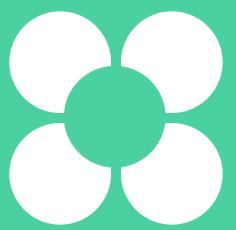
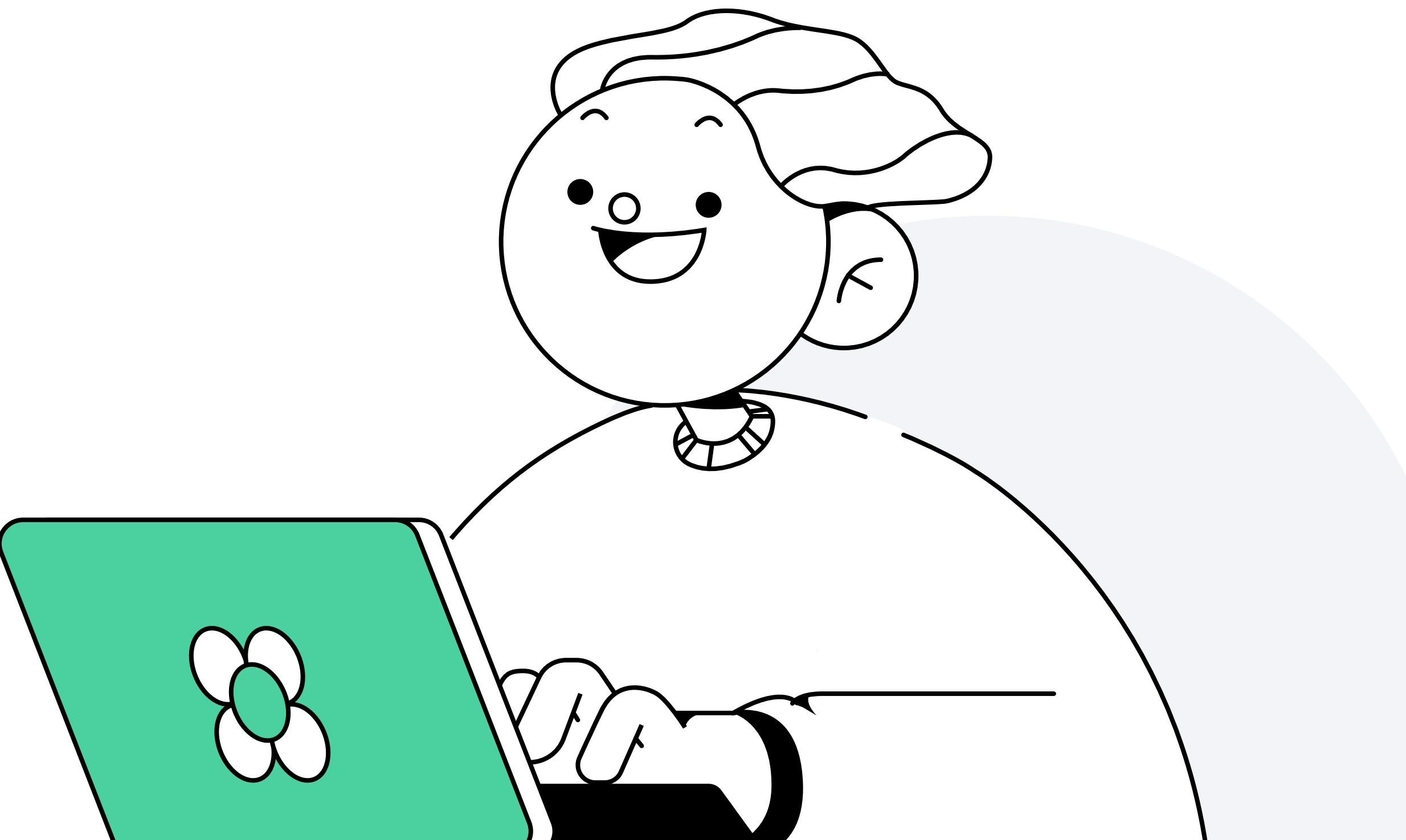


Express



План занятия

- 1 Express и Rest API
- 2 Утилиты Node.js
- 3 Тестирование API



Express & Rest API

REST

REST (Representational State Transfer – передача репрезентативного (самоописываемого) состояния) – стиль взаимодействия компонентов распределённого приложения в сети, для которого обязательны:

- модель клиент-сервер – масштабируемость и независимое развитие
- отсутствие состояния – сессия хранится на клиенте
- кеширование – допускается кешировать ответы
- унифицированный интерфейс
- слои – клиент не знает, сколько уровней взаимодействия за сервером

