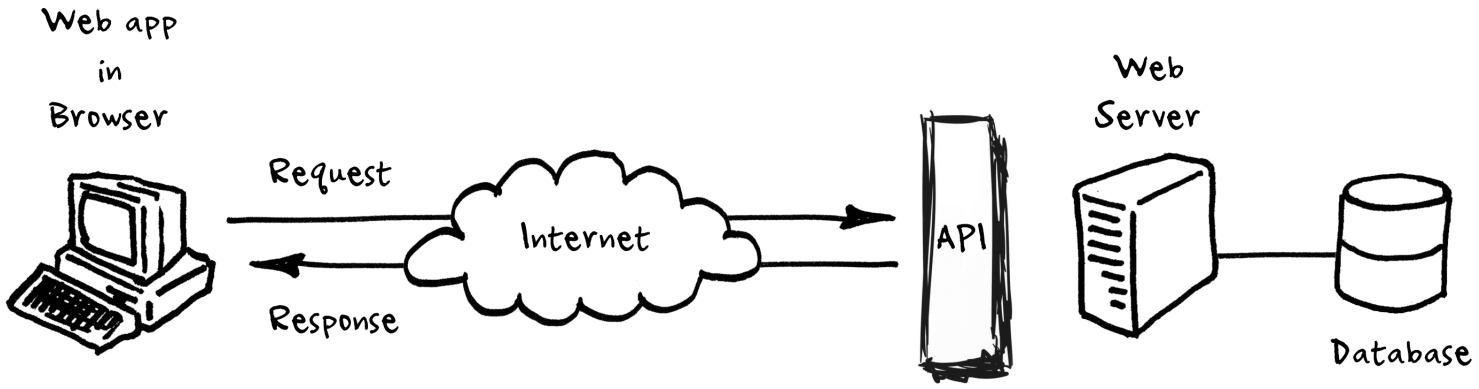
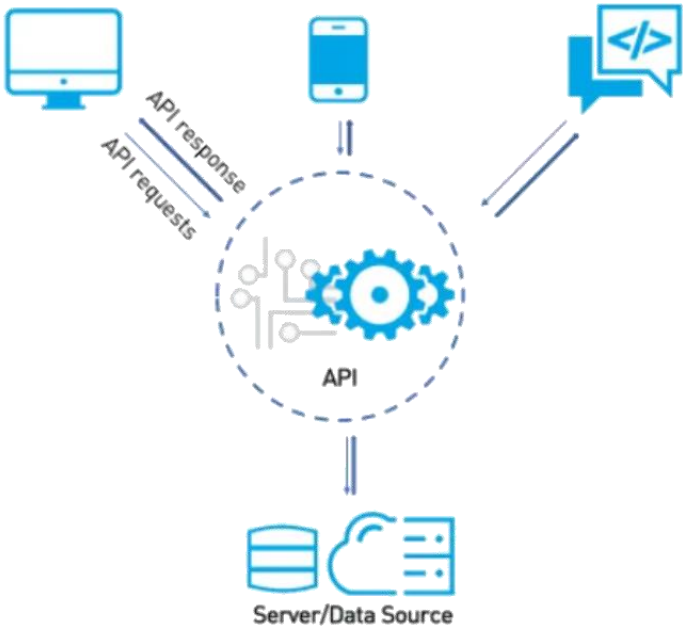
Web API using JavaScript



