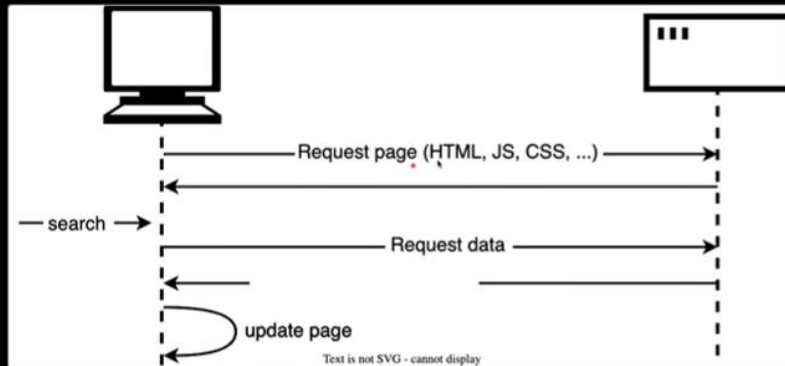


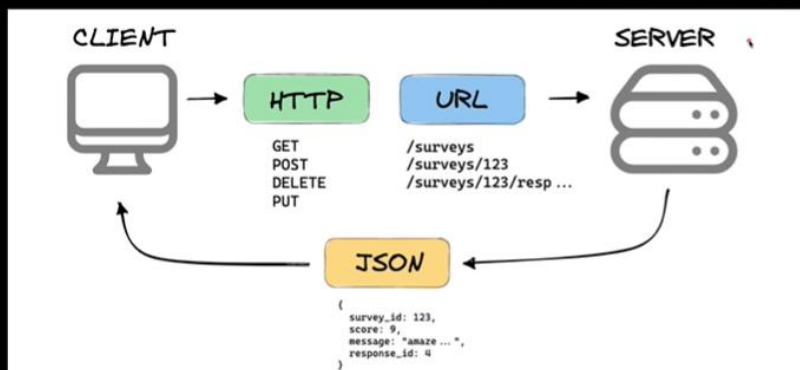
Json Request and Rest API

21.1 What are Async Requests



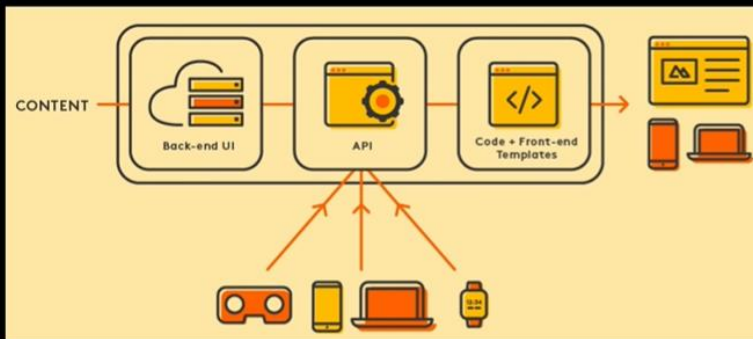
- Async network requests enable web pages to communicate with servers without reloading.
- The client sends JSON requests to the server asynchronously in single-page apps.
- The server processes the request and returns a JSON response.
- The page updates itself dynamically using the received data.

21.2 What are REST APIs



- REST APIs enable communication between clients and servers using HTTP.
- They are mainly identified by a URI.
- They use standard HTTP methods like GET, POST, PUT, and DELETE.
- Data is exchanged in formats like JSON or XML.
- REST APIs are stateless.
- REST APIs allow clients to access and manipulate web resources.

node 21.3 Decoupling Frontend & Backend



- Separating front-end and back-end allows independent development and scaling.
- REST APIs serve as a communication layer between them.
- Front-end interacts with back-end through standardized RESTful calls.
- Decoupling enhances flexibility and simplifies maintenance.
- REST APIs enable front-end updates without altering back-end code.

node 21.4 Routes & HTTP Methods

- REST API routes define the endpoints (URLs) where resources can be accessed by clients.
- **GET**: Retrieves data from the server at the specified route.
- **POST**: Sends new data to the server to create a resource.
- **PUT**: Updates or replaces an existing resource at a given route.
- **DELETE**: Removes a resource from the server at the specified route.
- **PATCH**: Partially updates an existing resource with new data.

superheroes			
GET	/api/dc/superheroes	Retrieves the collection of superheroes resources.	✓
POST	/api/dc/superheroes	Creates a superheroes resource.	✓
GET	/api/dc/superheroes/{id}	Retrieves a superheroes resource.	✓
PUT	/api/dc/superheroes/{id}	Replaces the superheroes resource.	✓
DELETE	/api/dc/superheroes/{id}	Removes the superheroes resource.	✓
PATCH	/api/dc/superheroes/{id}	Updates the superhero resource.	✓



21.5 REST Core Concepts



- **Statelessness:** Each request contains all necessary information; the server maintains no client session.
- **Uniform Interface:** Standardized communication using HTTP methods like GET, POST, PUT, DELETE.
- **Client-Server Separation:** Independent development of front-end and back-end components.
- **Cacheability:** Responses indicate if they can be cached to improve performance.
- **Layered System:** Architecture allows for multiple layers between client and server.
- **Code on Demand (Optional):** Servers can extend client functionality by sending executable code.



