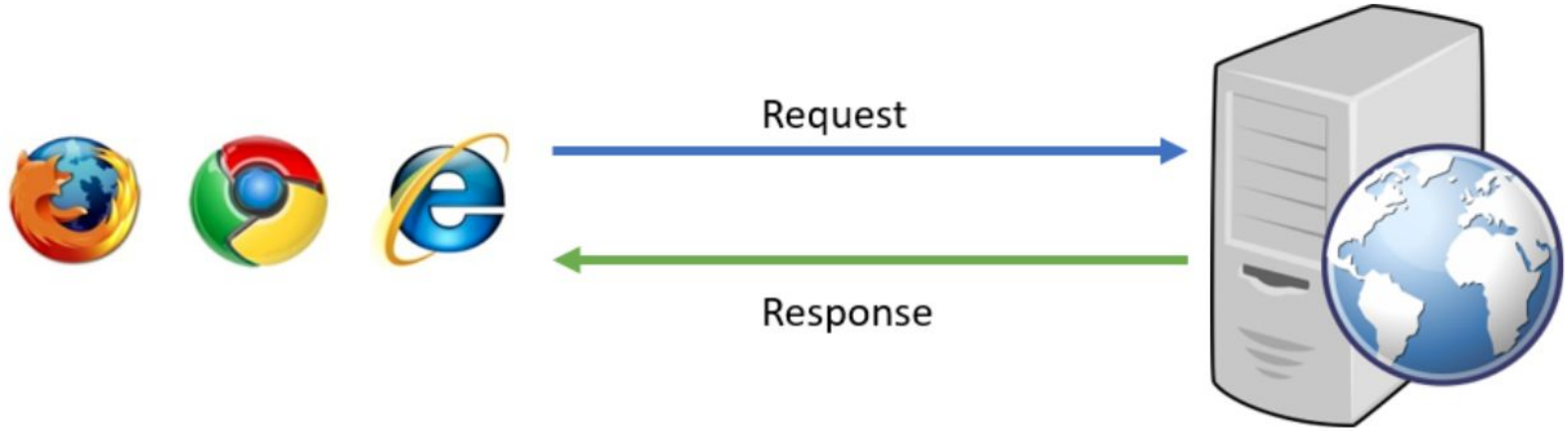


Module 3 - Lecture 9

Accessing Web APIs with JavaScript



HTTP Revisited



Live Score Tracking (2006)



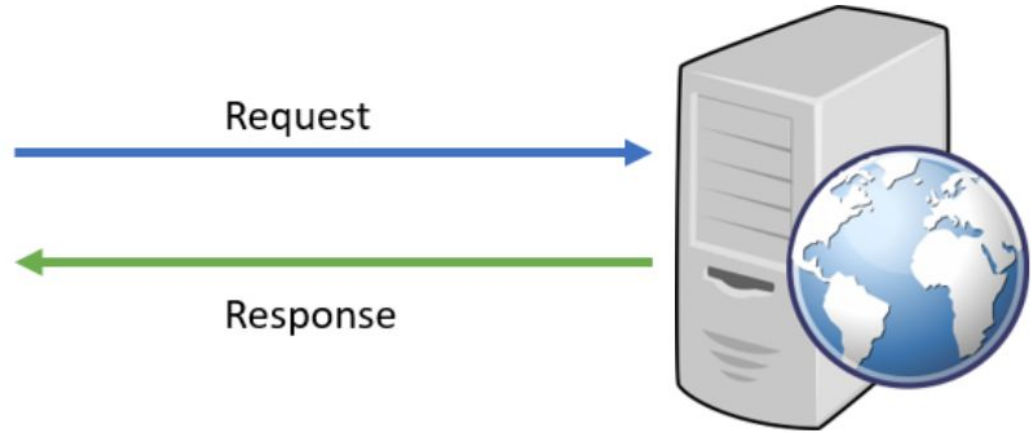
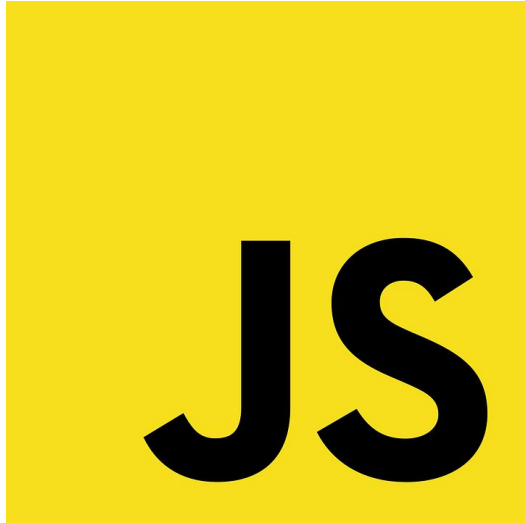
The screenshot shows a web browser window titled "ESPN - NHL RealTime Scoreboard". The page features the ESPN logo and "REALTIME" text. The main heading is "NHL Scoreboard" followed by the date "Thursday, January 12, 2006". Navigation links for "Schedule", "Standings", and "NHL Insider" are present. The scoreboard is organized into a grid of game boxes, each showing the home and away teams, the current score, the game period, and the time remaining. Some games are marked as "Final". At the bottom, there are links for "Feedback", "About ESPN RealTime", and "Message Board", along with a copyright notice for 2004 ESPN.com and the "insider" logo.