Outline



1. Web and HTTP
2. Web API
3. Web API using express
4. Web API using Next.js

Course Roadmap

Web Client

Request

Response

Web Server

Frontend development

HTML for page content & structure

CSS for styling

JavaScript for interaction

Backend development

We are HERE

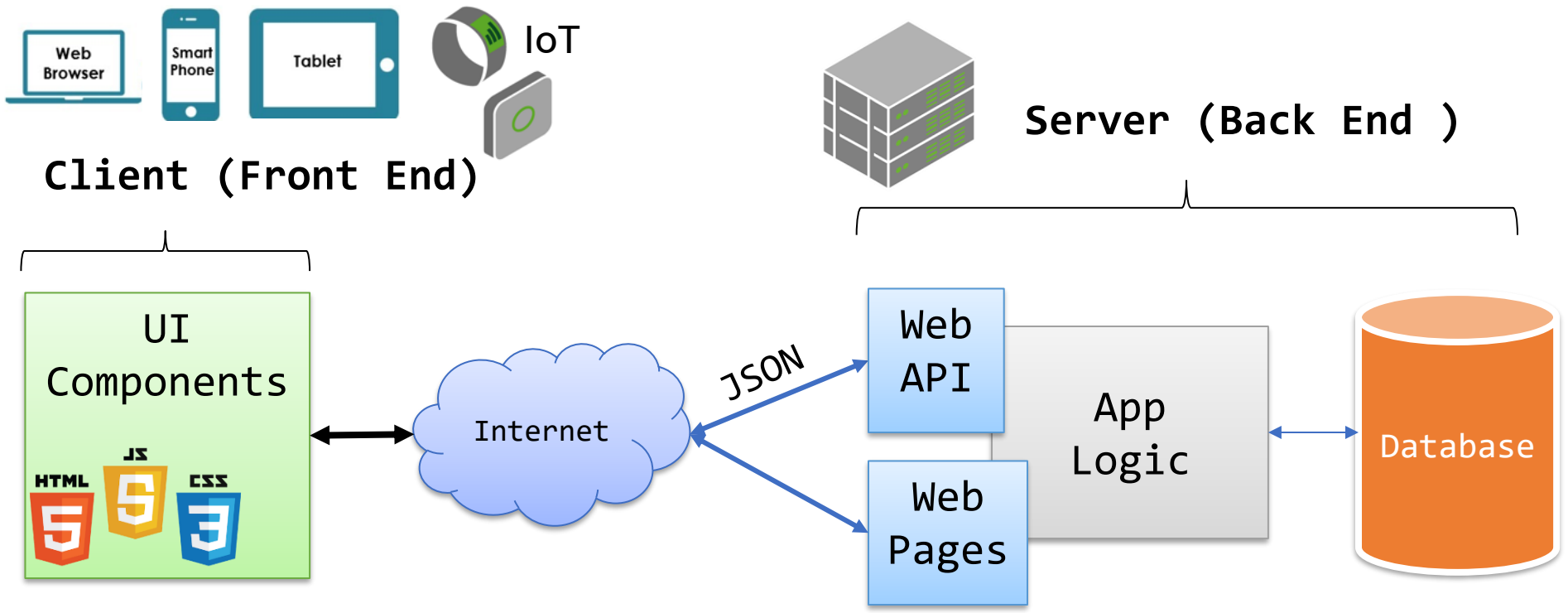
Web API

Web Pages

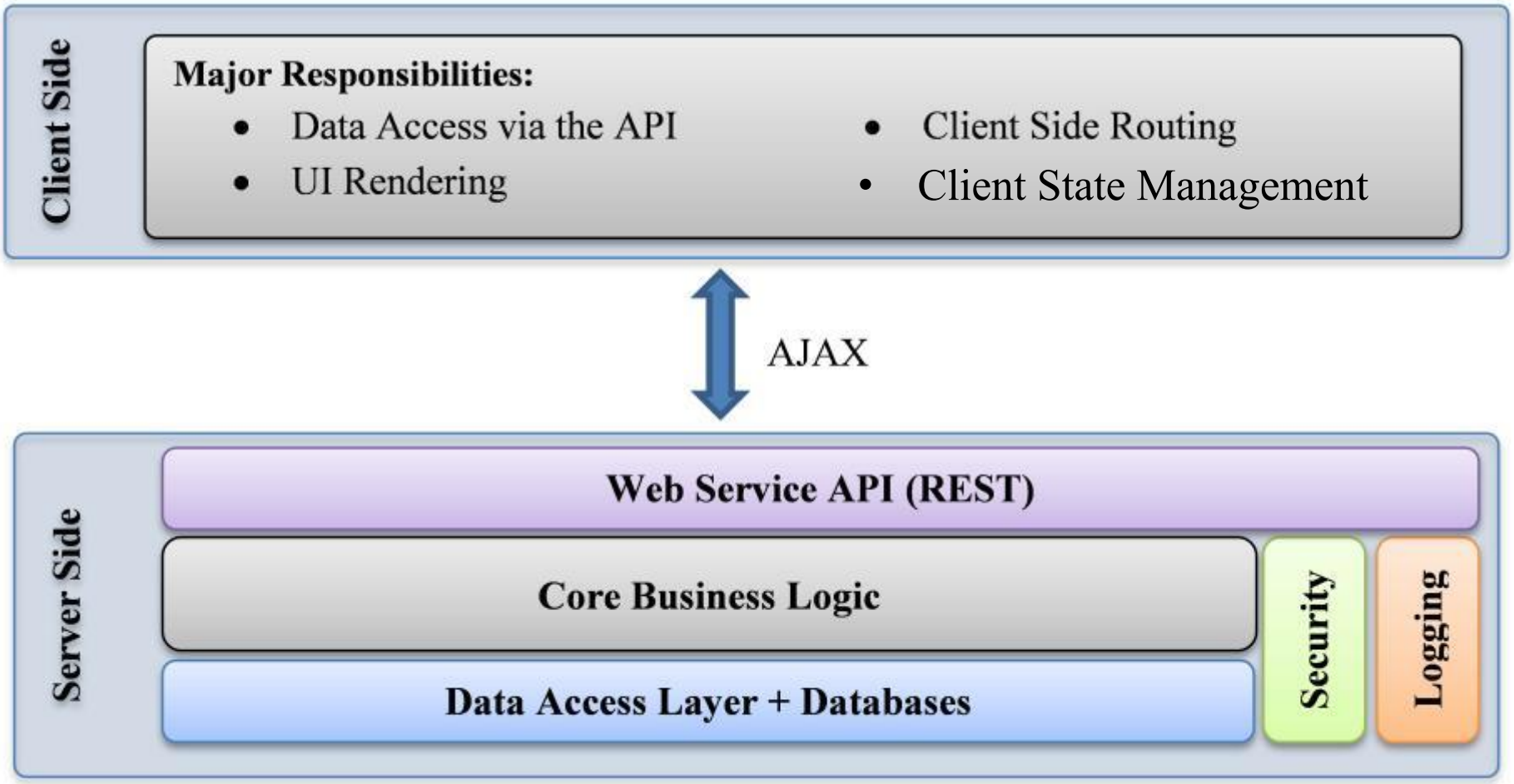
Data Management

Web App Architecture

- Front-end made-up of **multiple UI components loaded** in response to user actions
- Back-end Web API and Web pages



