26. Учебный проект: пришёл, увидел, загрузил (часть 1)

Рабочая ветка **module8-task1**

Задание

В этом задании мы начнём работать с сервером. Интерфейс API и структуры данных, используемые на сервере, теперь можно посмотреть в ТЗ.

Загрузка данных

В этом разделе мы синхронизируем локальные данные с сервером через REST API. Сервер, который мы подготовили для вас, работает по адресу: `https://11.ecmascript.pages.academy/task-manager`.

1. Добавьте в проект модуль, который будет отправлять на сервер REST запросы, и спроектируйте его интерфейс (публичные методы).

2. Создайте отдельную модель, в которой примените паттерн `Адаптер` для преобразования полученных данных с сервера: при скачивании вы получаете данные в формате сервера, и вам нужно преобразовать их в тот формат, который вы разработали в разделе о структурах данных.

Например, если где-то в своих данных вы используете `Set`, вам нужно создать его самостоятельно. Потому что данные по сети передаются в формате `JSON`, в котором нет множеств, а могут быть только массивы.

3. Удалите из проекта моковые данные и функции для работы с ними.

4. Создайте в точке входа экземпляр API.

Обратите внимание, что все запросы, которые вы будете делать, должны содержать заголовок `Authorization` со значением `Basic случайная_строка`, например `Basic eo0w590ik29889a` (пожалуйста, не используйте этот пример в проекте).

5. С помощью созданного модуля сделайте `GET`-запрос на адрес `/tasks`, чтобы получить список всех задач с сервера. Измените отрисовку списка задач так, чтобы она происходила только после получения данных. На время загрузки вместо списка выведите сообщение «Loading…» Сообщение должно показываться единожды на старте приложения, пока данные загружаются. В случае, если при загрузке произошла ошибка, приложение должно отработать так, как будто данных нет, и показать соответствующую заглушку.

6. Убедитесь, что страница отрисовывается корректно, но уже по данным с сервера. Создание, обновление и удаление данных пока не работает. Это нормально! С обновлением мы разберёмся далее, а с созданием и удалением — во второй части домашнего задания.

Обновление данных

Пришло время научиться синхронизировать обновлённые данные с сервером. Основная задача в том, чтобы полученные изменения от пользователя сначала отправить на сервер, получить от него одобрение и, в случае успеха, отобразить изменения в интерфейсе.

1. Опишите в API метод для обновления задачи на сервере. Для этого нужно выполнить `PUT`-запрос на адрес `/tasks/:taskId`, где `:taskId` — id конкретной задачи, передав в теле запроса обновлённые данные.

2. Замените в компонентах прямую работу с данными на работу с моделью, которую мы создали ранее.

3. Перепишите в контроллере код, который отвечает за обновление данных, чтобы он обновлял не моки, а данные на сервере с помощью модели.

Обратите внимание, перед отправкой запроса на обновление нужно преобразовать данные из внутреннего формата

приложения в формат сервера. Потребуется написать ещё один адаптер, теперь уже в обратную сторону — для преобразования данных в формат сервера.

4. В случае удачного запроса, сервер вернёт вам обновлённые данные. Передайте их контроллеру на отрисовку и обновите список задач. Форма редактирования при этом должна скрыться.

Обратите внимание, что изменения в DOM должны происходить только после успешного запроса к серверу, иначе мы получим несогласованность — ситуацию, когда интерфейс не отражает реальных данных.

Обработку неудачных запросов и обратную связь в форме создания и редактирования задачи мы реализуем во второй части задания.

28. Личный проект: пришёл, увидел, загрузил (часть 1)

Задание

В этом задании мы начнём работать с сервером. Интерфейс API и структуры данных, используемые на сервере, теперь можно посмотреть в ТЗ.

Киноман

Загрузка данных

В этом разделе мы синхронизируем локальные данные с сервером через REST API. Сервер, который мы подготовили для вас, работает по адресу: `https://11.ecmascript.pages.academy/cinemaddict`.