Game	Home Team	Score	Status	Time
Montreal vs Boston	Montreal	0 - 3	Final	
Detroit vs Colorado	Detroit	1 - 2	2nd Per.	10:35
Vancouver vs Phoenix	Vancouver			10:30 PM ET
Toronto vs Philadelphia	Toronto	4 - 5	Final	
San Jose vs Los Angeles	San Jose			10:30 PM ET
Buffalo vs Pittsburgh	Buffalo	2 - 1	3rd Per.	9:36
Florida vs Nashville	Florida	3 - 0	3rd Per.	11:35
Columbus vs Chicago	Columbus	3 - 3	2nd Per.	3:22

How does this work?



HTTP (2006)



Fetch API

- The Fetch API provides an interface for fetching resources.
- It uses a Request / Response model.
- It can work with other protocols other than HTTP, but we'll ignore those for now.
- Fetch API is somewhat equivalent to Spring's RestTemplate.



Asynchronous Programming

“Have a seat. We’ll bring out your order when it’s finished”



What's wrong with this?

```
function takeOrder() {  
    // take down the order and return it  
}  
  
function cookOrder(orderRequest) {  
    // prepare order per the request and return it|  
}  
  
function serveOrder(customer, cookedOrder) {  
    // serve cooked order to customer  
}  
  
function doRestaurant() {  
    customers.forEach(customer => {  
        const orderRequest = takeOrder(customer);  
        const cookedOrder = cookOrder(orderRequest);  
        serveOrder(customer, cookedOrder);  
    });  
}
```



Promises

- A promise to supply a value at some later point.
- Allows you to associate handlers with an asynchronous action's eventual success value or failure reason.
- 3 states of a Promise
 - ***pending***: initial state, neither fulfilled nor rejected.
 - ***fulfilled***: meaning that the operation was completed successfully.
 - ***rejected***: meaning that the operation failed.
 -
- .then() for accessing returned Promise
- .catch() for handling errors



Asynchronous Approach

```
function takeOrder() {  
    // take down the order and return it  
}  
  
function cookOrder(orderRequest) {  
    // prepare order per the request and return it  
}  
  
function serveOrder(customer, cookedOrder) {  
    // serve cooked order to customer  
}  
  
function doRestaurant() {  
    customers.forEach(customer => {  
        takeOrder(customer)  
            .then((orderRequest) => {  
                return cookOrder(orderRequest);  
            })  
            .then((cookedOrder) => {  
                serveOrder(customer, cookedOrder);  
            });  
    });  
}
```



Cross Origin Resource Sharing (CORS)

- Your browser enforces a policy that prevents requests from going to a different domain than the current one.
- The web server has influence over this. They can whitelist domains to permit them through.
- In Spring this can be done using the annotation **@CrossOrigin**
 - This can be added to the Controller or Method Handler.

```
@CrossOrigin(origins = { "http://127.0.0.1:5500", "http://localhost:5500" })
```



QUESTIONS?

