REST

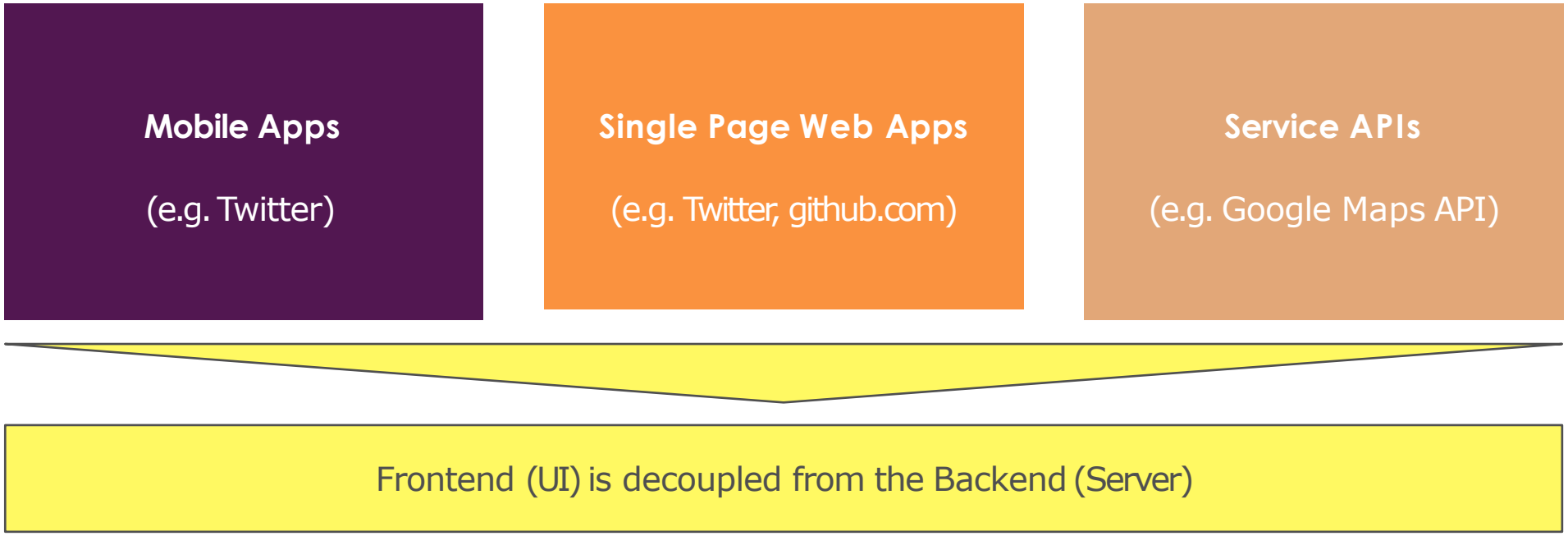
Rujuan Xing

What is REST?