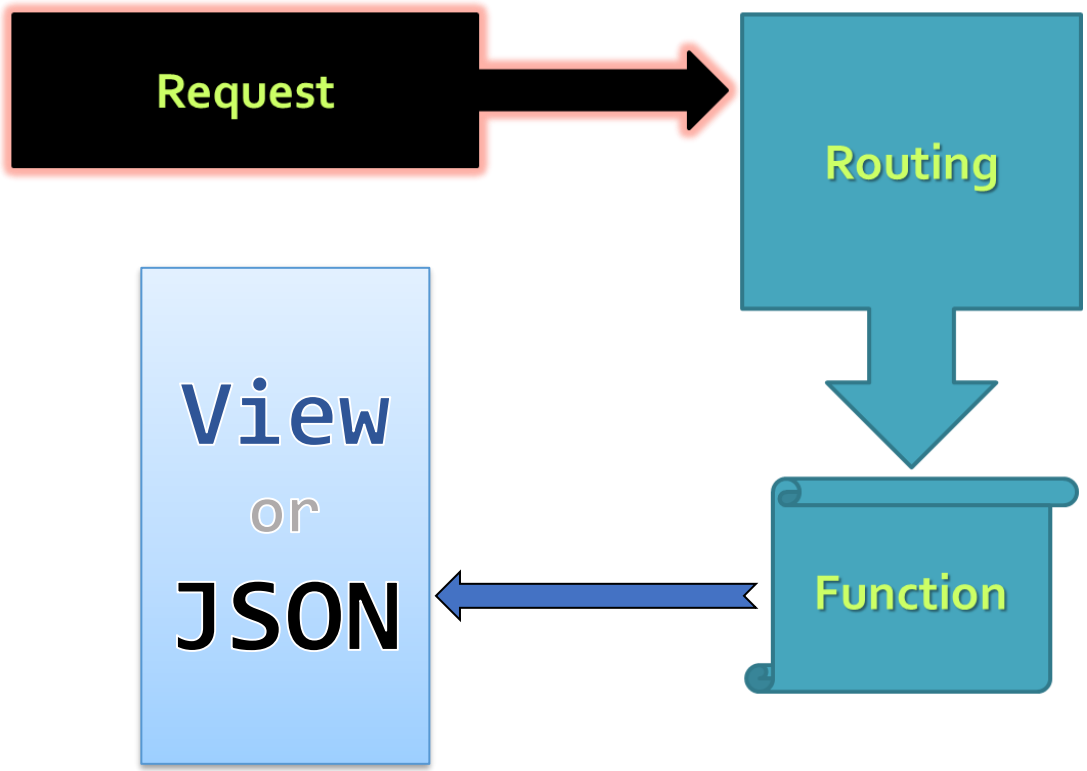
Role of Client and Server in SPA



What is a Web API

- Web API: A set of methods exposed over the web via HTTP to allow **programmatic access to applications**
- Web API are designed for **broad reach**:
  - Can be accessed by a broad range of clients including browsers and mobile devices
  - Can be implemented or consumed in any language
- Uses HTTP as an **application protocol**

Web API using Node.js Express



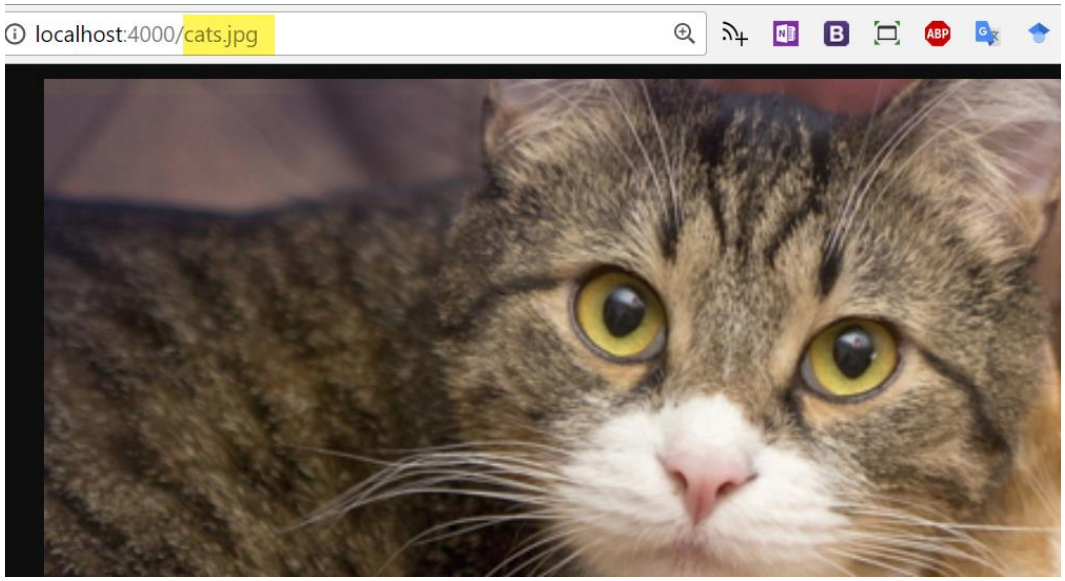
Create and Start an Express App

```
import express from 'express’;
const app = express();

app.get('/', (req, res) => {
    res.send('السلام عليكم ورحمة الله وبركاته’);
});

const port = 3000;
app.listen(port, () => {
    console.log(`App is available @ http://localhost:${port}`)
});
```

- The app listens for incoming request @ http://localhost:3000/
- When someone visits this Url the function associated with **get** **'/'** will run and **'السلام عليكم ورحمة الله وبركاته'** will be returned to the requester

Serving Static Resources using Express

- To serve up static files, the **express.static** middleware function is used with the folder path of the files to be served



Routing

- Requests can be routed based on:
  - **HTTP Verb** – GET, POST, PUT, DELETE
  - **URL Path** – e.g., /users
- App Route **maps** an **HTTP Verb** (e.g., GET or POST) + a **URI Path** (like /users/123) to a **route handler function**
  - The handler function is passed a **req** and a **res** objects
  - The **req** object represents the HTTP request and has the query string, parameters, body and HTTP headers
  - The **res** object represents the HTTP response and it is used to send the generated response

