1.1. Explain three possible features of a web application that require (or, at least, made easier by) a server-side component written in a language such as PHP. Don't just mention the feature, explain in detail what it involves.

Three possible features of a web application that require/made easier by using a server-side component are user authentication, dynamic content generation, and secure payment processing. For user authentication, when a user enters their credentials (username/password), the data must be sent to a server. The server-side script receives this input, hashes the password , and compares it against a record in a database. This is necessary because you cannot store sensitive user credentials or the logic to verify them in client-side code. For dynamic content generation, for example an online shopping site uses a server side component to allow us to use a template file, instead of creating a separate HTML file for every product. When a user requests a specific product, the PHP script queries the database for the item, retrieves the name, price, description, and image URL, and injects that data into the HTML template before sending the final page to the user's browser. For secure payment processing, the application must communicate with a payment gateway's API. This requires a Secret API Key provided by the payment processor where the PHP script sends the transaction details and this secret key to the processor to confirm the charge is legitimate.

1.2. Explain two actions that can be taken to secure a web application. These may be related to user-authentication & authorization, server configuration, codebase, and/or network infrastructure. Don’t just mention the feature, explain in detail what it involves.

Two actions that can be taken to secure a web application are implementing prepared statements and making different roles/permissions to the database. Implementing prepared statements prevents SQL Injection, one of the most common web security vulnerabilities. The application sends the SQL code template to the database first, leaving placeholders (like ?) for the data. The database parses, compiles, and optimizes this plan before seeing any data. By making different roles/permissions to the database this limits the access users have. This limits the damage to the database if the application is compromised. Instead of using a "root" user, you create a specific database user for the web app with restricted permissions. This user is granted only necessary privileges (like SELECT, INSERT, UPDATE) and is explicitly denied administrative powers like DROP TABLE or GRANT, preventing attackers from destroying the database structure.