Software Engineering - Internship Assignment

1. Design patterns are used by experienced object-oriented software developers as the best practices. These design patterns are the solution to the problems faced by the developer during software development. These solutions obtain trial and error by numerous software developers over quite a substantial period.

   Usage of design pattern

   - Common platforms for developers - the design pattern provides a standard terminology and are specific to a particular scenario. For example, a singleton design pattern signifies the use of the single object therefore all developers are familiar with the single pattern and make use of the single object and they can tell each other that the program is following a singleton.

   - Best practices – the design patterns have evolved over long periods and they provide the best solutions to some problems faced during software development. Learning these patterns helps inexperienced developers to learn software design quickly and easily.

   Types of design pattern

   - Creational patterns – these patterns provide the way to create the objects while hiding the creation logic, rather than instantiating objects directly using a new operator. This gives the program more flexibility in deciding which objects need to be created for a given use case.

   - Structural patterns – normally these design patterns consist of class and object composition. The concept of inheritance is used to compose the interfaces and define ways to compose objects to obtain new functionalities.

   - Behavioral patterns – These design patterns are specifically concerned with communication between objects.

   - J2EE Patterns - These design patterns are specifically concerned with the presentation tier. These patterns are identified by Sun Java Center.

   The singleton design pattern

The singleton pattern only allows a class or the object to have a single instance and it uses a global variable to store that instance. You can use lazy loading to create sure that there's only one instance of the category because it'll only create the category after you need it.

That prevents multiple instances from being active at identical times which could cause weird bugs. Most of the time this gets implemented within the constructor. The goal of the singleton pattern is usually to manage the world state of an application.

An example of a singleton that you simply probably use all the time is your logger.

```
class FoodLogger {
  constructor() {
    this.foodLog = []
  }

  log(order) {
    this.foodLog.push(order.foodItem)
    // do fancy code to send this log somewhere
  }
}

// this is the singleton
class FoodLoggerSingleton {
  constructor() {
    if (!FoodLoggerSingleton.instance) {
      FoodLoggerSingleton.instance = new FoodLogger()
    }
  }

  getFoodLoggerInstance() {
    return FoodLoggerSingleton.instance
  }
}

module.exports = FoodLoggerSingleton
```

2. DTOs or Data Transfer Objects are objects that carry data between processes to cut back the amount of methods calls. The pattern was first introduced by Martin Fowler in his book EAA.

   Fowler explained that the pattern's main purpose is to cut back roundtrips to the server by batching up multiple parameters in an exceedingly single call. This reduces the network overhead in such remote operations.

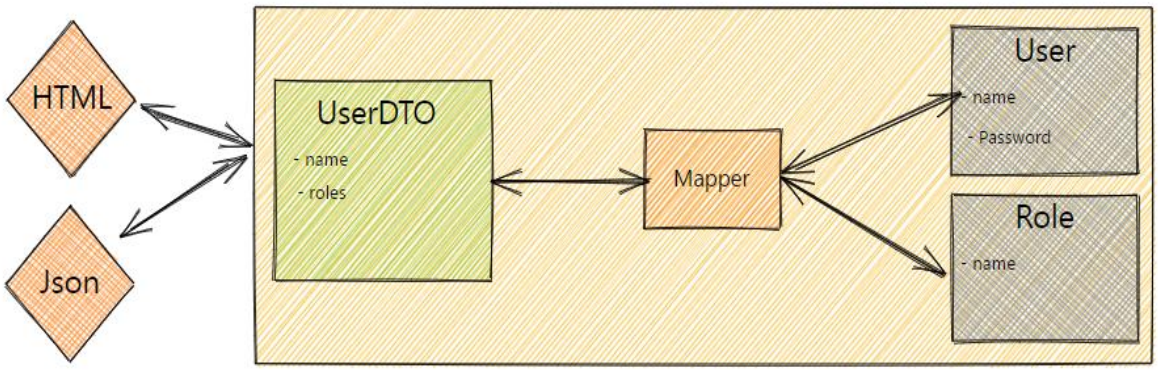
   Another benefit is the encapsulation of the serialization's logic (the mechanism that translates the thing structure and data to a selected format that may be stored and transferred). It provides one point of change within the serialization nuances. It also decouples the domain models from the presentation layer, allowing both to alter independently.