10

Path Parameters

- Named **path parameters** can be added to the URL path. E.g., **/students/:id**
- **req.params** is an object containing properties mapped to the named path parameters
  - E.g., if you have the path **/students/:id**, then the "id" property is available as **req.params.id**

```
app.get('/api/students/:id', (req, res) => {
    const studentId = req.params.id;
    console.log('req.params.id', studentId)
})
```

```
app.get('/authors/:authorId/books/:bookId', (req, res) => {
    // If the Request URL was http://localhost:3000/authors/34/books/8989
    // Then req.params: { authorId: "34", bookId: "8989" }
    res.send(req.params);
})
```

Query Parameters

- Named **query parameters** can be added to the URL path after a ? E.g., **/posts ?sortBy=createdOnDate**
- Query parameters are often used for **optional** parameters (e.g., optionally specifying the property to be used to sort of results)
- **req.query** is an object containing a property for each query parameter in the URL path
  - If you have the path /posts**?sortBy**=createdOnDate, then the "**sortBy**" property is available as req.query.**sortBy**

```
app.get('/api/students?sortBy=studentId', (req, res) => {
    // req.query.sortBy  => "studentId"
    const sortBy = req.query.sortBy
    console.log(req.query.sortBy', sortBy)
})
```


12

Working with a Request Body

- To access the request body a middleware is used to parse the request body
- **express.json()** is a middleware function that extracts the body portion of an incoming request and assigns it to **req.body**

```
import express from 'express';
const app = express();
app.use( express.json() );
app.post('/heroes', async (req, res) => {
    const hero = req.body;
    await heroRespository.addHero(hero);
    res.status(201);
});
```

Express Router

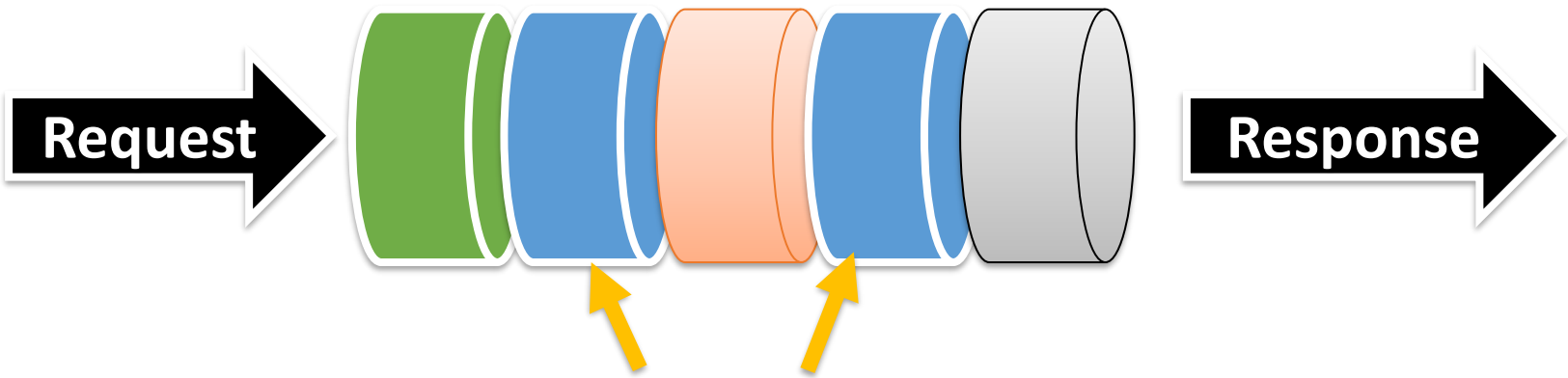
- For simple app routes can defined in **app.js**
- For large application, Express Router allows defining the routes in a separate file(s) then attaching routes to the app to:
  - Keep *app.js* clean, simple and organized
  - Easily find and maintain routes

```
// routes.js file
const router = express.Router()
router.get('/api/students', studentController.getStudents )
module.exports = router

//app.js file - mount the routes to the app
import { router } from './routes.js';
app.use('/', router);
```

Express Middleware

- Express middleware allows **pipelining** a request through a series of functions.
  - Each middleware function may **modify** the request or the response
- Request Processing Pipeline: the request passes through an *array* of functions before it reaches the route handler