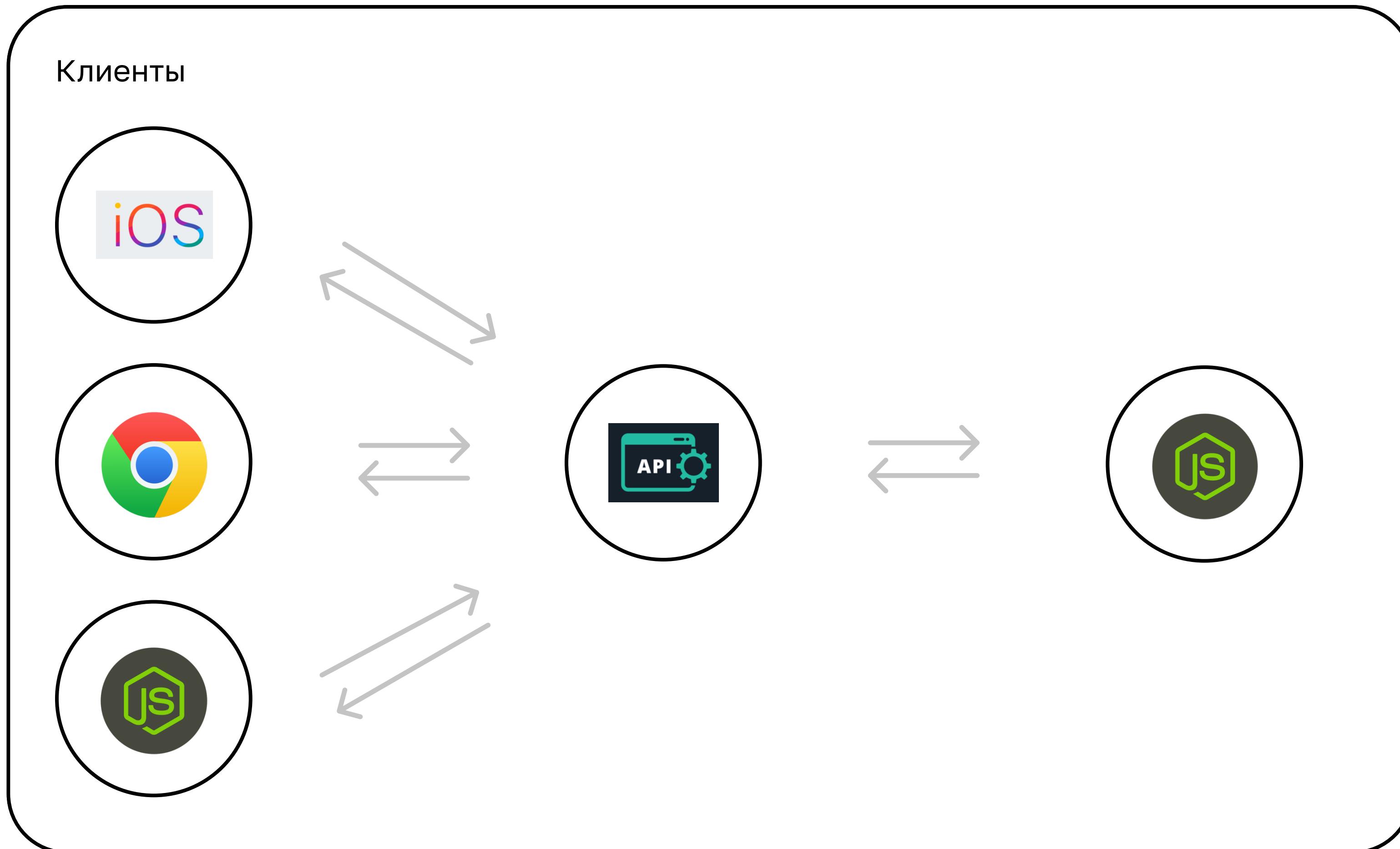
Rest API

В основе унификации интерфейса **REST API** – разделение разных операций (чаще всего CRUD) при обращении к одному и тому же URL с помощью HTTP-методов:

- **GET** – используется для получения данных
- **POST** – используется для создания новых записей
- **PUT/PATCH** – используется для обновления уже существующих записей
- **DELETE** – используется для удаления записей

Задача

Создать REST API для приложения «Список дел» – ToDo:



Реализация REST API на Node.js

Чтобы реализовать API, который работает по HTTP-протоколу и соответствует условиям REST, потребуется обработка четырёх-пяти видов запросов и соответствующих данных, передаваемых с ними.