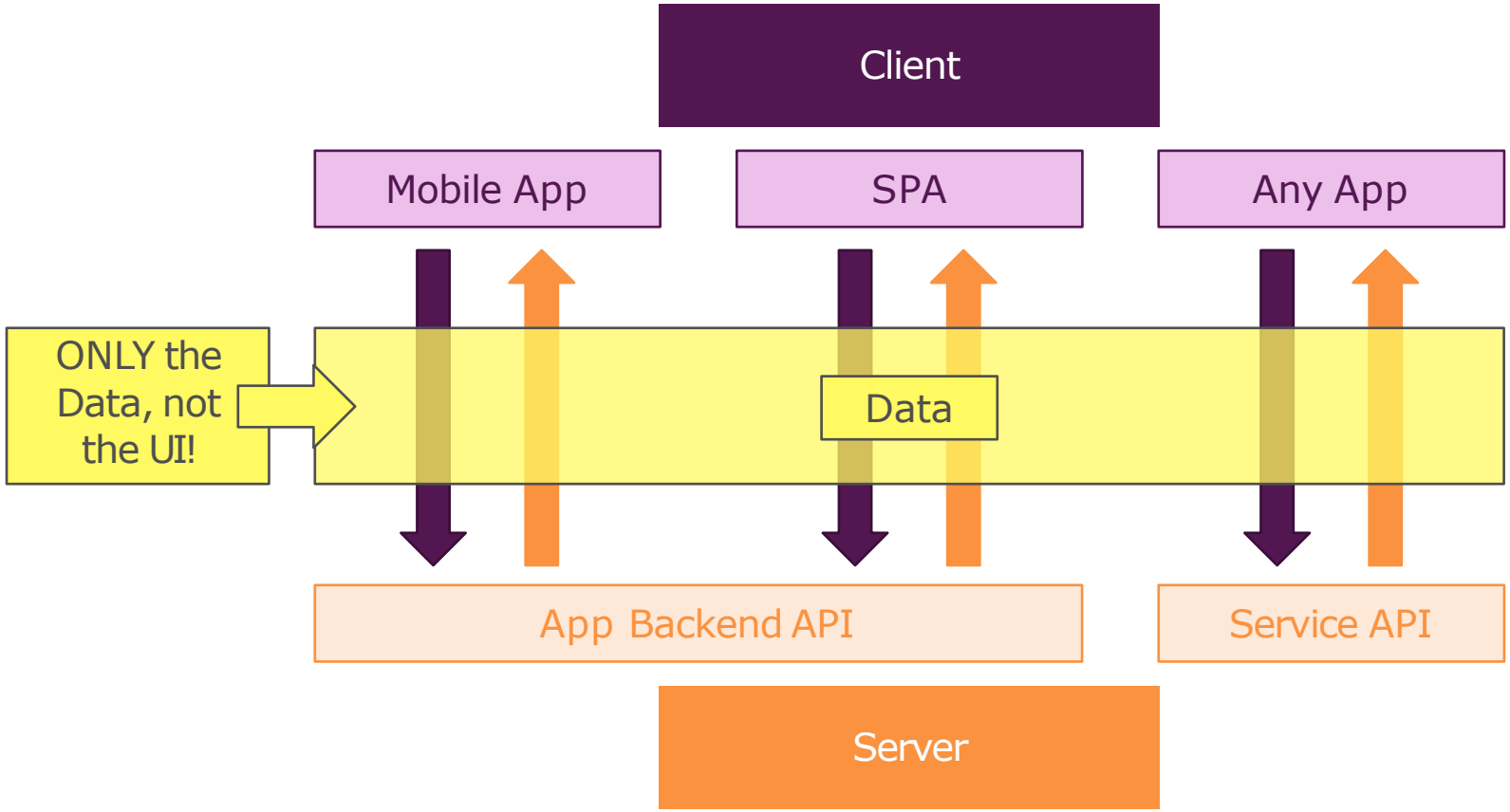
- REST = **RE**presentational **S**tate **T**ransfer
- REST is an architectural style consisting of a coordinated set of architectural constraints
- First described in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine
- RESTful is typically used to refer to web services implementing a REST architecture
- Alternative to other distributed-computing specifications such as SOAP
- Simple HTTP client/server mechanism to exchange data
- Everything – the UNIVERSE is available through a URI
- Utilizes HTTP: GET/POST/PUT/DELETE operations

Why REST?

Not every Frontend (UI) requires HTML Pages!



REST API Big Picture



Data Formats

HTML	Plain Text	XML	JSON
`<p>Node.js</p>`	`Node.js`	`<name>Node.js</name>`	`{"title": "Node.js"}`
Data +Structure	Data	Data	Data
Contains User Interface	No UI Assumptions	No UI Assumptions	No UI Assumptions
Unnecessarily difficult to parse if you just need the data	Unnecessarily difficult to parse, no clear data structure	Machine-readable but relatively verbose; XML-parser needed	Machine-readable and concise; Can easily be converted to JavaScript

Architectural Constraints

- Uniform interface
  - Individual resources are identified in requests, i.e.,using URIs in web-based REST systems.
- Client-server
  - Separation of concerns. A uniform interface separates clients from servers.
- Stateless
  - The client-server communication is further constrained by no client context being stored on the server between requests.
- Cacheable
  - Basic WWW principle: clients can cache responses.
- Layered system
  - A client cannot necessarily tell whether it is connected directly to the end server, or to an intermediary along the way.
- Code on demand (optional)
  - REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

Resource

- The key abstraction of information in REST is a **resource**.
  - a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on.
- Resource representation: consists of data, metadata describing the data and **hypermedia** links which can help the clients in transition to the next desired state.