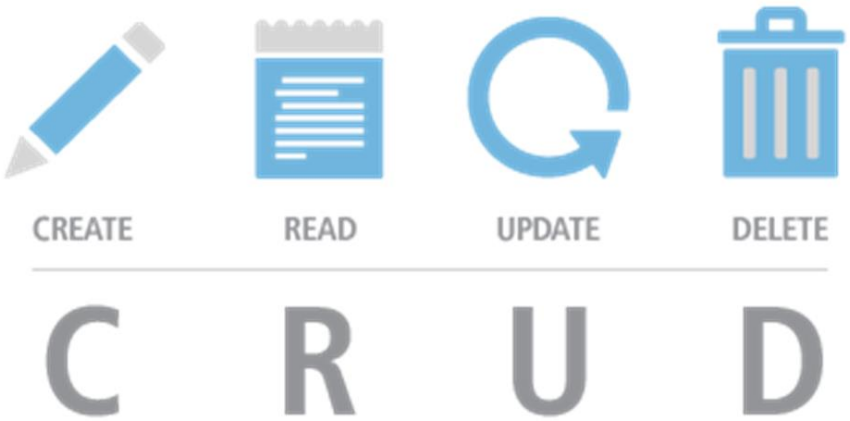
```
/* express.json() is a middleware function that extracts the body portion of
an incoming request and assigns it to req.body.
 */
app.use( express.json() );
```



Middleware (body parser, logging, authentication, router etc.)

Custom Middleware Example

```
import express from 'express';
const app = express();

//Define a middleware function
function logger (req, res, next) {
    req.requestTime = new Date();
    console.log(`Request received at ${req.requestTime}`);
    next();
}

// Attach it to the app
app.use(logger);

app.get('/', function (req, res) {
    const responseText = `Hello World! Requested at: ${req.requestTime}`;
    res.send(responseText);
})
```



Implementing CRUD Operations

CRUD Operations

- See the posted Hero and Student Examples

```
import heroService from './services/HeroService.js';

//Heroes Web API
router.route('/heroes')
    .get( heroService.getHeroes )
    .post( heroService.addHero );

router.route('/heroes/:id')
    .get( heroService.getHero )
    .put( heroService.updateHero )
    .delete( heroService.deleteHero );
```

Summary

- Express is a popular and easy to use web framework
- It makes building an Http Server and Web API a lot easier
- Provides routing and static content delivery out of the box
- Uses **express.json()** middleware to parse the request body

Resources

- Express Documentation

https://expressjs.com/en/5x/api.html

- Web API Design

- https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

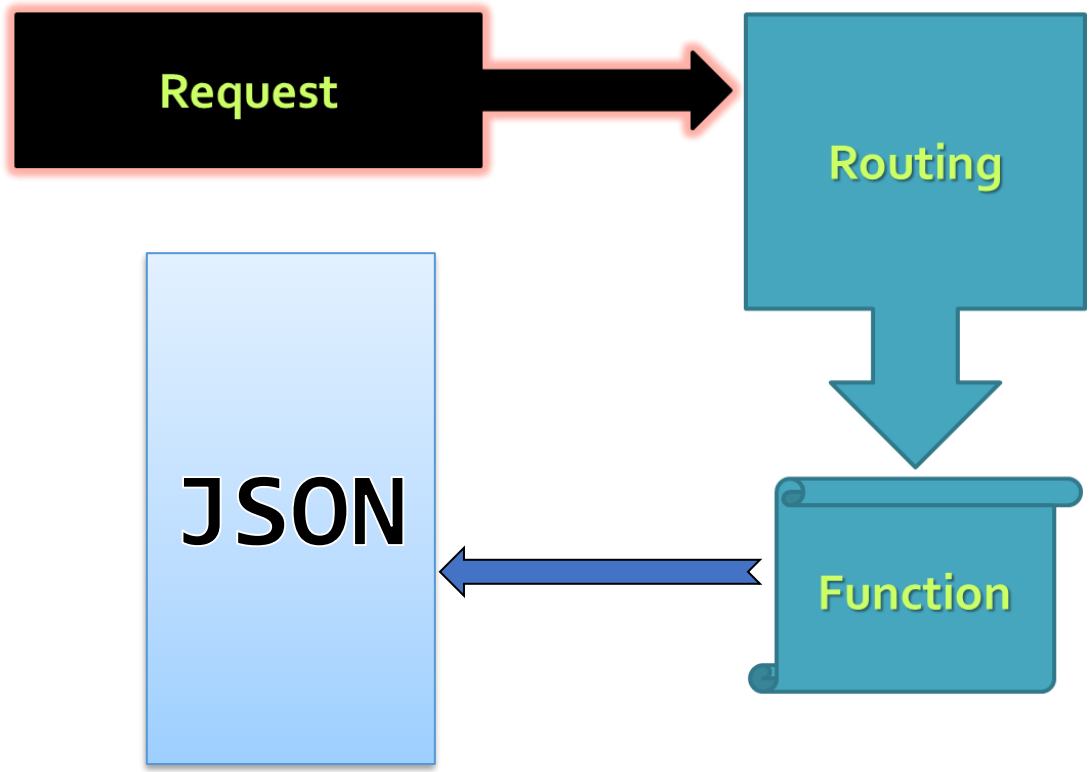
- https://cloud.google.com/files/apigee/apigee-web-api-design-the-missing-link-ebook.pdf

