1.1.

Three possible features would be Account Management, Input Validation, and Dynamic Content Generation. For instance, when it comes to account management, applications need to store usernames and passwords in databases. However, the user couldn't exactly do this in the front-end with javascript without exposing sensitive information, so what we could do instead is very easily have php do password salting/hashing (ie with using sha256) with that password. In this case, the data is always kept secret and php can also do quick validation testing/manage sessions that would make it easy to make a program that gatekeepers based on if you are signed in or not, with session ids. For Input Validation, php and server side languages makes this so much easier by allowing users to essentially run sql code from within php through queries while also having more security with validating text boxes because server-side code can sanitize output as they aren't available in the client so they can't be tampered with. Finally, with Database Content Generation, php has made-commands as I've said to execute SQL queries and also edit html since you can have html code in php, in this way you can make a site dynamic while also not exposing said dynamic code to intruders. For a more in depth example of using PHP to dynamically creating something:

```
$projectsResult = $conn->query("
SELECT p.projectId, p.name, p.description, u.firstName, u.lastName
    FROM projects p
    JOIN projectMembership pm ON p.projectId = pm.projectId
    JOIN users u ON pm.memberId = u.userId
    ORDER BY p.projectId DESC
");
$projects = [];
while ($row = $projectsResult->fetch_assoc()) {
    $projects[$row['projectId']]['name'] = $row['name'];
    $projects[$row['projectId']]['description'] = $row['description'];
    $projects[$row['projectId']]['members'][] = $row['firstName'] . ' ' .
$row['lastName'];
}
```

1.2. The first action one could do to secure a web application is by implementing some sort of server-side password hashing. For instance, no application should store plaintext passwords, so to solve this we can hash and salt these passwords in a cryptic way such as using functions like sha256 (there are many other options). Doing this way encodes the passwords so that bad actors can't just brute force your password. In addition, even if people do break into the database, the original password is never actually stored, instead it gets altered by the php which ensures there cannot be an easy way to retrieve this password. Another option one could use is input validation, for instance using php, we can essentially double check output and sanitize it

before we actually start using those values. For instance, prepared statements like mysqli can help prevent sql injection which is where people type sql code straight into the form. In this way malicious inputs will be treated as purely as data and won't alter the structure of the sql query. By validating inputs, the application can easily protect itself from a variety of attacks while also making sure the program does what it intends.