Resource Naming Best Practices -Use nouns to represent resources

- Document:
  - a singular concept, like an object instance or db record.
  - Use "singular" name to denote document resource archetype.
    - http://api.example.com/device-management/managed-devices/{device-id}
    - http://api.example.com/user-management/users/{id}
    - http://api.example.com/user-management/users/admin

- Collection: sever-managed directory of resources.
  - Use "plural" name to denote collection resource archetype
    - http://api.example.com/device-management/managed-devices
    - http://api.example.com/user-management/users
    - http://api.example.com/user-management/users/{id}/accounts

Resource Naming Best Practices
-Use nouns to represent resources

- **store**
  - a client-managed resource repository.
  - Use "plural" name to denote store resource archetype.
    - http://api.example.com/cart-management/users/{id}/carts
    - http://api.example.com/song-management/users/{id}/playlists

- **controller**
  - A controller resource models a procedural concept.
  - Use "verb" to denote controller archetype.
    - http://api.example.com/cart-management/users/{id}/cart/checkout
    - http://api.example.com/song-management/users/{id}/playlist/play

Resource Naming Best Practices -Consistency is the key

- Use forward slash (/) to indicate hierarchical relationships
  - The forward slash (/) character is used in the path portion of the URI to indicate a hierarchical relationship between resources.
  - http://api.example.com/device-management
  - http://api.example.com/device-management/managed-devices
  - http://api.example.com/device-management/managed-devices/{id}
- Do not use trailing forward slash (/) in URIs
  - http://api.example.com/device-management/managed-devices/
  - http://api.example.com/device-management/managed-devices /*This is much better version*/
- Use hyphens (-) to improve the readability of URIs
  - http://api.example.com/inventory-management/managed-entities/{id}/install-script-location //More readable
  - http://api.example.com/inventory-management/managedEntities/{id}/installScriptLocation //Less readable
- Do not use underscores (_)
  - http://api.example.com/inventory-management/managed-entities/{id}/install-script-location //More readable
  - http://api.example.com/inventory_management/managed_entities/{id}/install_script_location //More error prone
- Use lowercase letters in URIs
- Do not use file extensions
  - http://api.example.com/device-management/managed-devices.xml /*Do not use it*/
  - http://api.example.com/device-management/managed-devices /*This is correct URI*/

Resource Naming Best Practices
-Never use CRUD function names in URIs

- HTTP request methods should be used to indicate which CRUD function is performed.
  - HTTP GET http://api.example.com/device-management/managed-devices //Get all devices
  - HTTP POST http://api.example.com/device-management/managed-devices //Create new Device

  - HTTP GET http://api.example.com/device-management/managed-devices/{id} //Get device for given Id
  - HTTP PUT http://api.example.com/device-management/managed-devices/{id} //Update device for given Id
  - HTTP DELETE http://api.example.com/device-management/managed-devices/{id} //Delete device for given Id

Resource Naming Best Practices
-Use query component to filter URI collection

- Many times, you will come across requirements where you will need a collection of resources sorted, filtered or limited based on some certain resource attribute. For this, do not create new APIs – rather enable sorting, filtering and pagination capabilities in resource collection API and pass the input parameters as query parameters. e.g.
  - http://api.example.com/device-management/managed-devices
  - http://api.example.com/device-management/managed-devices?region=USA
  - http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ
  - http://api.example.com/device-management/managed-devices?region=USA&brand=XYZ&sort=installation-date

HTTP Methods for RESTful APIs

HTTP METHOD	CRUD	ENTIRE COLLECTION (E.G. /USERS)	SPECIFIC ITEM (E.G. /USERS/123)
POST	Create	201 (Created), 'Location' header with link to /users/{id} containing new ID.	Avoid using POST on single resource
GET	Read	200 (OK), list of users. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single user. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method not allowed), unless you want to update every resource in the entire collection of resource.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
PATCH	Partial Update/Modify	405 (Method not allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). Use 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method not allowed), unless you want to delete the whole collection — use with caution.	200 (OK). 404 (Not Found), if ID not found or invalid.

