# 30. Учебный проект: жизнь без интернета

Рабочая ветка **module9-task1**

## Задание

В этом разделе нам нужно добавить возможность запуска и работы с приложением даже без интернета. А когда интернет появится, синхронизировать данные с сервером.

### ServiceWorker

Для возможности запуска приложения без интернета мы будем использовать `ServiceWorker` (сервис-воркер).

1. Создайте файл `public/sw.js`. В нём будет код сервис-воркера.

2. В `public/index.html` подключите файл сервис-воркера первым.

3. В `src/main.js` зарегистрируйте сервис-воркер.

4. В файле `public/sw.js` опишите обработчики событий сервис-воркера:

- обработчик события `install`, который поместит в кэш все статические ресурсы, кроме файла с самим сервис-воркером. Браузер кэширует его автоматически;

- обработчик события `fetch`, который будет возвращать из кэша файл, если таковой в нём имеется, чтобы не загружать файл с сервера ещё раз;

- обработчик события `activate`, который на каждую активацию сервис-воркера будет удалять с компьютера пользователя устаревшие кэши.

5. После обновите страницу для установки сервис-воркера. Следующие загрузки страницы могут пройти без интернета.

## Offline

1. Создайте директорию `src/api`, или любую другую общую папку на ваше усмотрение, и перенесите туда модуль `Api`.

2. В этой же директории создайте модуль `Provider`, который свяжет `Api` и хранилище:

- конструктор `Provider` должен принимать экземпляр `Api` и экземпляр хранилища;

- интерфейс `Provider` должен полностью копировать интерфейс `Api`. Это позволит заменить одного на другого без переписывания кодовой базы. Пока что внутри методов `Provider` вызывайте методы `Api`. Есть даже такой паттерн — Прокси.

3. В этой же директории создайте модуль `Store` (хранилище), который станет обёрткой над локальным хранилищем (`localStorage`) и будет хранить данные на случай отсутствия интернета:

    - конструктор `Store` должен принимать ключ, по которому информация будет храниться в локальном хранилище и сам `localStorage`;

    - интерфейс `Store` придумайте самостоятельно. Как минимум он должен содержать методы получения всех данных, записи/перезаписи данных и удаления данных.

**Обратите внимание,** в `localStorage` по ключу можно хранить только одну строку текста. Поэтому перед записью превратите модели в объекты, а объекты в строку JSON с помощью метода `JSON.stringify`. Для получения

информации из хранилища и превращения её в модель используйте `JSON.parse` и методы модели.

4. В `src/main.js` в дополнение к экземпляру `Api` создайте экземпляр `Store`. Передайте их как зависимости при создании экземпляра `Provider` здесь же. Предположим, что вы назвали
   их `api`, `store` и `apiWithProvider` соответственно.

5. Замените вызовы `api` во всём приложении на вызовы `apiWithProvider` и убедитесь, что всё работает так же, как и раньше.

**Обратите внимание,** речь идёт не про работу без интернета, а про обращения к серверу. После
замены `api` на `apiWithProvider` приложение должно работать как раньше.

6. Реализуйте логику `Provider`. Необходимо, чтобы результат удачного обращения к серверу дублировался в хранилище. А в случае, если интернет отсутствует, ответ брался из хранилища.

   - реализуйте приватный метод `_isOnLine` для проверки наличия сети;

   - в каждый метод `Provider`, который дублирует `Api`, добавьте проверку на наличие сети.

Если сеть есть, то обработайте обращение к серверу, результат его запишите в хранилище и передайте дальше. Если сети нет, то отразите изменения в хранилище (в случае PUT-, POST-, DELETE-запросов) и верните `Promise.resolve()` с моделью в качестве значения промиса;

- добавьте приватное свойство-флаг, которое будет хранить булево значение, означающее необходимость синхронизации;

- реализуйте публичный метод для получения этого значения (геттер).

**Обратите внимание,** для дальнейшего удобства синхронизации нужно помечать изменённые или созданные сущности вне сети. А так же нужно подумать о полях, которые генерируются только на сервере — id, рейтинг и подобные.

7. Добавьте в `src/main.js` на `window` обработчики событий `offline` и `online`:
   - когда интернет исчез, сообщите пользователю об этом, добавив в заголовок текст `[offline]`;

- когда интернет вернётся, удалите из заголовка текст `[offline]`;

8. Добавьте в `Api` метод синхронизации по аналогии с другими методами.

9. Реализуйте в `Provider` метод синхронизации. Логика следующая:

    - нужно получить данные из хранилища;

    - затем вызвать метод синхронизации `Api`;

    - в случае успешного обращения к серверу нужно заменить в хранилище данные, изменённые или созданные без сети, на те, что придут в ответе сервера. Изменить свойство-флаг о необходимости синхронизации.

10. Добавьте в обработчик события `online` вызов метода `Provider` для синхронизации с сервером. Это необходимо сделать лишь в том случае, если синхронизация требуется.

