(Part 2)

API Design

Software System Design

Spring 2024 @ Ali Madooei

API Design

API design is the process of creating a set of rules and guidelines for building software applications and services that can communicate and share functionalities.

- **API Design Principles**: Best practices and guidelines for creating APIs that are efficient, scalable, and user-friendly.
- **API Design Patterns**: Reusable solutions to common problems encountered in API design.
- **API Design Tools**: Software tools and platforms that facilitate the design and development of APIs.
- **API Design Documentation**: Comprehensive documentation that provides information about the API’s structure, functionality, and usage.

Types of Web APIs

Web APIs are designed to serve different purposes and use cases, and they can be categorized based on their functionality and intended audience.

- **RESTful APIs**: Follow the principles of Representational State Transfer (REST).
- **GraphQL APIs**: Enable clients to request specific data and reduce over-fetching and under-fetching.
- **RPC APIs**: Use Remote Procedure Calls (RPC) to enable communication between software systems.
- **Other specialized APIs:** Webhooks, WebSockets, and more.

RESTful API Design

REST stands for Representational State Transfer.

- It is an architectural style for designing networked applications.
- RESTful APIs use HTTP requests to perform CRUD operations on resources.
- RESTful APIs are by far the most common type of Web APIs.

Principles of RESTful API Design

RESTful APIs are built around REST principles, providing a stateless, scalable, and cacheable way to interact with web resources.

- **Statelessness:** Each request contains all information needed by the server.
- **Client-Server Architecture:** Client manages UI/UX; server manages resources, operating independently.
- **Uniform Interface:** Ensures consistent and standardized interactions.
- **Cacheable:** Responses can be marked as cacheable to improve performance.
- **Layered System:** Enables scalability through unknown intermediary servers.

Resource Naming in RESTful APIs

RESTful APIs use URIs to identify resources.

- Use nouns to represent resources (e.g., `` `/posts` ``, `` `/users` ``).
- Keep resource names plural (e.g., `` `/posts` `` not `` `/post` ``).
- Use sub-resources for relations (e.g., `` `/posts/1/comments` ``).

```javascript
// Example in Express.js
app.get('/posts', (req, res) => {
  // Logic to retrieve and return posts
});

app.get('/posts/:postId/comments', (req, res) => {
  // Logic to retrieve and return comments for a specific post
});
```

CRUD Operations in RESTful APIs

RESTful APIs rely on standard HTTP methods.

- Create: Use `` `POST` `` to create a new resource.
- Read: Use `` `GET` `` to retrieve a resource.
- Update: Use `` `PUT` `` or `` `PATCH` `` to update a resource.
- Delete: Use `` `DELETE` `` to remove a resource.

CRUD Operations: Activity

1. Design the URI for updating a user’s email in a user resource.

2. Determine the HTTP method and the URI to delete a comment under a specific post.

CRUD Operations: Solution

1. URI: `` `/users/{userId}/email` ``, HTTP Method: `` `PATCH` `` or `` `PUT` ``.

```
// Example in Express.js
app.put('/users/:userId/email', (req, res) => {
  // Logic to update a user's email
});

app.patch('/users/:userId', (req, res) => {
  // the request body will contain the updated email
});
```

2. URI: `` `/posts/{postId}/comments/{commentId}` ``, HTTP Method: `` `DELETE` ``.

```
app.delete('/posts/:postId/comments/:commentId', (req, res) => {
  // Logic to delete a specific post
});
```

Error Handling in RESTful APIs

- Provide clear error messages.
- Use standard HTTP status codes.
- Include additional error details when necessary.

```
// Example in Express.js
app.get('/posts/:postId', (req, res) => {
  // Logic to retrieve a specific post
  if (!postExists) {
    res.status(404).send({
      status: 404
      error: 'Post not found',
      message: 'The requested post does not exist.'
    });
  }
});
```

When to Use RESTful APIs?

- For CRUD (Create, Read, Update, Delete) operations on resources.
- When a stateless architecture is preferred.
- For public APIs with a wide range of clients.
- When caching of responses is beneficial.
- For APIs that require versioning and scalability.

RESTful API: Activity

- Which of the three use cases (University Student Hub App, Boom Virtual Meeting Service, To-Do App) would be best suited for a RESTful API?

RESTful API: Activity Solution

- **To-Do App:** The app would benefit from using a RESTful API, as it requires CRUD operations on tasks and needs a stateless architecture, public accessibility, and caching of responses.
- **University Student Hub App:** Using a RESTful API is okay, as it involves CRUD operations on resources such as "news," but it requires complex queries and real-time updates, which could be better handled by other API types.
- **Boom Virtual Meeting Service:** While a RESTful API could be used for this service, it may be more efficient to use other API types due to requirements such as low-latency communication and bi-directional streaming capabilities, which can be better accommodated by those API types.

RESTful API Design: Best Practices

- Use nouns to represent resources.
- Keep resource names plural.
- Utilize sub-resources for relations.
- Use standard HTTP methods for CRUD operations.
- Provide clear and accurate status codes.
- Include informative error messages.

RESTful API Design: Activity

1. Design the URI for updating the due date of a task in a to-do app.

2. Determine the HTTP method and the URI to delete a task in the to-do app.

RESTful API Design: Solution

1. URI: `` `/tasks/{taskId}/dueDate` ``, HTTP Method: `` `PATCH` `` or `` `PUT` ``.

```
// Example in Express.js
app.put('/tasks/:taskId/dueDate', (req, res) => {
  // Logic to update the due date of a task
});

app.patch('/tasks/:taskId', (req, res) => {
  // the request body will contain the updated due date
});
```

2. URI: `` `/tasks/{taskId}` ``, HTTP Method: `` `DELETE` ``.

```
app.delete('/tasks/:taskId', (req, res) => {
  // Logic to delete a specific task
});
```