21.6 First API Todo App

8,9.

```
const bodyParser = require("body-parser");  
const cors = require("cors");
```

```
app.use(bodyParser.urlencoded({ extended: true }));  
app.use(cors());  
app.use(express.json());  
app.use(itemsRouter);  
app.use(errorController.get404);
```

```

itemsService.js X
20-RestApi_JsonRequest > todoApp-Frontend > services > itemsService.js > markTodoItems > res
1  const mapServerItemToLocalItem = (serverItem) => ({
2    Id: serverItem._id,
3    Name: serverItem.task,
4    Date: serverItem.date,
5    Completed: serverItem.completed,
6  });
7
8  export const addItemToServer = async (task, date) => {
9    const response = await fetch("http://localhost:3001/api/todos/items", {
10     method: "POST",
11     headers: {
12       "Content-Type": "application/json",
13     },
14     body: JSON.stringify({ task, date }),
15   });
16
17   if (!response.ok) {
18     throw new Error(`Server responded with status ${response.status}`);
19   }
20   const data = await response.json();
21   return mapServerItemToLocalItem(data);
22 };
23
24 export const getTodoItems = async () => {
25   const res = await fetch("http://localhost:3001/api/todos/getall");
26   const itemsList = await res.json();
27   // console.log('itemsList:', itemsList);
28   return itemsList.map(mapServerItemToLocalItem);
29 };
30
31 export const markTodoItems = async (id) => {
32   const res = await fetch(`http://localhost:3001/api/todos/completed/${id}`, {
33     method: "PUT",
34   });
35   const item = await res.json();
36   return mapServerItemToLocalItem(item);
37 };
38
39 export const deleteTodoItems = async (id) => {
40   const res = await fetch(`http://localhost:3001/api/todos/delete/${id}`, {
41     method: "DELETE",
42   });
43   const item = await res.json();
44   return item._id;
45 };

```

```

todoItemsRouter.js X
20-RestApi_JsonRequest > todoApp-Backend > routes > todoItemsRouter.js > ...
1  const express = require('express');
2  const { createTodoItems, getTodoItems, deleteTodoItems, markTodoItems } = require('../controllers/todoItemsController');
3
4  const todoItemsRouter = express.Router();
5
6  todoItemsRouter.post('/items', createTodoItems);
7  todoItemsRouter.get('/getall', getTodoItems);
8  todoItemsRouter.delete('/delete/:id', deleteTodoItems);
9  todoItemsRouter.put('/completed/:id', markTodoItems);
10
11 module.exports = todoItemsRouter;

```

```
20-RestApi_JsonRequest > todoApp-Backend > controllers > todoItemsController.js > ...
1  const TodoItem = require("../models/todoItem");
2
3  exports.createTodoItems = async (req, res, next) => {
4    // console.log("todoItems:", req.body);
5    const { task, date } = req.body;
6    const todoItem = new TodoItem({ task, date });
7    await todoItem.save();
8    res.status(201).json(todoItem);
9  };
10
11 exports.getTodoItems = async (req, res, next) => {
12   const todoItems = await TodoItem.find();
13   // console.log('all items are:', todoItems);
14   res.json(todoItems);
15 };
16
17 exports.deleteTodoItems = async (req, res, next) => {
18   const Id = req.params.Id;
19   const deleteItem = await TodoItem.findByIdAndDelete(Id);
20   res.status(201).json({ _id: Id });
21 };
22
23 // exports.markTodoItems = async (req, res, next) => {
24 //   const Id = req.params.Id;
25 //   const todoItem = await TodoItem.findById(Id);
26 //   todoItem.completed = true;
27 //   await todoItem.save();
28 //   res.json(todoItem);
29 // };
30
31 exports.markTodoItems = async (req, res, next) => {
32   const Id = req.params.Id;
33   try {
34     const todoItem = await TodoItem.findById(Id);
35     if (!todoItem) {
36       return res.status(404).json({ error: "Todo item not found" });
37     }
38
39     // Toggle completed value
40     todoItem.completed = !todoItem.completed;
41
42     await todoItem.save();
43     res.json(todoItem);
44   } catch (error) {
45     res.status(500).json({ error: "Server error" });
46   }
47 };
48
```

Sanar Gaudam (2 hours ago) | Ln 27, Col 28 | Spaces: 2 | UTF-8 | GBK | {} |