JavaScript Object Notation (JSON)

- JSON (JavaScript Object Notation) is a lightweight data-interchange format.
  - Based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999.
  - A text format that is completely language independent.
  - Easy for machines to parse and generate.
    - Can convert any JavaScript object into JSON, and send JSON to the server.
  - Natively supported by all modern browsers
  - Replaced XML (Extensible Markup Language)

JavaScript Object Notation (JSON)

- JSON is a syntax similar to JS Objects for storing and exchanging data and an efficient alternative to XML.
- A name/value pair consists of a field name **in double quotes**, followed by a colon, followed by a value. Values can be any JS valid type except functions.

```
{"students":[
    {"firstName":"Ashim", "lastName":"Ghimire"},
    {"firstName":"Mohamed", "lastName":"Hassan"},
    {"firstName":"Leul", "lastName":"Necha"},
    {"firstName":"Shawn", "lastName":"Daudi"},
]}
```

- JSON values can be:
  - A number (integer or floating point)
  - A string (in double quotes)
  - A Boolean (true or false)
  - An array (in square brackets)
  - An object (in curly braces)
  - null

Browser JSON Methods

Method	Description
`JSON.parse(`*`string`*`)`	Converts the given string of JSON data into an equivalent JavaScript object and returns it
`JSON.stringify(`*`object`*`)`	Converts the given object into a string of JSON data (the opposite ofJSON.parse)

JSON expressions exercise

```
const jsonString = `
{
    "window": {
        "title":"Sample Widget",
        "width":500,
        "height":500
    },
    "image":{
        "src":"images/logo.png",
        "coords": [250,150,350,400],
        "alignment": "center"
    },
    "messages":[
        {"text": "Save", "offset":[10, 30]},
        {"text": "Help", "offset":[0, 50]},
        {"text": "Quit", "offset":[30, 10]},
        {"text": "Quit", "offset":[30, 60]}
    ],
    "debug":"true"
}
`;
const data = JSON.parse(jsonString);
```

Given the JSON data at right, what expressions would produce:

Using JavaScript Syntax on `data` object.

- The window's title?

```
let title = data.window.title;
```

- The image's third coordinate?

```
let coord = data.image.coords[2];
```

- The number of messages?

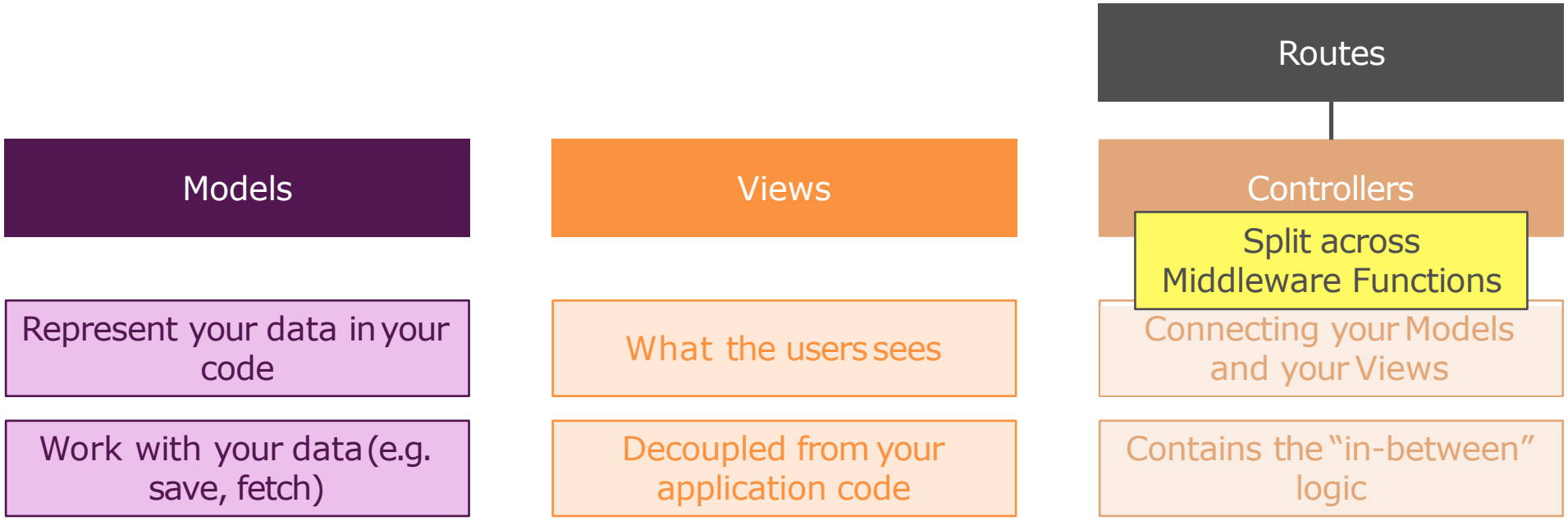
```
let len = data.messages.length;
```

- The y-offset of the last message?

```
let y = data.messages[len
1].offset[1];
```

What's MVC?