1. Добавьте в проект модуль, который будет отправлять на сервер REST запросы, и спроектируйте его интерфейс (публичные методы).

2. Создайте отдельную модель, в которой примените паттерн `Адаптер` для преобразования полученных данных с сервера: при скачивании вы получаете данные в формате сервера, и вам нужно преобразовать их в тот формат, который вы разработали в разделе о структурах данных.

Например, если где-то в своих данных вы используете `Set`, вам нужно создать его самостоятельно. Потому что данные по сети передаются в формате `JSON`, в котором нет множеств, а могут быть только массивы.

3. Удалите из проекта моковые данные и функции для работы с ними.

4. Создайте в точке входа экземпляр API.

Обратите внимание, что все запросы, которые вы будете делать, должны содержать заголовок `Authorization` со значением `Basic случайная_строка`, например `Basic eo0w590ik29889a` (пожалуйста, не используйте этот пример в проекте).

5. С помощью созданного модуля сделайте `GET`-запрос на адрес `/movies`, чтобы получить список всех фильмов с сервера. Измените отрисовку списка фильмов так, чтобы она происходила только после получения данных. На время загрузки вместо списка выведите сообщение «Loading…» Сообщение должно показываться единожды на старте приложения, пока данные загружаются. В случае, если при загрузке произошла ошибка, приложение должно отработать так, как будто данных нет, и показать соответствующую заглушку.

Обратите внимание, что комментарии — это отдельная сущность на сервере, а модели фильмов хранят только их `id`. Принцип работы с комментариями описан в следующих шагах.

6. Для загрузки комментариев для конкретного фильма сделайте `GET`-запрос на адрес `/comments/:movieId`, где `:movieId` — id конкретного фильма. Комментарии можно загружать сразу при получении данных о всех фильмах или загружать комментарии только к конкретному фильму при открытии попапа с подробной информацией.

Обратите внимание, вы сами решаете, что делать, если не удалось загрузить только дополнительную информацию — комментарии. Либо обязательно так же показывать заглушку и

не давать работать с приложением, либо придумать свой вариант обработки ошибки.

7. Убедитесь, что страница отрисовывается корректно, но уже по данным с сервера. Создание, обновление и удаление данных пока не работает. Это нормально! С обновлением мы разберёмся далее, а с созданием и удалением — во второй части домашнего задания.

Обновление данных

Пришло время научиться синхронизировать обновлённые данные с сервером. Основная задача в том, чтобы полученные изменения от пользователя сначала отправить на сервер, получить от него одобрение и, в случае успеха, отобразить изменения в интерфейсе.

1. Опишите в API метод для обновления фильма на сервере. Для этого нужно выполнить `PUT`-запрос на адрес `/movies/:movieId`, где `:movieId` — id конкретного фильма, передав в теле запроса обновлённые данные.

2. Замените в компонентах прямую работу с данными на работу с моделью, которую мы создали ранее.

3. Перепишите в контроллере код, который отвечает за обновление данных, чтобы он обновлял не моки, а данные на сервере с помощью модели.

Обратите внимание, перед отправкой запроса на обновление нужно преобразовать данные из внутреннего формата приложения в формат сервера. Потребуется написать ещё один адаптер, теперь уже в обратную сторону — для преобразования данных в формат сервера.

4. В случае удачного запроса, сервер вернёт вам обновлённые данные. Передайте их контроллеру на отрисовку и обновите список фильмов.

Обратите внимание, что изменения в DOM должны происходить только после успешного запроса к серверу, иначе мы получим несогласованность — ситуацию, когда интерфейс не отражает реальных данных.

Обработку неудачных запросов и обратную связь при добавлении и удалении комментария мы реализуем во второй части задания.

Bigtrip

Загрузка данных

В этом разделе мы синхронизируем локальные данные с сервером через REST API. Сервер, который мы подготовили для вас, работает по адресу: `https://11.ecmascript.pages.academy/big-trip`.

