The questions are deleted for the sake of privacy.

1. It depends on how the JWT is generated, transmitted, and stored.

JWT must be generated in the server with a secret key that must be visible only in the server itself. If the content of the JWT is modified, the server can not be able decode it and therefore rejects the request.

It must be transmitted only over HTTPS in order to avoid some abusers intercepting the transmission and obtain it.

And one other issue with safety is that where the JWT is being stored on the client-side. You can store it in the local storage (LocalStorage API, IndexedDB, etc) or in the cookies but one must consider the safety cautions of each preferred way.

2. The first possible attack that comes to my mind is injecting some JavaScript code that runs on the evaluation of the HTML in the browser that may compromise some valuable information such as JWT. In order to avoid this, you can *sanitize* the HTML content before evaluating and rendering it which means getting rid of any script in it.

As the second, even if the HTML does not contain any script in it, it can submit a form that can contain malicious code and try some SQL injection attack on the server-side.

3.

a. You can have an immutable object by using “Object.freeze()” which freezes an object and prevent any possible change of any of its properties.

```
const a = Object.freeze({
        foo: ‘bar’,
        buzz: ‘fizz’
})

a.foo = ‘foo’ // Throws error
```

b. With immutability, you can have a single source of truth, such as a state-management mechanism. You avoid the risk of unintentional modification of the data. But if the immutable data is a large collection of objects and if you want to modify it (let’s say you want to update the state), you need to create a clone of the whole collection which can cause huge allocations and performance problems.

c. I prefer the way that I create the store (the original data source) in the service as private property. So only the service can modify it. And when some subscribers request the data (for example: subscribing to an observable of the service), the service passes a clone of the data to the observable, subject to be published/replayed.

4. If not already done, I would choose lazy-loading modules to decrease the initial loading time since lazy-loading loads a module only when it is needed.

I would use service workers to cache the application and the static files.

5. I would choose “b” because the hardware is hardware but it is the software that makes you free what you can do with that.