# 31. Личный проект: жизнь без интернета

Киноман/Bigtrip

## Задание

В этом разделе нам нужно добавить возможность запуска и работы с приложением даже без интернета. А когда интернет появится, синхронизировать данные с сервером.

### ServiceWorker

Для возможности запуска приложения без интернета мы будем использовать `ServiceWorker` (сервис-воркер).

1. Создайте файл `public/sw.js`. В нём будет код сервис-воркера.

2. В `public/index.html` подключите файл сервис-воркера первым.

3. В `src/main.js` зарегистрируйте сервис-воркер.

4. В файле `public/sw.js` опишите обработчики событий сервис-воркера:

    - обработчик события `install`, который поместит в кэш все статические ресурсы, кроме файла с самим

сервис-воркером. Браузер кэширует его автоматически;

- обработчик события `fetch`, который будет возвращать из кэша файл, если таковой в нём имеется, чтобы не загружать файл с сервера ещё раз;

- обработчик события `activate`, который на каждую активацию сервис-воркера будет удалять с компьютера пользователя устаревшие кэши.

5. После обновите страницу для установки сервис-воркера. Следующие загрузки страницы могут пройти без интернета.

## Offline

1. Создайте директорию `src/api`, или любую другую общую папку на ваше усмотрение, и перенесите туда модуль `Api`.

2. В этой же директории создайте модуль `Provider`, который свяжет `Api` и хранилище:

   - конструктор `Provider` должен принимать экземпляр `Api` и экземпляр хранилища;

- интерфейс `Provider` должен полностью копировать интерфейс `Api`. Это позволит заменить одного на другого без переписывания кодовой базы. Пока что внутри методов `Provider` вызывайте методы `Api`. Есть даже такой паттерн — Прокси.

3. В этой же директории создайте модуль `Store` (хранилище), который станет обёрткой над локальным хранилищем (`localStorage`) и будет хранить данные на случай отсутствия интернета:

    - конструктор `Store` должен принимать ключ, по которому информация будет храниться в локальном хранилище и сам `localStorage`;

    - интерфейс `Store` придумайте самостоятельно. Как минимум он должен содержать методы получения всех данных, записи/перезаписи данных и удаления данных.

**Обратите внимание,** в `localStorage` по ключу можно хранить только одну строку текста. Поэтому перед записью превратите модели в объекты, а объекты в строку JSON с помощью метода `JSON.stringify`. Для получения информации из хранилища и превращения её в модель используйте `JSON.parse` и методы модели.

4. В `src/main.js` в дополнение к экземпляру `Api` создайте экземпляр `Store`. Передайте их как зависимости при создании экземпляра `Provider` здесь же. Предположим, что вы назвали
   их `api`, `store` и `apiWithProvider` соответственно.

5. Замените вызовы `api` во всём приложении на вызовы `apiWithProvider` и убедитесь, что всё работает так же, как и раньше.

**Обратите внимание,** речь идёт не про работу без интернета, а про обращения к серверу. После
замены `api` на `apiWithProvider` приложение должно работать как раньше.

6. Реализуйте логику `Provider`. Необходимо, чтобы результат удачного обращения к серверу дублировался в хранилище. А в случае, если интернет отсутствует, ответ брался из хранилища.

   - реализуйте приватный метод `_isOnLine` для проверки наличия сети;

   - в каждый метод `Provider`, который дублирует `Api`, добавьте проверку на наличие сети. Если сеть есть, то обработайте обращение к серверу, результат его запишите в хранилище и передайте дальше. Если сети нет, то отразите изменения в

хранилище (в случае PUT-, POST-, DELETE-запросов) и верните `Promise.resolve()` с моделью в качестве значения промиса;

- добавьте приватное свойство-флаг, которое будет хранить булево значение, означающее необходимость синхронизации;

- реализуйте публичный метод для получения этого значения (геттер).

**Обратите внимание,** для дальнейшего удобства синхронизации нужно помечать изменённые или созданные сущности вне сети. А так же нужно подумать о полях, которые генерируются только на сервере — id, рейтинг и подобные.

7. Добавьте в `src/main.js` на `window` обработчики событий `offline` и `online`:
   - когда интернет исчез, сообщите пользователю об этом, добавив в заголовок текст `[offline]`;

   - когда интернет вернётся, удалите из заголовка текст `[offline]`;

8. Добавьте в `Api` метод синхронизации по аналогии с другими методами.

9. Реализуйте в `Provider` метод синхронизации. Логика следующая:

   - нужно получить данные из хранилища;

   - затем вызвать метод синхронизации `Api`;

   - в случае успешного обращения к серверу нужно заменить в хранилище данные, изменённые или созданные без сети, на те, что придут в ответе сервера. Изменить свойство-флаг о необходимости синхронизации.

10. Добавьте в обработчик события `online` вызов метода `Provider` для синхронизации с сервером. Это необходимо сделать лишь в том случае, если синхронизация требуется.