Software Engineering - Internship Assignment

Name : W. M. Paramee Dilanka Weesinghe

Email : parameeweesinghe@gmail.com

Github : https://github.com/ParameeDilanka

Written Questionnaire

1. Explaining what is design pattern and how we can use design patterns in projects

In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Usage in project:

- Common development platform

Design patterns offer a common language and are tailored to a given context. For instance, a singleton design pattern denotes the usage of a single object, thus all developers who are familiar with it will use a single object and be able to identify it in a program.

- Best Practice

Design patterns have been developed over a considerable amount of time and offer the best answers to certain issues that arise during the development of software. Unexperienced engineers can learn software design more quickly and easily by becoming familiar with these patterns.

2. What is DTO and explain the use of it.

An object that transfers data across processes is known as a data transfer object (DTO). can make advantage of this technology to make it easier for two systems to communicate (such as an API and a server) without perhaps disclosing sensitive data.

DTOs will be used in environments that support object-oriented programming (OOP) languages like Python, C++, and Java. DTOs are simple to create and maintain.

OOP environments rely on a "calls" system. Each one requires time and processing speed and is somewhat akin to a data lookup. Calls can also reveal private information you'd prefer to keep secret if you're not careful, such as:

- Employee addresses
- Account numbers
- Social Security numbers
- Business logins

Data-only objects (DTOs) should not have business logic. It's a little, straightforward item that should perform just one job.
A quality DTO will:

- **Cut back on boilerplate**. Will start fresh with each one
- **Be easy to create.** DTOs shouldn't be so complicated that struggle to write them. (Code like this is easy to break.)

- **Be readable.** Anyone should be able to parse code.

3. How are you going to store secrets in an application without exposing it to the internet?

- Don’t store API key directly in code

  Although it may seem reasonable to embed your API key in your source code, doing so poses a security risk because source code can appear on numerous screens. Put your API secret and key directly in environment variables as an alternative. Environment variables are movable, dynamic objects that have their values set externally to the application. This will make it possible to retrieve them quickly and prevent the accidental disclosure of a key while pushing code (by using the dotenv package in a Node project).

- Don’t store API key in client side

  Remember to always save credentials on the backend side when creating a web application. From there, retrieve the API results, and then feed them to the front end.

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

If you're a mobile developer, it's especially crucial to keep your login information outside of your app because even a novice user may quickly reserve engineer your program and locate your credentials. Try to keep your API login information on a different server that you control and use that same server to retrieve API results before sending them to the client.

- Consider using an API secret management service.

However, if you are working on a project with a large team, you may find it difficult to keep everyone in the loop. Storing API credentials as environment variables will save you a lot of headaches. Using a secret management service like AWS Secret Manager is one way to be convenient and worry-free. Not only will this safeguard your keys, it will also make it easier for you to access and manage the team's login information.

- Generate a new key if you suspect a breach.

If API credentials have been compromised, keep calm and simply revoke key. This can easily be done on the Amadeus for Developers portal by following these steps:

1. Log in to your account.
2. Visit your Self-Service Workspace
3. Select the application in question
4. Generate a new key by clicking on the refresh button

4. What is JWT and how does it work?

JWT, or JSON Web Token, is an open standard used to share security information between two parties — a client and a server. Each JWT contains encoded JSON objects, including a set of claims. JWTs are signed using a cryptographic algorithm to ensure that the claims cannot be altered after the token is issued

JWTs are unique among web tokens in that they include a list of claims. Information is sent between two parties through claims. The specific use case will determine what these assertions are. A claim might specify, for instance, who issued the token, how long it is valid, or what authorizations the client has received. Three sections of a string called a JWT are separated by dots (.) and serialized with base64. The JWT appears something like this in compact serialization, the most used serialization format: xxxxx.yyyyy.zzzzz.

You will obtain two JSON strings after decoding:

1. the payload and the header.
2. The signature

The JOSE (JSON Object Signing and Encryption) header contains the type of token — JWT in this case — and the signing algorithm.

The payload contains the claims. This is displayed as a JSON string, usually containing no more than a dozen fields to keep the JWT compact. This information is typically used by the server to verify that the user has permission to perform the action they are requesting. For a JWT, there are no required claims, although any overlaying standards may require claims. For instance, iss, sub, aud, and exp are required when utilizing JWT as the bearer access token in OAuth2.0. Others are more typical than others.

The token can't have been changed, thanks to the signature. With a secret that is known to both the issuer and the receiver, or with a private key that is known only to the sender, the party that creates the JWT signs the header and payload. The receiving party checks that the header and payload match the signature before using the token.

5. What is the difference between SQL and NoSQL databases?

SQL	**NoSQL**
RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS)	Non-relational or distributed database system.
These databases have fixed or static or predefined schema	They have 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	Examples: MongoDB, GraphQL, HBase, Neo4j, Cassandra

6. Suggest a good state management for frontend application and explain why you recommend it.

Redux

Redux is a state container created in 2015. It became wildly popular because:

- When it first started, there was no viable alternative.
- It allowed for the separation of states and acts.
- Simple state connection was made possible by react-redux magic.

Data is stored in a global storage. You dispatch an action that goes to the reducer each time you need to update the store. Depending on the type of operation, the reducer immutably modifies the state. Must use react-redux to subscribe the component to the store updates in order to use Redux with React.

Followings are reasons why I recommend redux as good state management for frontend application

- Usability - With the launch of the official toolkit package, Redux became quite straightforward. The initial state, reducers, and actions are combined to create a slice, which is then created, passed to the store, and accessed in a component via hooks.
- Maintainability - Redux is easy. Deep knowledge is not necessary to comprehend how to improve or fix something.
- Performance - The software engineer with Redux is the main performance factor. Redux is a simple tool with little logic. If you notice that state updates are taking too long, you can follow the recommended practices to speed them up.
- Testability - Redux is ideal for unit testing because it only uses pure functions (actions and reducers). Additionally, it offers a way to create integration tests that demonstrate how the store, actions, and reducers interact.
- Scalability - Redux only has one global state by default, which makes scaling difficult. However, a package called redux-dynamic-modules makes it possible to build middleware and reducers that are modular.
- Modifiability - Because Redux supports middleware, customization is simple.
- Reusability - Redux excels at reuse because it is framework independent.
- Ecosystem - Redux provides a vast ecosystem of functional libraries, tools, and add-ons.
- Community - The most established state management framework in our comparison, Redux, has a sizable community and a substantial knowledge base. On Stack Overflow, there are 30,000 queries with the redux tag, of which 19,000 are answered.
- Pulse - Redux is continuously updated and maintained.

Development task

Github Repo Link: https://github.com/ParameeDilanka/Note_Collection_Web_Application.git

YouTube Demo Video Link: https://youtu.be/zLfMCfSyh58