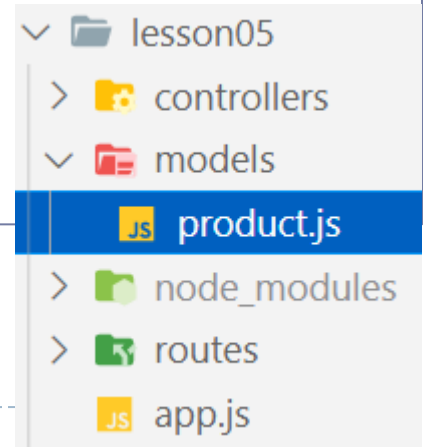
Separation of Concerns



Demo: Shopping Cart - Model

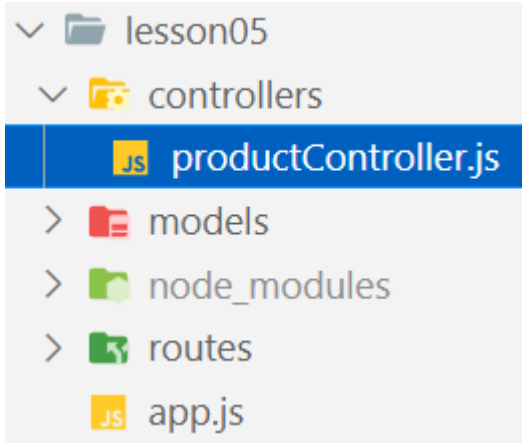
```
let products = [];

module.exports = class Product {

    constructor(id, title, price, description) {
        this.id = id;
        this.title = title;
        this.price = price;
        this.description = description;
    }

    save() {
        this.id = Math.random().toString();
        products.push(this);
        return this;
    }

    update() {
        const index = products.findIndex(p => p.id === this.id);
        if (index > -1) {
            products.splice(index, 1, this);
            return this;
        } else {
            throw new Error('NOT Found');
        }

    }
```

```
    static fetchAll() {
        return products;
    }

    static findById(productId) {
        const index = products.findIndex(p => p.id === productId);
        if (index > -1) {
            return products[index];
        } else {
            throw new Error('NOT Found');
        }
    }

    static deleteById(productId) {
        const index = products.findIndex(p => p.id === productId);
        if (index > -1) {
            products = products.filter(p => p.id !== productId);
        } else {
            throw new Error('NOT Found');
        }
    }

}
```



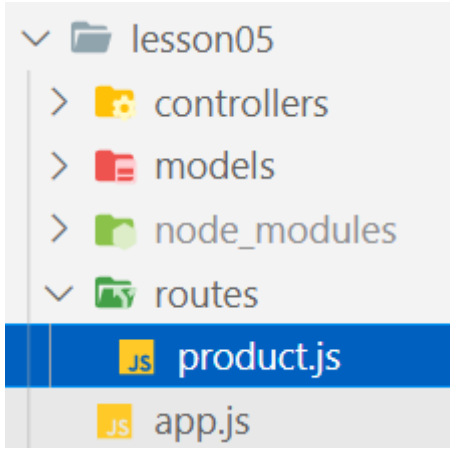
Demo: Shopping Cart – Controller

```
const Product = require('../models/product');

exports.getProducts = (req, res, next) => {
    res.status(200).json(Product.fetchAll());
}

exports.getProductById = (req, res, next) => {
    res.status(200).json(Product.findById(req.params.prodId));
}

exports.save = (req, res, next) => {
    const prod = req.body;
    const savedProd = new Product(null, prod.title, prod.price, prod.description).save();
    res.status(201).json(savedProd);
}

exports.update = (req, res, next) => {
    const prod = req.body;
    const updatedProd = new Product(req.params.prodId, prod.title, prod.price, prod.description).update();
    res.status(200).json(updatedProd);
}

exports.deleteById = (req, res, next) => {
    Product.deleteById(req.params.prodId);
    res.status(200).end();
}
```



