратите внимание. После авторизации в случае долгого простоя или по иным причинам сервер может разлогинить пользователя, из-за чего на дальнейшие запросы сервер может возвращать код 401 (Unauthorized). Это нормально. Мы обработаем эту ситуацию в следующем модуле.

В приложении отсутствует регистрация, поэтому для тестирования вы можете вводить любые комбинации email/пароль.

Доработайте компонент формирования главной страницы. Для гостей в правом углу шапки должна отображаться ссылка «Sign In». С ссылкой ничего пока делать не нужно. Для зарегистрированных пользователей (прошедших авторизацию) вместо ссылки «Sign In» отображается электронная почта зарегистрированного пользователя.

Не забудьте написать все необходимые тесты для новой функциональности.

**Личный проект:**

На этот раз мы займёмся комментариями. Авторизованные пользователи могут выразить своё мнение с помощью отправки комментариев. Гости приложения не должны иметь такую возможность — они только могут их прочитать. В этом задании мы реализуем функциональность для отправки комментариев.

Помните, если вы делаете это задание следом за первым заданием модуля 7, у вас по-прежнему нет библиотеки react-router и для реализаци этого задания нужно воспользоваться самописным роутером из модуля 3. Помните, что «наш» роутер не поддерживает многоуровневые маршруты.

***Что посмореть:***

Задача

Создайте новый компонент «Страница добавления нового комментария». Все необходимые данные компонент принимает через props. Разметку для компонента вы найдёте в файле add-review.html. Напишите всю необходимую логику для отправки комментария на сервер. Информация о взаимодействии с сервером приведена в техническом задании. Для взаимодействия с сервером используйте пакет axios.

Создайте для react-router новый маршрут /films/:id/review. По данному маршрута должна открываться страница добавления нового комментария или проще говоря рендерится компонент, созданный в первом шаге.

Измените логику рендера компонента «Страница добавления нового комментария»: если значение поля isAuthorizationRequired равно true, значит вместо страницы добавления комментария следует рендерить компонент «Sign In», созданный в предыдущем задании. В случае значения false выполняется рендеринг компонента «Страница добавления нового комментария».

Обратите внимание. После авторизации в случае долгого простоя или по иным причинам сервер может разлогинить пользователя, из-за чего на дальнейшие запросы сервер может возвращать код 401 (Unauthorized). Это нормально. Мы обработаем эту ситуацию в следующем модуле.

Откройте компонент «Детальная информация о фильме» и доработайте отображение кнопки «Add review». Кнопка должна отображаться только для авторизованных пользователей. Используйте тот же флаг isAuthorizationRequired.

Не забудьте написать все необходимые тесты.

***Шесть городов:***

Задача

Создайте новый компонент «Форма отправки комментария». Разметку для компонента вы найдёте в файле property.html. Напишите всю необходимую логику для отправки комментария на сервер. Информация о взаимодействии с сервером приведена в техническом задании. Для взаимодействия с сервером используйте пакет axios.

Подключите компонент, созданный в первом шаге, к компоненту «Детальная информация о предложении», который вы создали в 3 модуле. Компонент с формой отправки комментария должен отображаться только для авторизованных пользователей. Форма отправки комментария отображается после списка комментариев (см. property.html).

Измените логику рендера компонента «Детальная информация о предложении»: если значение поля isAuthorizationRequired равно true, значит вместо страницы с информацией о предложении следует рендерить компонент «Sign In», созданный в предыдущем задании. В случае значения false выполняется рендеринг компонента «Детальная информация о предложении».

Обратите внимание. После авторизации в случае долгого простоя или по иным причинам сервер может разлогинить пользователя, из-за чего на дальнейшие запросы сервер может возвращать код 401 (Unauthorized). Это нормально. Мы обработаем эту ситуацию в следующем модуле.

Не забудьте написать все необходимые тесты.