Explain what are prototypes and how does class inheritance make use of them?

To create objects that adopt properties and methods from other objects, many programming languages employ class-based inheritance. In contrast, JavaScript utilizes what's known as prototypal inheritance. Within JavaScript, one object can adopt attributes from another. The source object, which lends its attributes, is termed the prototype. Essentially, objects can acquire attributes from their prototype objects.

When attempting to retrieve an object's property, JavaScript doesn't only look within the object itself. It also seeks that property in the object's prototype, and then in the prototype's prototype, and so on. This continues until a matching property is located or the end of the prototype sequence is hit. If JavaScript doesn't find the property or method throughout this chain, it will then yield 'undefined'.

The foundational layer of the prototype sequence is Object.prototype. Behind the scenes, everything in JavaScript stems from Object.prototype, serving as a foundational pillar of the prototype sequence.

In the realm of inheritance, JavaScript's structure is singularly based on objects. Every object possesses an internal property (often termed its [[Prototype]]) which links it to its parent prototype object. This parent prototype has its own link to another prototype, and this linkage continues until reaching an object with a null prototype. By nature, null lacks a prototype, signifying the conclusion of the prototype chain.

When starting a new project how would you choose between OOP and Functional Programming?

When starting a new project the best choice often depends on the specifics of the project and the context in which you're working. We must consider the nature of your project, the strengths of each method, and the expertise of our team and the maturity of the ecosystem. Specificaly, if we want more modularity, resusability, statefulness e.g (Games, GUI Apps) OOP might be a better choice. However, if we want to head towards Safety, Predictability, Immutability, Concurrency e.g (Data processing pipelines) Functional Programming might be preferable. It is important to note also that your team matters a lot, for example, if your team has extensive experience with OOP but is new to FP, there will be a learning curve. Consider the time and resources needed for training. Last but not least, maturity of the ecosystem matters a lot, some ecosystems naturally lean towards one paradigm. For example, OOP Languages like Java,C# have OOP ecosystems, while Haskell and Erlang are leaning in FP. We should consider the libraries, frameworks, and tools available for your chosen language and how they fit with your project's needs.

How does Proxy work in JS and when is it useful?

In JavaScript, proxies (proxy object) are used to wrap an object and redefine various operations into the object such as reading, insertion, validation, etc. Proxy allows you to add custom behavior to an object or a function.

Proxies are usefull: - For Validation, you can use a proxy for validation. You can check the value of a key and perform an action based on that value. - Read Only View of an Object, there may be times when you do not want to let others make changes in an object. In such cases, you can use a proxy to make an object readable only. - Side Effects, you can use a proxy to call another function when a condition is met.

What patterns/practices/tools would you use to implement simple cache for NoSQL database?

Caching is a powerful mechanism to enhance the performance of applications, especially when working with databases, by reducing the need to fetch data repeatedly from the primary datastore.

Here are some patterns, practices, and tools to implement a simple cache:

Caching Patterns: -Cache-Aside (Lazy Loading): The application code controls the cache. When a piece of data is requested, the system first checks the cache. If it's not there, it fetches from the database, stores it in the cache, and then returns it.
-Read-Through Cache: The cache itself manages the data loading. If data isn't found, the cache fetches it from the database, stores it, and returns it. This often requires more sophisticated caching solutions.

Cache Eviction Policies: Deciding how to remove old or unnecessary data is crucial. - LRU (Least Recently Used): Removes the least recently accessed items. - FIFO (First

In, First Out): Removes the oldest data first.

Cache Storage Options: -Redis: A high-performance, in-memory key-value store. It supports strings, hashes, lists, sets, and other data structures. -Memcached: An in-memory key-value store for small chunks of arbitrary data.

Middleware and Libraries: Depending on the application's language and framework, there might be middleware or libraries to simplify caching. - Node.js: node-cache, express-redis-cache - Python: django-redis, Flask-Caching

Consistency: It's essential to maintain consistency between the cache and the database, Ensure the cache is updated immediately upon a write operation.

What libraries do you consider necessary for any application? Which ones do you use most commonly?

I would say for me the most essentials are the following:

Security: OAuth Libraries: Authorization tool. JWT Libraries: JSON Web Tokens for securely transmitting information. Bcrypt: Library for hashing passwords

General(not a libraries, but a crucial tools): Git: Version control system. Docker: Platform to develop, ship, and run applications inside containers. Postman: A collaboration platform for API development. CI/CD Tools: GitHub Actions, Jenkins Swagger/OpenAPI: tool for auto-generating documentation

Web Development (Frontend): Typescript: Strongly typed JS, giving you better tooling at any scale. React/Vue: A library for building user interfaces. Redux: Tool to manage the State Material UI: fully-loaded component library Vite: tool that aims to provide a faster and leaner development experience

Web Development (Backend, will depend on the language of choice): Web development frameworks: Flask, Django, Express, Spring: ORMs : SQLAlchemy, Django ORM, Mongoose,Hibernate Configuration Management: python-dotenv, dotenv Testing Tools: unittest, Jest, JUnit Caching: redis-py, node-redis

How would you choose a backend? When would you use HTTP server, serverless functions or Websockets?

Choosing the right backend approach depends on a lot of factors such as the application's requirements, expected traffic, user experience expectations, cost, and maintenance.

HTTP Servers: Can be preferable in many applications because of several reasons such as that they are mature and well-understood, lots of frameworks, tools, and community support and will give you a lot of flexibility in setting up the server environment, middleware, routes, etc. When you have standard request-response interaction patterns they might be a better choice.

Example: You're building an online store where users can browse products, add them to their cart, and proceed to checkout.

Serverless Functions: They offer important advantages because the help with auto-scaling, can be less costly cause you only pay for the execution time of your functions. Also, for CPU-intensive tasks you don’t want to run dedicated servers 24/7 for a task that might only occur occasionally or they can be suitable if you want to reduce maintanance overhead in which case the cloud provider will handle those events concurrently without any manual intervention.

Example: Users upload photos to your platform, and you need to process these photos (resize, apply filters, etc.) before storing them, event driven functionalities in general.

WebSockets: May be the best option if you plan to build real-time applications like chat apps, online gaming, and live sports updates because of the immediate data push from the server to the client without the client polling for data and the reduced latency compared to HTTP.

Example: Real-time Chat Application, Users can join chat rooms and communicate with others in real-time, in general real-time data apps.

Code below is supposed to print [{name: "Tom", id: 0}, {name: "Kate", id: 1}]. Explain why it doesn't and explain how would you fix it?

class IdGenerator { lastId = 0; getId() { return this.lastId++; } } const { getId } = new IdGenerator(); const people = ["Tom", "Kate"].map((name) => ({ name, id: getId() })); console.log(people);

Problem: It doesn't work because of the descructor on the getId. When you destructure the getId method out of the instance of IdGenerator, you're essentially detaching it from its original context. the "this.lastId" is undefined and undefined++ results in NaN

Fix: A potential fix would be to use a function without destructuring: Instead of destructuring, directly reference the method from the instance. const generator = new IdGenerator(); const people = ["Tom", "Kate"].map((name) => ({ name, id: generator.getId() }));

Another way would be binding const generator = new IdGenerator(); const getId = generator.getId.bind(generator);