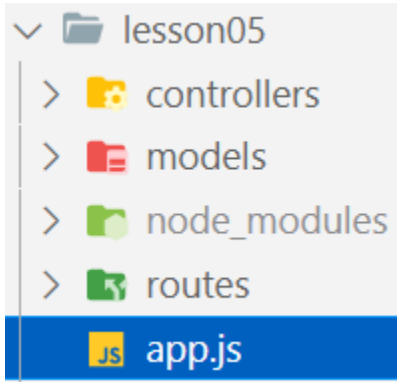
Demo: Shopping Cart – Route

```
const express = require('express');
const productController = require('../controllers/productController');

const router = express.Router();

router.get('/', productController.getProducts);

router.get('/:prodId', productController.getProductById);

router.post('/', productController.save);

router.put('/:prodId', productController.update);

router.delete('/:prodId', productController.deleteById);

module.exports = router;
```

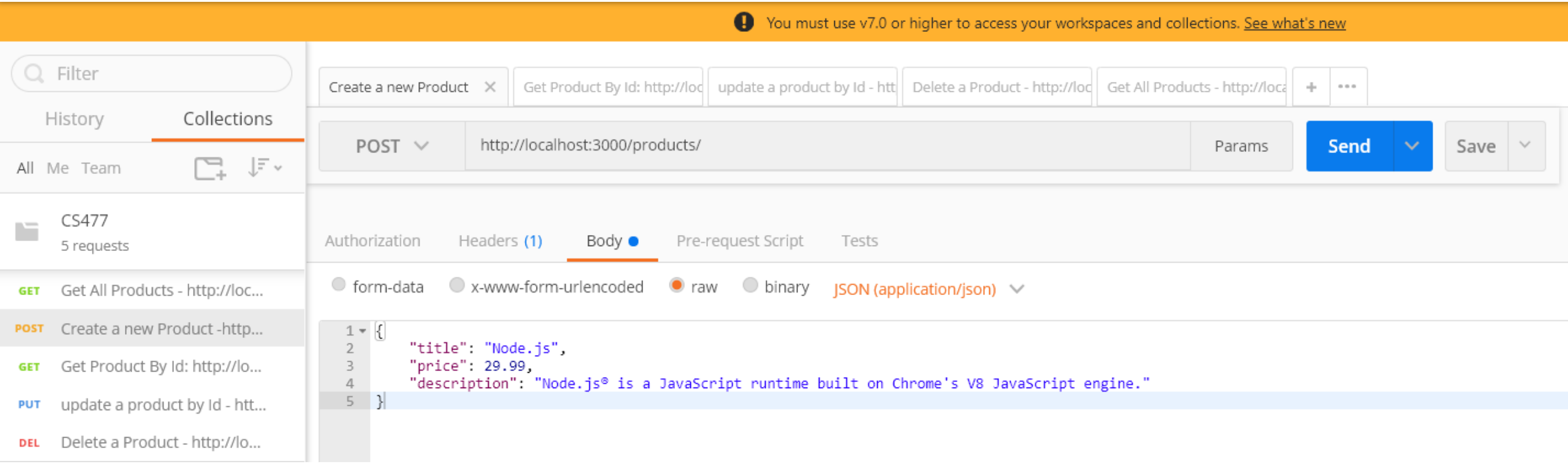


Demo: Shopping Cart – app.js

```
const express = require('express');
const productRouter = require('./routes/product');
const cors = require('cors');

const app = express();

app.use(cors());
app.use(express.json());

app.use('/products', productRouter);

app.use((req, res, next) => {
    res.status(404).json({ error: req.url + ' API not supported!' });
});

app.use((err, req, res, next) => {
    if (err.message === 'NOT Found') {
        res.status(404).json({ error: err.message });
    } else {
        res.status(500).json({ error: 'Something is wrong! Try later' });
    }
});

app.listen(3000, () => console.log('listening to 3000...'));
```



