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

Prototypes are a way to implement object-oriented concepts and facilitate code reuse. A prototype is an object that serves as a blueprint or template for creating new objects. It contains a set of properties and methods that define the behavior and characteristics of the objects created from it.

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

When starting a new project, the choice between Object-Oriented Programming (OOP) and Functional Programming can be illustrated with an example:

Let's say we're developing a web application for an e-commerce platform. The project requirements include modeling various entities such as customers, products, and orders, as well as implementing complex interactions between them.

In this case, OOP might be a suitable choice. We can define classes for each entity, such as a Customer class, Product class, and Order class. These classes can have their own properties and methods, allowing us to encapsulate behavior and data within each object. us can leverage concepts like inheritance to create specialized classes (e.g., a PremiumCustomer class inheriting from Customer) and polymorphism to handle different types of products or orders.

On the other hand, if our project primarily focuses on data transformations and composing functions, Functional Programming could be more appropriate. For example, we can use functional concepts like immutability and pure functions to handle data operations. We might have functions to calculate discounts, filter products based on certain criteria, or aggregate order data.

**- How does proxy work in JS and when is it useful?**

In JavaScript, a proxy is an object that intercepts and controls the fundamental operations of another object. It allows us to define custom behavior for property access, assignment, function invocation, and more. Proxies are useful for validation, filtering, logging, debugging, virtual properties, object observing, security, and access control. They provide a middle layer between code and an object, enabling modification, validation, and monitoring of object operations. However, proxies may introduce some performance overhead and are typically used selectively in scenarios where their benefits are significant.

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

In-Memory Data Store (e.g., Redis, Memcached)

Caching Strategy (e.g., LRU, TTL)

Cache Invalidation

Key Generation

Cache-Control Headers

Query Result Caching

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

React, Express, Axios, Lodash, Jest, Redux, Sequelize

One library that is commonly used is React Router. It enables navigation and routing capabilities in React applications, allowing developers to create single-page applications with multiple views and URLs.

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

HTTP Server: If we need a traditional client-server architecture for web applications or APIs, you can use backend libraries such as Express.js (Node.js), Django (Python), or Ruby on Rails (Ruby). These libraries provide a framework to handle HTTP requests and responses, allowing us to build robust server-side logic.

Serverless Functions: When we have event-driven or microservices architecture and want to focus on specific tasks or functions, serverless functions are a good choice. AWS Lambda, Azure Functions, and Google Cloud Functions are popular serverless platforms. They automatically scale and manage infrastructure, allowing us to run code in response to events without worrying about server management.

WebSockets: For real-time, bidirectional communication between the server and client, we can use WebSockets. Libraries like Socket.IO (Node.js) or SignalR (.NET) provide abstractions and utilities to work with WebSockets. This is useful for applications that require instant updates, like chat applications or live dashboards.

**-Code below is supposed to print [{name: "Tom", id: 1}, {name: "Kate", id: 2}]. Explain why it doesn't and explain how would you fix it.**

The code provided does not print the expected output because the getId() function is not bound to the correct context. When destructuring getId from new IdGenerator(), it loses its reference to the IdGenerator instance, resulting in this inside getId being undefined.

class IdGenerator {

lastId = 0;

getId() {

return this.lastId++;

}

}

const idGenerator = new IdGenerator(); // Create an instance of IdGenerator

const { getId } = idGenerator; // Destructure the getId method from the instance

const people = ["Tom", "Kate"].map((name) => ({ name, id: getId.bind(idGenerator)() }));

console.log(people);