Summary - part 2

Outline

1. ES6
2. Promise
3. Generators
4. Database
5. REST API design
6. Session state and Security
7. Cloud (Guest lecture)

ES6

Has many features to ease programming, but mostly syntactic sugar

Use of **class** and **constructor** keywords to replace prototypal construction

Use of **extends** and **super keywords** for inheritance and super class

Arrow functions for unnamed functions with cleaner syntax

- Cannot support everything that traditional functions do
- Binding of ‘this’ is lexical rather than dynamic - more intuitive
- Allows the use of multiple levels of nesting without losing ‘this’

Promise

Built-in object introduced in ES6

Cleaner interface for handling asynchronous operations - no callback hell

Promise constructor takes an executor function with 2 arguments, resolve, reject

- Executor function is called synchronously and right away
- resolve() and reject() handlers are called asynchronously
- Executor function must explicitly call the resolve and reject handlers
- .then clause if promise resolves (second argument if promise rejects)
- .catch clause if promise rejects (equivalent to second argument of then)
- .finally clause to handle cleanup etc regardless of rejection or resolution

Promise (continued)

Promises can be cascaded by adding .then handlers in sequence - each will be called when (if) the previous promise resolves

Promise.all(promises)

- Returns a promise that resolves when *all* promises resolve
- Rejects right away if even a *single promise* rejects

Promise.race(promises)

- Returns a promise that resolves if *any* promise resolves
- Rejects right away if any of the promises rejects

Generators

Used to keep track of state of the iteration in a synchronous manner

Generator functions denoted by function*

Yield statement to return control to caller, and remember state of iteration

Initial call to generator function yields an ‘Iterable’ object, with next() method

Two ways to use Iterable object

- Using ‘for-of’ loop: Cleaner syntax, easier to use; some limitations
- Using .next() of the iterable: Need to check for .end explicitly; more powerful

Generators (continued)

Generators cannot be easily used within (asynchronous) callbacks as the yield must only be within the generator function

How to use generators with asynchronous code ?

- Use Promise; generator function yields promises on every call
- Promise can resolve or reject later - this is not in the generator’s control
- Can be used to create complex asynchronous workflows

Databases

Traditional (SQL) DBMS were relational - data stored in tables, rigid schema

- Native support for join to avoid redundancy - expensive to perform
- Support for ACID semantics (Atomicity, Consistency, Isolation, Durability)
- Performance was an issue; not horizontally scalable

New age Databases - NoSQL

- Compromise consistency for availability and partition tolerance (CAP thrm.)
- Exhibit horizontal scalability; provide eventual consistency

Databases (continued)

MongoDB

- Document oriented database
- Provides tunable consistency
- Document can have arbitrary structure; JSON formatted
- Documents stored in the form of collections
-

Operations to add, remove, find and modify documents in a collection (JSON)

Need to provide support for joins programmatically

RESTful API Design

REST: Architectural style that underpins the web

- 6 principles (Client-Server, Statelessness, Cacheable, Layered, Uniform interfaces, and code on demand)
- Implemented in the form of the HTTP protocol: GET, PUT, POST, DELETE
- Idempotent: GET, PUT, DELETE; Safe: GET; Neither: POST, PATCH

Making an API RESTful

- Identify resources; both collections and individual elements of the collection
- Collection must typically support GET (retrieve), POST (add to the collection)
- Elements must support: GET (Retrieve), PUT (modify), DELETE (Remove)
- Supporting pagination and query; don’t expose unnecessary resources

Session and Cookie

- At a high-level, a **session** is something that **keeps track of the series of interactions** between communicating parties
- In the context of **web applications**, a session keeps track of the communication **between the server and the client**
- **Cookie** is a piece of data that is always passed between the server and the client in consecutive HTTP messages
- At the minimum, a cookie can store a session ID to relate multiple HTTP requests and responses

Web Security: Cross-site Scripting

- **Cross-site Scripting** is executing a foreign (and malicious) piece of code as if it was included in the compromised webpage
  - Somehow inject JavaScript code into a resource of the attacked domain so that the code executes with the authority of the attacked domain
- **Defense:**
  - Sanitizing user input by checking for JavaScript code
  - Use “HttpOnly” flag for cookies
  - Content security policy (allow servers to specify approved origins of content for web browsers)

Web Security: Cross-site Request Forgery

- **Cross-site Request Forgery** is when an attacker attempts to make the victim submit a compromised form using their own credentials
  - Relies on the use of reproducible and guessable URLs or POST forms
  - Cookies are automatically sent with every request, and hence the URL can perform malicious actions on behalf of the victim
- **Defense:**
  - Make the URL hard to guess by attaching a random nonce or client-specific key to it
  - Create a short-lived, unique, and private form for each legitimate client, preventing the attacker from setting up a malicious form in advance

Some thoughts about the final exam

Make sure you study well; practice the class activities yourself by writing code

Do NOT assume that because the exam is open book, open notes, you can look up core concepts (you can, but you won’t have time to digest them and apply them)...

Make sure you really understand how to use the things we learned in class

Exam will have approx. equal coverage of pre mid-term and post mid-term portions

9 Multiple choice questions, 8 programming questions of 2 marks each

- Test cases will be provided for programming questions; **no partial credit**

Final thoughts…

I’ll be active on Piazza during final exam period until Dec 17th (2 days before the exam). The final exam will be administered similar to midterm exam (on Zoom).

Final exam is open-book, open notes, but no collaboration is allowed whatsoever

Class participation marks will be assigned **after the final exam** (say Dec 21st)

Hope you had fun in this course - thanks for all the interactions and questions

We have undergraduate internship positions open in my lab in the area of IoT

- Many of these require strong knowledge of JS; contact me if you’re interested