- Mozilla Developer Network

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs

3

Web API using NEXT.JS



Getting started

- Create an empty folder (with no space in the name use **dash -** instead)
- Create next.js app (select **No** for all questions)

```
npx create-next-app@latest .
```

```
√ Would you like to use TypeScript with this project? ... No / Yes
√ Would you like to use ESLint with this project? ... No / Yes
√ Would you like to use `src/` directory with this project? ... No / Yes
√ What import alias would you like configured? ... @/*
```

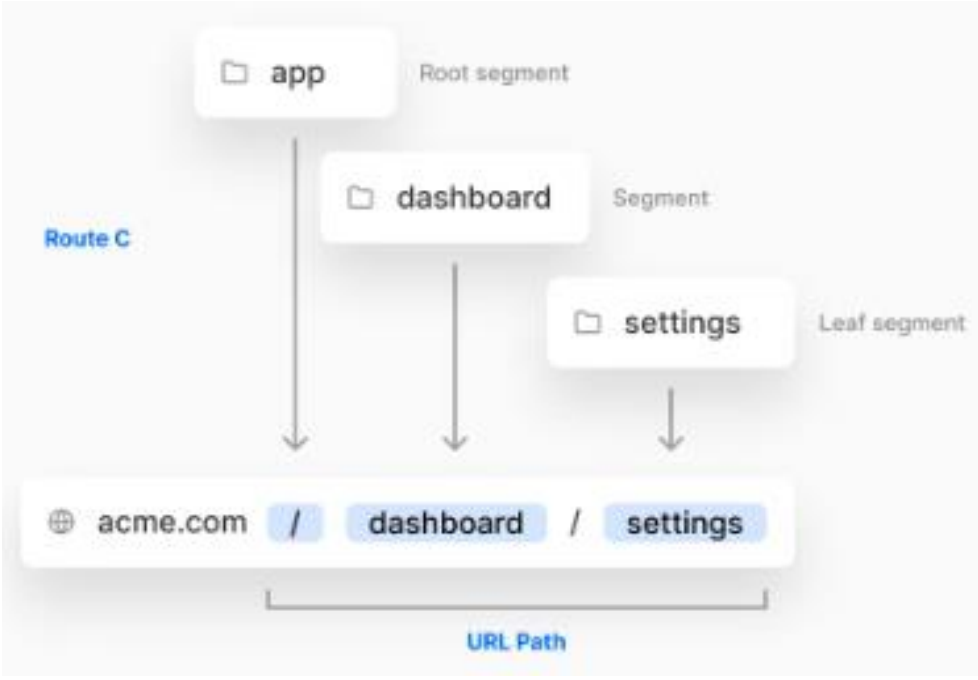
- Creates a new **`Next.js`** project and downloads all the required packages
- Run the app in dev mode: **`npm run dev`**

Next.js Routing

- Next.js has a **file-system based App Router**:
  - Folders inside the **app** directory are used to define routes
    - A route is a single path of nested folders, from the root folder down to a leaf folder
  - Files are used to create Web pages (`page.js`) or Web API (`route.js`)

- Each folder in the subtree represents a route segment in a URL path

- E.g., create `/dashboard/settings` route by nesting two subfolders in the app directory



API Routes

- Simply add a **`route.js`** file under the app folder
- A **`route.js`** file can export an async function named by the HTTP verbs: GET, HEAD, OPTIONS, POST, PUT, DELETE, and PATCH to handle incoming requests
- Every subfolder inside app folder having route.js is treated as a Web API endpoint e.g., **route.js** in **app/api/hello**

```
export async function GET(request) {
  return new Response('Hello, Next.js!');
}
```

- Visiting http://localhost:3000/api/hello will return Hello, Next.js!