   How to use DTO

   DTOs normally are created as POJOs. they're flat data structures that contain no business logic. They only contain storage, accessors, and eventually, methods associated with serialization or parsing.

   The data is mapped from the domain models to the DTOs, normally through a mapper component within the presentation or facade layer.

   The below image represents the interaction between the components.



3. Just like credit card details or personal information, API credentials are valuable information that must be kept secret

   - Don’t store the API keys directly in your code. - Embedding your API key in your ASCII text file could appear sort of a practical idea, but it’s a security risk as your ASCII text file can find itself on many screens. Instead, store your API key and secret directly in your environment variables. Environment variables are dynamic objects whose values are set outside of the appliance. this can allow you

to access them easily (by using the os. getenv() method in Python, as an example, or using the dotenv package in an exceedingly Node app), and avoid accidentally exposing a key once you push your code.

- Don't store the API key on the client side -If you are developing a web app, remember to always store your credentials on the backend side. Fetch the API results from there and then pass them to the frontend.

- Don't expose unencrypted credentials on code repositories, even private ones

- Consider using an API secret management service - Storing your API credentials as environment variables will prevent lots of pain, but if you're functioning on a project with an enormous team you will find that's it hard to stay everyone in sync. One solution for convenience and peace of mind is to use a secret management service like AWS Secret Manager. this may not only protect your keys but facilitate your retrieving and managing of the credentials of your entire team.

- Generate a new key if you suspect a breach

4. JWT, or JSON Web Token, is an open standard accustomed share security information between two parties — a client and a server. Each JWT contains encoded JSON objects, including a collection of claims. JWTs are signed by employing a cryptographic algorithm to make sure that the claims cannot be altered after the token is issued.

How JWT works

JWTs differ from other web tokens in this they contain a group of claims. Claims are wont to transmit information between two parties. What these claims are counting on the utilization case at hand. as an example, a claim may assert who issued the token, how long it's valid for, or what permissions the client has been granted.

A JWT may be a string made of three parts, separated by dots (.), and serialized using base64. within the commonest serialization format, compact serialization, the JWT looks something like this: aaaa.bbbbb.ccccc.

Once decoded, you'll get two JSON strings:

The header and therefore the payload.
The signature.
The JOSE (JSON Object Signing and Encryption) header contains the sort of token — JWT during this case — and therefore the signing algorithm.

The payload contains the claims. this can be displayed as a JSON string, usually

containing no over a dozen fields to stay the JWT compact. This information is often utilized by the server to verify that the user has permission to perform the action they're requesting.

There aren't any mandatory claims for a JWT, but overlaying standards may make claims mandatory. for instance, when using JWT as a bearer access token under OAuth2.0, iss, sub, aud, and exp must be present. some are more common than others.

The signature ensures that the token hasn’t been altered. The party that makes the JWT signs the header and payload with a secret that's known to both the issuer and receiver, or with a personal key known only to the sender. When the token is employed, the receiving party verifies that the header and payload match the signature.

5. Difference between SQL and NoSQL

SQL	NoSQL
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have a dynamic schema
These databases are not suited for hierarchical data storage.	These databases are best suited for hierarchical data storage.
These databases are best suited for complex queries	These databases are not so good for complex queries
Vertically Scalable	Horizontally scalable
Follows ACID property	Follows CAP(consistency, availability, partition tolerance)
Examples: MySQL, PostgreSQL, Oracle, MS-SQL Server, etc.	Examples: MongoDB, GraphQL, HBase, Neo4j, Cassandra, etc.

6. React components have a built-in state object. The state is encapsulated data where you store assets that are persistent between component renderings. The state is just a fancy term for a JavaScript data structure

Why do we need to react to state management?

React applications are built using components and they manage their state internally it works well for applications with few components, but when the appliance grows bigger, the complexity of managing states shared across components becomes difficult.

The react state management uses react hooks and redux

What is Redux?

Redux is solely a store to store the state of the variables in your app. Redux creates a process and procedures to interact with the shop in order that components won't just update or read the shop randomly. the same as the bank. It doesn't mean because you have got money within the bank that you simply can go anytime, open the vault, and make money. you have got to travel through certain steps to withdraw money.

What is React Hook?

Hooks are the new feature introduced within the React 16.8 version. It allows you to use state and other React features without writing a category. Hooks are the functions that "hook into" React state and lifecycle features from function components. It doesn't work inside classes.