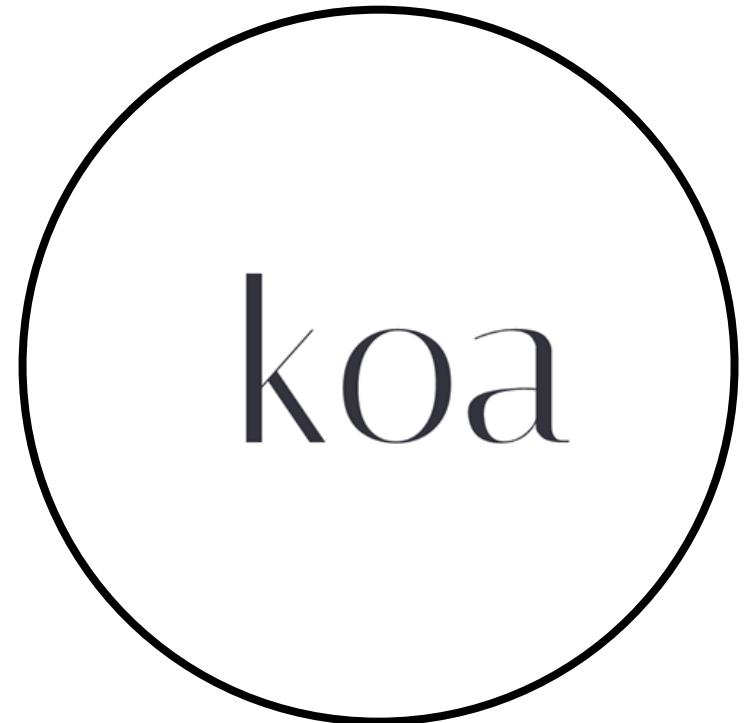
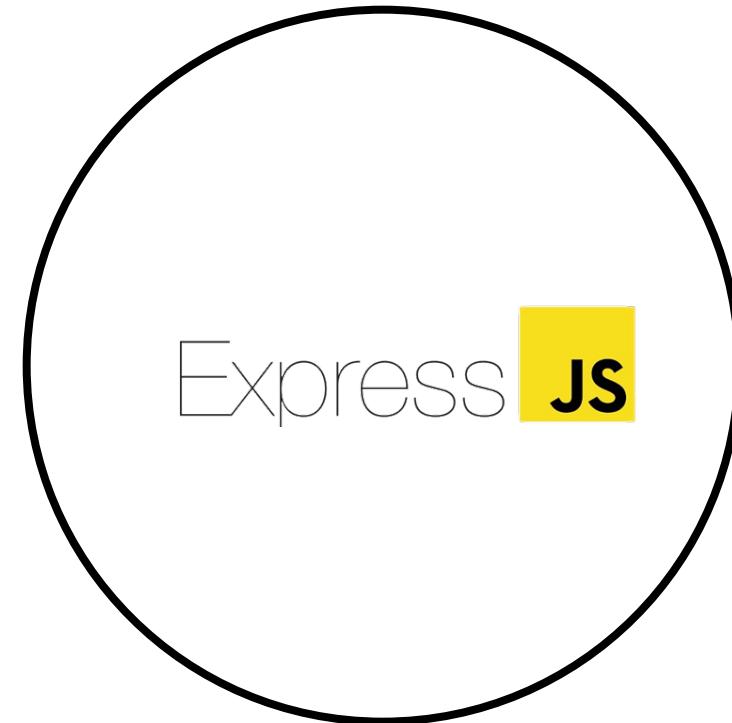
Кроме того, потребуется реализовать какую-то, пусть и небольшую, маршрутизацию.

Всё это можно сделать, используя модуль `http` Node.js, но код в таком случае получится сложным и плохо структурированным. Работать с ним будет тяжело.

Удобная альтернатива – использовать фреймворк

Фреймворки Node.js

Для **Node.js** существует ряд схожих фреймворков:



Мы будем использовать **Express**, потому что:

- простая и гибкая система маршрутизации
- интегрируется со множество шаблонизаторов
- реализует структуру промежуточного программного обеспечения

Express

Некоторые функции Express:

- позволяет настроить посредников для обработки **HTTP-запросов**
- определяет таблицу маршрутизации, которая используется для выполнения различных действий на основе метода **HTTP** и **URL-адреса**
- позволяет динамически создавать **HTML-страницы** на основе передачи аргументов шаблонам

Express: установка

Установите **Express** с помощью пакетного менеджера как основную зависимость проекта

```
npm i express
```