API Routes

- Requests can be routed based on:
  - **HTTP Verb** – GET, POST, PUT, DELETE
  - **URL Path** – e.g., /users
- App Route **maps** an **HTTP Verb** (e.g., GET or POST) + a **URI Path** (like /users/123) to a **route handler function**
  - The handler function is passed a **request** object and returns a **Response** object
  - The **request** object represents the HTTP request and has the query string, body and HTTP headers
  - The **Response** object represents the HTTP response and it is used to send the generated response

25

Dynamic API Routes

- To create a dynamic route (having named path parameters) simply wrap the folder's name in square brackets **`[folderName]`**
  - Allows adding **path parameters** to the URL path. E.g., **/blogs/123**

Route	Example URL	params
app/blogs/[id]/route.js	/blog/123	{ id: '123' }
app/blogs/[id]/route.js	/blog/234	{ id: '234' }

- Dynamic segments are passed as **params** argument to the handler functions
  - E.g., if you have the path **/blogs/[id]**, then the "id" property is available as **params.id**

```
// app/blogs/[id]/route.js
export default function GET(request, { params }) {
  return new Response(`Blog id# ${params.id}`)
}
```

Catch-all dynamic routes

- **catch-all dynamic routes**: allows a dynamic route to catch all paths by simply adding ellipsis(...) inside the brackets [...folderName]
  - e.g., The catch all page in **app/blogs/[...filterBy]** will match any path underneath **/blogs** such as: /blogs/2023, /blogs/2023/3/10, and so on
  - Matched parameters array can be access using the **params**, so the path /blogs/2022/3/10 will have the following **params** object ["2022", "3", "10"]

Route	Example URL	params
app/blogs/[...filterBy]/route.js	/blogs/2023	{ filterBy: ['2023'] }
app/blogs/[...filterBy]/route.js	/blogs/2023/3	{filterBy: ['2003', '3']}

Optional Catch-all Segments

- Catch-all Segments can be made optional by including the parameter in double square brackets: **`[[...folderName]]`**
- For example, **`app/shop/[[...slug]]/route.js`** will also match `/shop`, in addition to `/shop/clothes`, `/shop/clothes/tops`, `/shop/clothes/tops/t-shirts`
- The difference between catch-all and optional catch-all segments is that with optional, the route without the parameter is also matched (`/shop` in the example above).

Query Parameters

- Named **query parameters** can be added to the URL path after a **?** E.g., **/products?sortBy=price**
- Query parameters are often used for **optional** parameters (e.g., optionally specifying the property to be used to sort of results)
- **request.url.searchParams** is an object containing a property for each query parameter in the URL path
  - If you have the path /products**?sortBy**=**price**, then the "**sortBy**" property can accessed as shown below:

```
export async function GET(request) {
  const { searchParams } = new URL(request.url)
  const sortBy = searchParams.get('sortBy')
  const res = await fetch(`https://data.api.com/products/?sortBy=${sortBy}`)
  const products = await res.json();

  return Response.json(products)
}
```


29

Working with a Request Body

- The request body can be retrieved using one of the following request methods:

.json() , **.text()** or **.formData()**

```
export async function POST(request) {
        let hero = await request.json()
        hero = await addHero(hero)
        return Response.json(hero, { status: 201 })
}
```

Headers

- You can read http headers with the **headers** library from next/headers package
- You can also return a new Response with new headers

```
import { headers } from 'next/headers'

export async function GET(request) {
  const headersList = headers()
  const referer = headersList.get('referer')

  return new Response('Hello, Next.js!', {
    status: 200,
    headers: { 'referer': referer }
  })
}
```

Redirect

- Sends a redirect response to another Url

```
import { redirect } from 'next/navigation'

export async function GET(request) {
  redirect('https://nextjs.org/')
}
```

Summary

- Next.js = React-based full stack web framework that allows creating server-side rendered pages, and Web API
- Next.js has a **file-system based router**: when a folder is added to the **`app`** directory, it's automatically available as a route
  - In Next.js you can add brackets to the folder name to create a dynamic route
- To create API Route simply add a **`route.js`** under `app/api`

Resources

- Learn Next.js

https://nextjs.org/docs

- Next.js blog

https://nextjs.org/blog