1. Добавьте в проект модуль, который будет отправлять на сервер REST запросы, и спроектируйте его интерфейс (публичные методы).

2. Создайте отдельную модель, в которой примените паттерн `Адаптер` для преобразования полученных данных с сервера: при скачивании вы получаете данные в формате сервера, и вам нужно преобразовать их в тот формат, который вы разработали в разделе о структурах данных.

Например, если где-то в своих данных вы используете `Set`, вам нужно создать его самостоятельно. Потому что данные по сети передаются в формате `JSON`, в котором нет множеств, а могут быть только массивы.

3. Удалите из проекта моковые данные и функции для работы с ними.

4. Создайте в точке входа экземпляр API.

Обратите внимание, что все запросы, которые вы будете делать, должны содержать заголовок `Authorization` со значением `Basic случайная_строка`, например `Basic eo0w590ik29889a` (пожалуйста, не используйте этот пример в проекте).

5. С помощью созданного модуля сделайте `GET`-запрос на адрес `/points`, чтобы получить список всех точек маршрута с сервера. Измените отрисовку списка точек маршрута так, чтобы она происходила только после получения данных. На время загрузки вместо списка выведите сообщение «Loading…» Сообщение должно показываться единожды на старте приложения, пока данные загружаются. В случае, если при загрузке произошла ошибка, приложение должно отработать так, как будто данных нет, и показать соответствующую заглушку.

Обратите внимание, что пункты назначения и дополнительные опции — это отдельные сущности на сервере. Принцип работы с ними описан в следующих шагах.

6. Сделайте `GET`-запрос на адрес `/destinations`, чтобы получить список всех возможных пунктов назначения. Запишите полученные данные в отдельную структуру и используйте их вместо моковых данных.

7. Сделайте `GET`-запрос на адрес `/offers` чтобы получить дополнительные опции по всем возможным типам и также сохраните в отдельную структуру и используйте их вместо

моковых данных.

Обратите внимание, вы сами решаете, что делать, если не удалось загрузить только дополнительную информацию: по пунктам назначения и/или дополнительным опциям. Либо обязательно так же показывать заглушку и не давать работать с приложением, либо придумать свой вариант обработки ошибки.

8. Убедитесь, что страница отрисовывается корректно, но уже по данным с сервера. Создание, обновление и удаление данных пока не работает. Это нормально! С обновлением мы разберёмся далее, а с созданием и удалением — во второй части домашнего задания.

Обновление данных

Пришло время научиться синхронизировать обновлённые данные с сервером. Основная задача в том, чтобы полученные изменения от пользователя сначала отправить на сервер, получить от него одобрение и, в случае успеха, отобразить изменения в интерфейсе.

1. Опишите в API метод для обновления точки маршрута на сервере. Для этого нужно выполнить `PUT`-запрос на адрес `/points/:pointId`, где `:pointId` — id конкретной точки маршрута, передав в теле запроса обновлённые данные.

2. Замените в компонентах прямую работу с данными на работу с моделью, которую мы создали ранее.

3. Перепишите в контроллере код, который отвечает за обновление данных, чтобы он обновлял не моки, а данные на сервере с помощью модели.

Обратите внимание, перед отправкой запроса на обновление нужно преобразовать данные из внутреннего формата приложения в формат сервера. Потребуется написать ещё один адаптер, теперь уже в обратную сторону — для преобразования данных в формат сервера.

4. В случае удачного запроса, сервер вернёт вам обновлённые данные. Передайте их контроллеру на отрисовку и обновите список точек маршрута. Форма редактирования при этом должна скрыться.

Обратите внимание, что изменения в DOM должны происходить только после успешного запроса к серверу, иначе мы получим несогласованность — ситуацию, когда интерфейс не отражает реальных данных.

Обработку неудачных запросов и обратную связь в форме создания и редактирования точки маршрута мы реализуем во второй части задания.