Express: базовая настройка

Выполним базовую настройку Express:

```
// подключение express
const express = require("express");

// создаём объект приложения
const app = express();

app.listen(3000);
```

Express

Для обработки подключений к серверу у объекта приложения express можно вызывать методы:

`.get() | .post() | .put() | .delete()`

Каждый из методов соответствует одноимённому методу протокола HTTP и принимает на вход 2 параметра: строку `url` и `Callback-функцию`

```
.get("/", (request, response) => {
    //request - хранит информацию о запросе;
    //response - управляет отправкой ответа.
});
```

Express: параметр `request`

Параметр `request` позволяет получить информацию о запросе.
Некоторые основные свойства этого объекта:

- `headers` – возвращает заголовки запроса
- `method` – тип запроса (GET, POST, DELETE, PUT)
- `url` – представляет запрошенный адрес
- `DELETE` – используется для удаления записей
- `body` – возвращает тело запроса
- `params` – возвращает параметры запроса

Express: параметр `request`

Параметр `response` управляет отправкой ответа и представляет объект, его основные методы:

- `status` – устанавливает статусный код ответа
- `setHeader` – добавляет в ответ заголовок
- `send` – пишет в поток ответа некоторое содержимое в формате `text/html`
- `json` – пишет в поток ответа некоторое содержимое в формате `application/json`
- `redirect` – совершает переадресацию на указанный URL

Базовая структура Express-приложения

```
// создаём объект приложения
const express = require('express');
const app = express();

// определяем обработчики для маршрутов
app.get('/api/todo/', (req, res) => {...});
app.get('/api/todo/:id', (req, res) => {...});

app.post('/api/todo/', (req, res) => {...});
app.put('/api/todo/:id', (req, res) => {...});
app.delete('/api/todo/:id', (req, res) => {...});

// начинаем прослушивать подключения на порту 3000
app.listen(3000);
```

Пример запроса и обработчика

HTTP-запрос:

```
PUT {{host}} /api/todo/100 create HTTP/1.1
Content-Type: application/json

{
  "title" : "todo title",
  "desc" : "todo desc"
}
```

app.js:

```
app.put('/api/todo/:id', (req, res) => {
  // получение именованного параметра из URL
  const {id} = req.params;

  // получение параметров из тела запроса
  const {title, desc} = req.body;
});
```

Утилиты Node.js

Nodemon

Nodemon – вспомогательный пакет для разработки, он следит за файлами проекта.

Если зайти в консоль и запустить сервер, Nodemon будет следить за изменениями. Как только мы сохраним файл, пакет автоматически перезагрузит наш сервер.

Это очень удобно, так как изменения будут применены автоматически

Nodemon: настройка пакета

Установите пакет `nodemon` из `npm` в качестве зависимостей разработчика, так как нам это не нужно в продакшне

```
npm i -D nodemon
```

Nodemon: настройка пакета

Чтобы запустить приложение с использованием пакета nodemon, необходимо изменить сценарии в `package.json`

```
"scripts": {  
  "dev": "nodemon index.js"  
},
```

PM2

- PM2 – это диспетчер процессов рабочей среды для приложений Node.js, в который встроен инструмент распределения нагрузки
- PM2 позволяет поддерживать приложения в рабочем состоянии, перезагружать их без простоев и обеспечивает выполнение общих задач системного администрирования
- PM2 помогает управлять ведением протоколов, мониторингом и кластеризацией приложений

Настройка пакета PM2

- 1 Установите пакет **PM2** из **npm** в качестве зависимостей проекта

```
npm i pm2
```

- 2 Обновите **package.json**

```
"scripts": {  
  "start": "pm2 start index.js"  
},
```

Основные команды РМ2

- **list** – список всех запущенных процессов
- **stop 0** – остановка приложения
- **restart 0** – перезапуск приложения
- **show 0** – просмотр подробной информации о приложении
- **delete 0** – удаление приложения из реестра рм2

Тестирование API

Postman

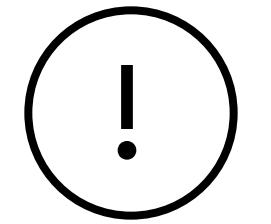
→ Postman – кроссплатформенная среда для тестирования API,
мощный набор инструментов для тестирования API

Postman

- Postman – кроссплатформенная среда для тестирования API, мощный набор инструментов для тестирования API
- ! Postman позволяет создавать, тестировать, контролировать и публиковать документацию для API



Postman



Desktop Agent доступен для всех популярных платформ

macOS

Linux

Windows

