**Client-Server Pattern**

The client-server architectural pattern is an essential design principle in software development that divides application logic into two main components: the client and the server. This separation ensures modularity and flexibility, enabling each component to evolve independently. In a web-based game application, the client-server pattern is particularly beneficial as it allows the application to run on multiple operating platforms, including Windows, macOS, Linux, Android, and iOS.

By employing a REST-style API, the server-side logic is abstracted away from the client-side, providing a consistent interface for communication. This API facilitates various operations such as fetching game data, updating user profiles, and managing game sessions. As a result, developers can implement platform-specific features on the client-side without altering the server-side logic. This pattern efficiently solves the problem of cross-platform compatibility and ensures that the application meets the diverse software requirements of different platforms.

**Server Side**

Developing the application from the server side involves creating a robust backend that handles requests from multiple clients, manages game data, authenticates users, and enforces security policies. The server-side communicates with the client-side through REST APIs, which are designed to be stateless, scalable, and platform-independent.

The server provides endpoints for various functionalities, such as retrieving game data, updating user profiles, and handling game sessions. By using annotations like **@Path**, **@GET**, **@POST**, **@RolesAllowed**, and **@Auth**, the server defines routes for API requests and secures them based on user roles. For example, the **@RolesAllowed("ADMIN")** annotation restricts certain actions to users with the admin role, ensuring that only authorized users can perform specific operations. This RESTful approach allows the server to handle a high volume of requests efficiently and scale horizontally by adding more server instances.

**Client Side**

Developing the client-side application for multiple environments, such as web, mobile, and desktop, requires developers to ensure that each client can interact seamlessly with the server. This involves implementing HTTP requests to interact with the REST API endpoints, handling responses, and managing authentication tokens.

**Next Steps for Client-Side Development:**

1. **Adding More Users to the Database:**
   * Implement a registration feature on the client-side where new users can sign up by providing their details.
   * Validate user inputs on the client-side before sending the data to the server's user creation endpoint.
   * Handle server responses to confirm successful registration or display appropriate error messages to the user.
2. **Including Additional Features:**
   * Integrate real-time features such as in-game chat using WebSockets or a similar technology to enhance user interaction.
   * Add social features like friend lists, leaderboards, and achievements to increase user engagement.
   * Introduce new game modes, levels, or challenges to keep the game interesting and retain players.
3. **Hosting on Additional Clients (Xbox and PS4):**
   * Develop client applications specifically for Xbox and PS4, adhering to the respective platforms' development guidelines and API requirements.
   * Utilize platform-specific features such as achievements/trophies, multiplayer capabilities, and voice chat to enhance the gaming experience on these consoles.
   * Conduct extensive testing on both platforms to ensure performance, compatibility, and a seamless user experience.