Demo: Shopping Cart – Testing APIs



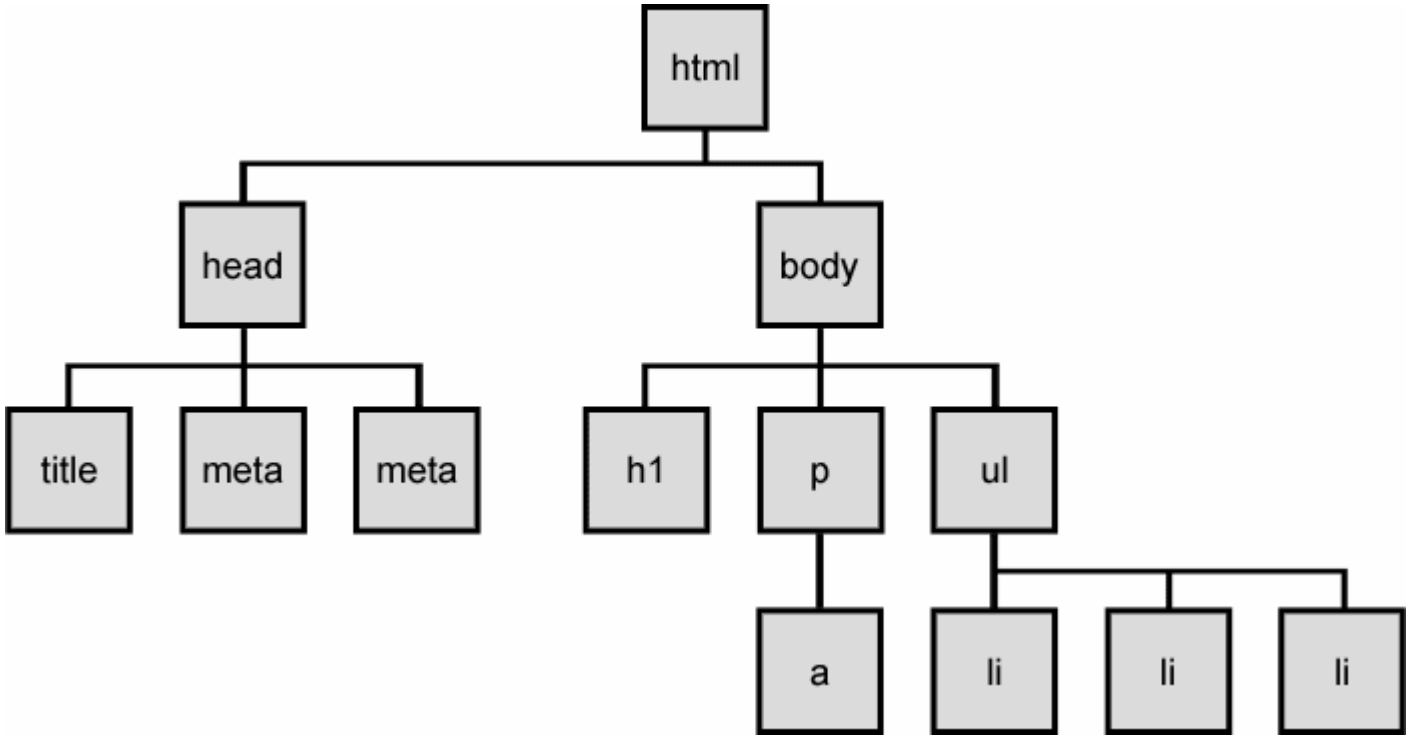
Event Handling

Global Objects

- The **`window`** object the top-level object in hierarchy
- The **`document`** object the DOM elements inside it
- The **`location`** object the URL of the current web page
- The **`navigator`** object information about the web browser application
- The **`screen`** object information about the client's display screen
- The **`history`** object the list of sites the browser has visited in this window

