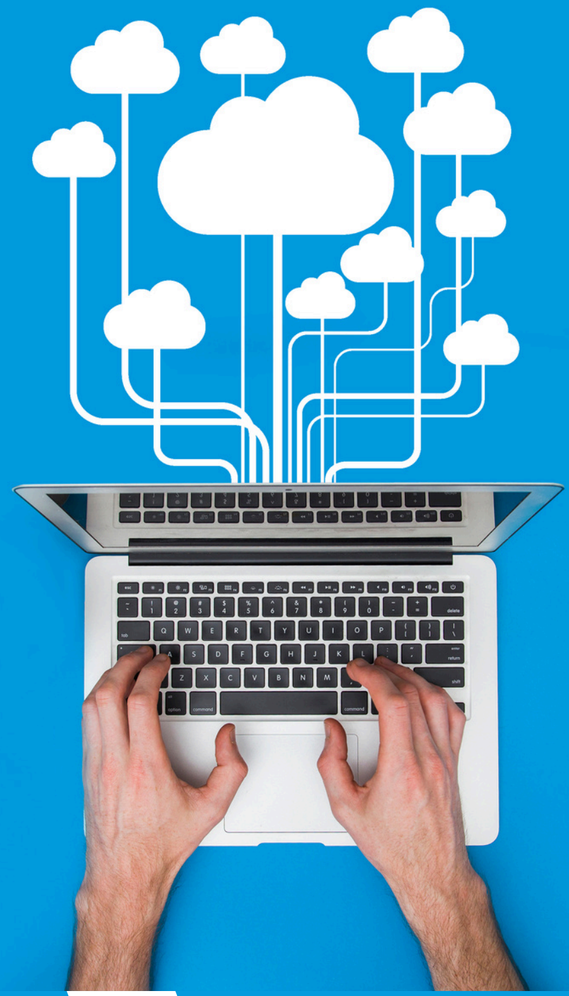
BACK-END BASICS WITH SERVER-SIDE LANGUAGES



Objectives

By the end of this chapter, students should be able to:

- explain the role of the back-end in web applications, how it differs from the front-end, and how the request-response cycle works.
- identify and describe several common server-side languages, understanding their strengths and typical applications.
- write basic server-side code in at least one language to handle common tasks like processing forms, interacting with databases, and generating dynamic content.

Introduction to Back-End Development

- **What is Back-End Development?**
  - The part of web development responsible for processing data, managing server operations, and serving dynamic content to users.
  - It runs behind the scenes, handling business logic, data storage, user authentication, and more.

- **Why is it important?**
  - Ensures web applications are functional, secure, and personalized.
  - Manages interaction with databases, external APIs, and other services.

Introduction to Back-End Development (cont')

- **Front-End vs. Back-End:**
  - Front-End: The user interface, built with HTML, CSS, JavaScript—what users see and interact with.
  - Back-End: The server, application code, database. It processes input, runs logic, and responds.
- **Practical Examples of Back-End Roles:**
  - Handling user login authentication
  - Processing form submissions
  - Serving personalized content based on user data
  - Managing inventory for an e-commerce site

Overview of Server-Side Languages

- **What are server-side languages?**
  - Programming languages that run on the server to generate web content and perform backend tasks.
  - They execute logic, interact with databases, and generate responses such as HTML or JSON.

Overview of Server-Side Languages (cont')

- **Popular server-side languages:**
  - PHP: Easy to embed in HTML; widely used in shared hosting environments.
  - Python: Known for simplicity and readability; frameworks include Django, Flask.
  - Node.js (JavaScript): Allows JavaScript to be used on the server; highly scalable.
  - Ruby: Focused on developer happiness; Ruby on Rails is a popular framework.
  - Java: Enterprise applications; robust; frameworks include Spring.
  - C# (.NET): Used mainly in Windows environments; ASP.NET for web applications.

How Server-Side Languages Work

- **Request-Response Cycle:**
  - Client (browser) sends an HTTP request (e.g., clicking a link or submitting a form).
  - Server receives request, determines which code to run based on URL routing.
  - Code executes, may access a database or perform calculations.
  - Server generates a response: usually HTML, JSON, or files.
  - Response is sent back to the client and rendered by the browser.

How Server-Side Languages Work (cont')

- **Dynamic Content Generation:**
  - Pages that change based on user input, time, or data.
  - Example: A logged-in user's dashboard displaying their data.

- **Interaction with Databases:**
  - CRUD operations: Create, Read, Update, Delete.
  - Server-side code constructs queries to manage stored data.

Key Concepts in Server-Side Programming

- **Routing & Endpoints:**
  - URLs mapped to specific functions or scripts.
  - Example: /login route executes login authentication code.

- **Server-side Logic:**
  - Validating user input
  - Processing transactions
  - Applying business rules

Key Concepts in Server-Side Programming (cont')

- **Session & Cookie Management:**
  - Tracking user sessions (e.g., logged-in status).
  - Cookies store small pieces of data to maintain state.

- **Security Best Practices:**
  - Input validation to prevent SQL injection, XSS
  - Use of HTTPS to secure data transfer
  - Proper authentication and authorization methods

Basic Workflow of a Server-Side Application

- **Step-by-step example:**
    - User enters login credentials and submits form.
    - Browser sends POST request to server at /login.
    - Server routes request to login handler code.
    - Backend validates username/password, queries database.
    - If successful, server creates session, sets cookie.
    - Server responds with a redirect or dashboard page.
    - User now authenticated; subsequent requests maintain session.

