Interactive Digital Notice Board API

Understanding RESTful APIs

In todays world, new software is developed in such ways that applications do not always function in isolation. They usually need to communicate with other systems in order to fetch the data or else send it. RESTful APIs provide a structured way for online applications to be able to exchange the information over the internet in a secure manner. These APIs depend solely on HTTP requests to be able to access and in order data, thus making them crucial for the web development.

The RESTful API systems should follow with the specific design principles. This should ensure that the communication between the client as well as the server remains efficient and predictable. These principles include statelessness, cacheability, a uniform interface, and also a client server architecture.

Key Features of RESTful APIs

- **HTTP Methods:** REST APIs leverage standard HTTP methods— *GET, POST, PUT, PATCH, DELETE* — to perform *CRUD* (Create, Read, Update, Delete) operations.
- **Statelessness:** Each client request must contain all the necessary information, as the server can not store session data.
- **Cacheability:** Some responses are stored in order for them to be reused so that they will be able to improve performance outcome of the product.
- **Uniform Interface:** APIs should consist a consistent structure to further enable easy interaction between different systems.

Client-Server Model

Client server model; this is the foundation of the majority of modern web applications. This allows clients to send the requests to the central servers, which processes and returns the required data.

How the Client-Server Model Works:

1. A client (for example. a browser or mobile app) sends a request using a network-enabled device.
2. The server receives and processes the requested information, often interacting with a database.
3. The server returns a response containing the requested data or the result of the action.

This architecture allows for separation of concerns, meaning the client handles user interactions while the server focuses on data processing and business logic.

Examples of Client-Server Use:

- **E-commerce Platforms:** Online stores including Amazon rely soley on a client-server model to handle numerous customer requests, display products, and process their transactions.
- **Streaming Services:** Platforms such as Netflix and Spotify serve video and audio content from central servers to users worldwide.
- **Online Banking:** Banking applications that use client-server interactions to retrieve account details, process transactions, and to manage user sessions.

Core Principles of REST APIs

1. Statelessness

When it comes to how APIs work, each request is treated separately. The server doesn't keep track of any previous requests — it only looks at the information that comes with the current one. This is known as a stateless design. The benefit of this is that it makes the system easier to scale, since the server isn't wasting resources trying to remember what happened before. It can handle lots of requests at once without slowing down. It also makes the system more reliable because even if one request fails, it doesn't affect the others.

2. Cacheability

Another thing that makes APIs more efficient is caching. When a server sends a response back, the information can be saved for a while so the client does not have to keep asking for the same thing every time. It helps speed up the process and reduces the amount of work the server has to complete. But not all daya should be catched. Some data changes all the time, this data needs to stay updated. For this reason the server usually includes a small instruction telling the client whether it's okay to store the information and for how long. This way, the system stays fast without showing users old or incorrect data.

3. Layered System

Layered system this system basically means that there can be other servers or systems between the client and the main server. Similar to a load balancers or gateways. This helps with things like security and making sure the system can handle multiple requests without slowing down. Like this the client does not need to know or worry about what processes are happening in the background — they just send a request and get a response like normal.

4. Uniform Interface

This API means that APIs are designed in such a way that predicts and follows the same structures, so clients can always know what to expect. When APIs stick to the same basic rules, they makes it easier to combine different services together. Developers don't have to relearn how to work with each service.

How API Requests Work

API communication involves the following three main steps:

1. **A Client Sends the Request:** The clients send an HTTP request as one of the following (GET, POST, PUT, DELETE), then fowarded to the endpoint in the API.
2. **Then the Server Processes the Request:** The server performs requested operation, such as querying a database or executing a command.
3. **Finally the Server Sends a Response:** The server returns the requested data (often in JSON) or an error message if something goes wrong.

Common HTTP Methods in REST APIs

1. GET (Retrieve Data)

GET: This method is used when one wants to fetch or read some data from a server without making any changes. For example, when your browser sends a GET request to the server that tries to get the information. It's safe to use over and over because it doesn't alter anything on the server, it is only asking for data, not changing anything.

Examples:

- Fetching a list of users: `get/users`
- Retrieving product details: `get/products/123`

2. POST (Create Data)

POST is used when you want to send some data to the server to create something new. When you fill out and sign a form or post a comment on a blog. You send data to the server, and it creates a new entry with that information.

Examples:

- Creating a new user: `post/users`
- Submitting a contact form: `post/messages`

3. PUT (Update the Data)

PUT is used when you want to update an entire resource on the server. It replaces the existing data with a new one.

Examples:

- Updating user details: `put/users/123`
- Changing a blog post: `put/posts/45`

4. PATCH (Partial Update)

PATCH is similar to PUT, but it's more specific. It's used when you want to update just a small part, rather than replacing the whole thing. So if you wanted to change just your email address for example, you'd just use PATCH as it is quicker and more efficient.

Example:

- Updating a user's email: `patch/users/123`

5. DELETE (Remove Data)

It is used when you want to remove something from the server completely. For example, if you delete a file or close your account on a website, DELETE is the method that takes care of permanently removing that data.

Example:

- Deleting an account: `delete/users/123`

Query Parameters and Request Body

- **Query Parameters:** These are key-value pairs added to a URL to filter the results being displayed on the page.