The DOM tree

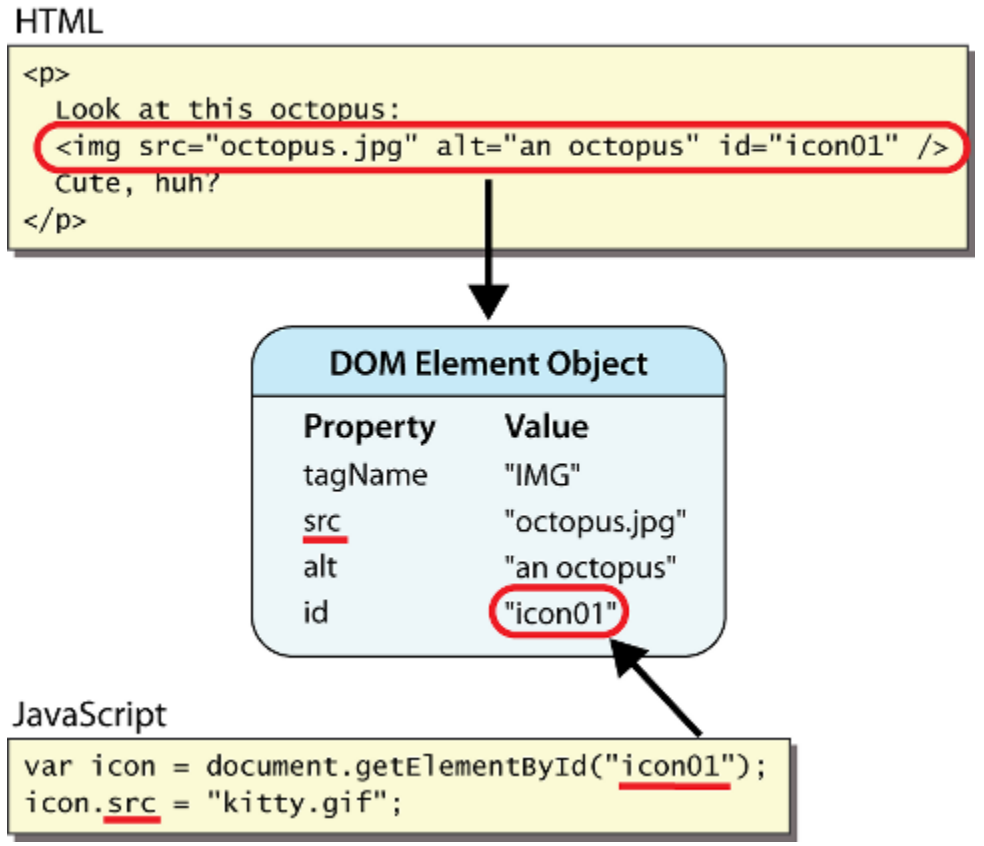
The elements of a page are nested into a tree-like structure of objects



DOM element objects

- Every element on the page has a corresponding DOM object

- We can simply read/modify the attributes of the DOM object with **`objectName.propertyName`**



Accessing the DOM in JS

- DOM Selectors are used to select HTML elements within a document using JavaScript.
- A few ways to select elements in a DOM:
  - `getElementsByTagName()`
  - `getElementsByClassName()`
  - `getElementById()`
  - `querySelector()`
  - `querySelectorAll()`
- All those methods are methods in the document object.
- What is the performance difference between the methods? return types?

HTML DOM Events

- HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document.
  - `click, touch, load, drag, change, input, error, resize` — there are more…
- Events can be triggered on any part of a document, whether by a user's interaction or by the browser.
- Events are normally used in combination with functions, and the function will not be executed before the event occurs (such as when a user clicks a button).

Listen for an event

- Two ways to listen for an event:
  - `element.onevent = function1;`
  - `element.addEventListener('event', myFunction1);`

- What's the difference between them?
  - The main difference is that `onclick` is just a property. If you write more than once, it will be overwritten.
  - `addEventListener()` on the other hand, can have multiple event handlers applied to the same element. It doesn't overwrite other present event handlers.

```
const btn1 = document.getElementById("btn1");
btn1.onclick = function () {
    console.log('Button 1 clicked.....1');
}
btn1.onclick = function () {
    console.log('Button 1 clicked.....2');
}
```

```
const btn2 = document.getElementById("btn2");
btn2.addEventListener('click', function () {
    console.log('Button 2 clicked.....1');
});
btn2.addEventListener('click', function () {
    console.log('Button 2 clicked.....2');
});
```

Mouse Event

- Events that occur when the mouse interacts with the HTML document belongs to the MouseEvent Object.

Event	Description
onclick	The event occurs when the user clicks on an element
oncontextmenu	The event occurs when the user right-clicks on an element to open a context menu
ondblclick	The event occurs when the user double-clicks on an element
onmousedown	The event occurs when the user presses a mouse button over an element
onmouseenter	The event occurs when the pointer is moved onto an element
onmouseleave	The event occurs when the pointer is moved out of an element
onmousemove	The event occurs when the pointer is moving while it is over an element
onmouseout	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children
onmouseover	The event occurs when the pointer is moved onto an element, or onto one of its children
onmouseup	The event occurs when a user releases a mouse button over an element

Window Event

- Events triggered for the window object (applies to the <body> tag):

Attribute	Description
onerror	Script to be run when an error occurs
onload	Fires after the page is finished loading
onresize	Fires when the browser window is resized
onunload	Fires once a page has unloaded (or the browser window has been closed)

```
window.onload = function() {
    alert('hi');
}
```