Sample Code Snippets

- **The code to create a simple web server that responds with "Hello, World!" when visiting the root URL.**
- **Node.js with Express (JavaScript):**

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
    res.send('Hello, World!');
});

app.listen(3000, () => console.log('Server listening on port 3000'));
```

Sample Code Snippets (cont')

- **PHP:**

```
<?phpecho "Hello, World!";
?>
```

Sample Code Snippets (cont')

- **Python with Flask:**

```
from flask import Flask
app = Flask(__name__)

@app.route('/')def home():
    return "Hello, World!"if __name__

if __name__ == "__main__":
    app.run(debug=True)
```

Sample Code Snippets (cont')

- Sample JavaScript code snippets demonstrating how to connect to a database using Ajax (fetch API) from a frontend webpage to interact with a Node.js backend.
- Fetch and Display Users:

```
function fetchUsers() {
  fetch('/users')  // Backend endpoint returning list of users
    .then(response => response.text()) // Expecting HTML or JSON
    .then(data => {
      document.getElementById('userListContainer').innerHTML = data;
    })
    .catch(error => console.error('Error fetching users:', error));
}

// Call fetchUsers() on page loadwindow.onload = fetchUsers;
```

Sample Code Snippets (cont')

- Add New User

```
function addUser(name) {
  fetch('/add_user', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: new URLSearchParams({ 'name': name }) // Send data as form URL encoded
  })
  .then(response => response.text()) // Response as text
  .then(message => {
    alert(message);
    fetchUsers(); // Refresh list
  })
  .catch(error => console.error('Error adding user:', error));
}

// Example: call addUser('Will Smith') when needed
```

Sample Code Snippets (cont')

- Delete a User

```
function deleteUser(userId) {
  fetch(`/delete_user/${userId}`)
    .then(response => response.text())
    .then(msg => {
      alert(msg);
      fetchUsers(); // Refresh list after deletion
    })
    .catch(error => console.error('Error deleting user:', error));
}

// Example: deleteUser(3)
```

Sample Code Snippets (cont')

- Update User Name

```
function updateUser(userId, newName) {
  fetch(`/update_user/${userId}`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: new URLSearchParams({ 'name': newName }) // Send new name
  })
  .then(response => response.text())
  .then(msg => {
    alert(msg);
    fetchUsers(); // Refresh list
  })
  .catch(error => console.error('Error updating user:', error));
}

// Example: updateUser(2, 'Jane Smith')
```

Sample Code Snippets (cont')

- Javascript file (script.js)

```
// Load and display users when page loadswindow.onload = fetchUsers;

// Function to fetch user data from backend and display
function fetchUsers() {
  fetch('/users')  // Assumes your backend has this route
    .then(response => response.text())
    .then(data => {
      document.getElementById('userListContainer').innerHTML = data;
    })
    .catch(error => console.error('Error fetching users:', error));
}
```

Sample Code Snippets (cont')

- Javascript file (script.js)

```
// Function to add a new user
function addUser() {
  const nameInput = document.getElementById('newUserName');
  const name = nameInput.value.trim();
  if (!name) {
    alert('Please enter a name');
    return;
  }
```

Sample Code Snippets (cont')

- Javascript file (script.js)

```
fetch('/add_user', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded'
    },
    body: new URLSearchParams({ 'name': name })
  })
  .then(response => response.text())
  .then(msg => {
    alert(msg);
    fetchUsers(); // refresh list
    nameInput.value = ''; // clear input
  })
  .catch(error => console.error('Error adding user:', error));
}
```

Sample Code Snippets (cont')

- HTML file:

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8" />
<title>Database Connection Example</title>
</head>
<body>
<h1>User List with Database Connection</h1>
<!-- Container for user list -->
<div id="userListContainer"></div>
<!-- Simple form to add new user -->
<h2>Add New User</h2>
<input type="text" id="newUserName" placeholder="Enter name" />
<button onclick="addUser()">Add User</button>
<!-- Include the JavaScript just before closing body tag -->
<script src="script.js"></script>
</body>
</html>
```

Tools and Frameworks

- **Frameworks:**
  - Express.js (Node.js): Minimalist, fast, and flexible web framework.
  - Django (Python): Full-featured, batteries-included, good for complex sites.
  - Flask (Python): Lightweight, flexible micro-framework.
  - Laravel (PHP): Elegant syntax, built-in ORM (Eloquent).
  - Spring Boot (Java): Rapid development of Java-based web apps.
  - ASP.NET Core (C#): Modern, cross-platform framework.

Tools and Frameworks

- **Development Environment:**
  - Popular IDEs: Visual Studio Code, PyCharm, IntelliJ IDEA, Visual Studio.
  - Local servers: XAMPP, WAMP, MAMP for PHP; built-in servers for Flask, Django, Node.js.
  - Database management tools: phpMyAdmin, PgAdmin, MySQL Workbench.
- **Version Control:**
  - Git, GitHub, GitLab for collaboration and code management.

Best Practices

- **Code Organization:**
  - Follow MVC architecture.
  - Separate routes, logic, and views.
  - Use modules and packages to keep code maintainable.

Best Practices (cont')

- **Security:**
  - Always validate and sanitize user input.
  - Use prepared statements or ORM to prevent SQL injection.
  - Implement authentication (e.g., sessions, JWT).
  - Use HTTPS for secure data transmission.
  - Manage user sessions securely.
  - Protect against Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF).

Best Practices (cont')

- **Performance Optimization:**
    - Use caching mechanisms (Redis, Memcached).
    - Optimize database queries.
    - Minimize server load through efficient code.
    - Use Content Delivery Networks (CDNs) for static assets.