  Example: `/products?category=electronics&sort=price`.

- **Request Body:** Used in POST and PUT requests to send structured data in JSON or XML format.

API Security with OAuth 2.0

OAuth 2.0 is a widely used authorization framework that allows a secure access to the user resources without exposing their passwords.

Advantages of using OAuth 2.0:

- **Improved Security:** Users do not need to share passwords with other third-party apps.
- **Permissions:** Users can control what data is shared.
- **Token-Based Authentication:** Secure, revocable access tokens that replace passwords.

OAuth 2.0 Workflow:

1. A client requests access to a user's data.
2. The user grants or denies permission.
3. If granted, the server issues an access token.
4. The token is used in API requests for authentication.

OAuth 2.0 in dept:
OAuth 2.0 is a system designed to be used to let apps or websites access certain information from your account, like this the system can access the infomration without needing your password. Instead of giving the app your login details, you can give it permission to access only the data it needs. One has probably seen this when a website asks to Sign in with "Google"— this is an example of OAuth 2.0.

Scopes
When an app asks to access your information, it also tells you exactly what information it will be using. These permissions are called **SCOPES**. For example, an app might only ask to see your email address. Scopes make sure that the app can only do what you allow it to ensuring that nothing more is accessed or reveiled.

Access Token
Once you allow access, the app gets an **access token**. This is basically a special code that lets the app access the data you agreed to share. The token is only valid for a certain amount of time and only works for the specific information you approved. It's much safer than sharing your actual password because the token doesn't give full access to your account.

Client ID & Client Secret
When an application uses OAuth 2.0 it has its own **Client ID** and **Client Secret**. The Client ID is like the app's public name, so the system knows which app is asking for access. The Client Secret is a private key that proves the app is trustworthy. Together, connect the app securely to the server and request the access token.

Security Considerations for APIs

APIs must be protected from security vulnerabilities. The OWASP (The Open Worldwide Application Security Project) API Security that highlights the key threats:

Common API Security Threats:

- **Broken Authentication:** Weak authentication mechanisms that can allow attackers to access user data.
- **Excessive Data Exposure:** APIs should return only the necessary data that prevents leaks.
- **Rate Limiting:** Restricting the number of requests per user prevents abuse and denial-of-service attacks.
- **Injection Attacks:** APIs should validate and sanitize all user input to prevent SQL and script injections.

Security Considerations

The OWASP API Security is an industry-recognized list that consists of the most critical security risks facing APIs. It is published by the Open Worldwide Application Security Project (OWASP) and helps developers and organizations understand and protect their APIs against potential attacks.

The latest OWASP API Security Top 10 includes the following threats:

1. Broken Object Level Authorisation
2. Broken Authentication
3. Broken Object Property Level Authorisation
4. Unrestricted Resource Consumption
5. Broken Function Level Authorisation
6. Unrestricted Access to Sensitive Business Flows
7. Server Side Request Forgery (SSRF)
8. Security Misconfiguration
9. Improper Inventory Management
10. Unsafe Consumption of APIs

For the Interactive Digital Notice Board API, we will address the following key threats:

Broken Authentication:

This occurs when attackers can exploit weak authentication systems to gain unauthorized access.

Excessive Data Exposure:

This happens when APIs return more data than necessary, potentially leaking sensitive information.

Mitigation:

This ensures that API responses only include the required data fields. Data filtering will be implemented based on user roles and access permissions.

Mitigation Strategies:

1. **Implement OAuth 2.0** for secure authentication.
2. **Use HTTPS** to encrypt communication.
3. **Validate and sanitize user input** to prevent injection attacks.
4. **Enforce rate limits** to prevent API abuse.

Mitigating Security Threats in Our API

For the Interactive Digital Notice Board API, we will address the following security concerns:

1. **Broken Authentication**: To mitigate this threat, we will implement OAuth 2.0 for secure and authorised access. We will also enforce strong password policies and

use multi-factor authentication (MFA) to enhance the security of user logins.
2. **Excessive Data Exposure**: We will implement proper data filtering and ensure that only authorised users can access certain types of data. For example, users will only be able to access the notice boards relevant to their organisation, ensuring that sensitive information is not exposed to unauthorised individuals.
By focusing on these key security measures, we aim to develop a secure, efficient, and user-friendly API for our Interactive Digital Notice Board system, ensuring both robust protection against potential vulnerabilities and a seamless experience for users.

Conclusion

The RESTful APIs plays a vital role in web application, thus allowing a seamless communication between systems. By using such practices, such as maintaining a client server architecture, by ensuring interactions and implementing strong security measures like Oauth 2.0, developers can create an effective and secure API.

By understanding the HTTP methods, handling requests and security considerations that ensure APIs remain reliable and safe from vulnerable. The Interactive Digital Notice Board will follow the API principles to provide a more user-friendly system, facilitating secure and effective digital communication.

Referances

Gruhn, V., & Köhler, A. (2012). *Software Architecture (BOOK)*

Microsoft Learn – REST API concepts and principles
https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

REST API Tutorial – What is REST?
https://restfulapi.net/

MDN Web Docs – HTTP Methods
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods