Reproducible Learning Plan for an Undergraduate Full-Stack Web Development Course

Author: Dr. Abraham N. Aldaco-Gastelum
Affiliation: Department of Computer Science / Software Engineering
University of Wisconsin Platteville
Contact: aldacoa@uwplatt.edu

1. Course Overview

This document presents a **reproducible and transferable learning plan** for an undergraduate full-stack web development course. It is derived from multiple course offerings and reflects iterative refinement of both content and pedagogy. The plan is intentionally **semester-agnostic** and **institution-agnostic**, allowing instructors to adapt it to different academic calendars, class sizes, and program structures.

The course emphasizes **learning by doing**, **incremental competence development**, **team-based collaboration**, and **industry-aligned practices**. Rather than focusing on specific tools or vendors, the plan highlights conceptual dependencies, pacing, and instructional strategies that support student success in applied web development contexts.

2. Target Audience and Prerequisites

The course is designed for **junior- or senior-level undergraduate students** in Computer Science, Software Engineering, or related disciplines.

Expected prerequisites

- Introductory programming experience (e.g., Python, Java, or JavaScript)
- Basic understanding of variables, control structures, and functions
- Familiarity with fundamental computing concepts

No prior web development experience is assumed. Students with prior exposure to HTML, CSS, or JavaScript are accommodated through differentiated pacing and asynchronous learning resources.

3. Learning Objectives

By the end of the course, students should be able to:

1. Design and implement interactive client-side web applications.
2. Develop server-side applications that expose RESTful APIs.
3. Integrate frontend, backend, and database components into a cohesive system.
4. Apply asynchronous programming and data-driven techniques.
5. Collaborate effectively in small development teams using version control.
6. Deploy web applications using cloud-based services.
7. Apply foundational security concepts such as authentication and protected routes.

4. Curriculum Phases and Topic Sequencing

The curriculum is organized into **conceptual phases**, each spanning approximately two to three weeks. This phased structure reflects dependencies between topics and supports progressive competence development.

Phase 1: Web Foundations (≈ 3 weeks)

Focus: HTML, CSS, and basic JavaScript

- Structure and semantics of web pages
- Styling and layout fundamentals
- JavaScript syntax and basic interaction

This phase is intentionally compressed. Students are encouraged to engage with asynchronous learning resources to accommodate varying levels of prior exposure.

Phase 2: Core JavaScript Concepts (≈ 3 weeks)

Focus: Dynamic and data-driven client-side programming

- Document Object Model (DOM) manipulation
- Event handling and user interaction
- Higher-order functions (map, filter)
- JSON data format
- Asynchronous programming (callbacks, promises, async/await)

These concepts form the foundation for modern frontend and backend development.

Phase 3: Frontend Frameworks (≈ 3 weeks)

Focus: Component-based application development

- Components and state management
- Application structure and modularity
- Routing and navigation

A modern JavaScript framework (e.g., React) is used as an example to illustrate scalable frontend design principles.

Phase 4: Backend Development and Data Persistence (≈ 2 weeks)

Focus: Server-side logic and data storage

- Server-side application development
- RESTful API design
- Data exchange using JSON

- Persistent storage using databases

Cloud-hosted database services may be used to reduce setup complexity and introduce students to contemporary deployment models.

Phase 5: Full-Stack Integration and Deployment (≈ 2 weeks)

Focus: End-to-end system development

- CRUD operations across the stack
- Client–server communication
- Deployment of frontend and backend components

Students develop complete applications that integrate all previously introduced concepts.

Phase 6: Security and Course Integration (≈ 2 weeks)

Focus: Secure development practices and synthesis

- Authentication and authorization concepts
- Protected routes and APIs
- Introduction to browser extensions
- Course review and project integration

Security topics are introduced after students understand the full system context.

5. Learning Activities Model

Instruction is centered on **guided in-class activities** where students implement concepts in real time under instructor supervision. These activities emphasize practice, experimentation, and immediate feedback.

- Activities are typically completed during class
- Additional time is allowed for completion without penalty
- Late submissions are permitted with modest penalties

This structure prioritizes competence development over time-based performance.

6. Assessment Structure and Orchestration

Assessment is designed to align with concept readiness and progressive integration.

- Short assignments reinforce recent topics
- Project proposals encourage early ideation
- Implementation phases overlap with proposal refinement

Major assessments include: - Multiple individual or team-based assignments - A midterm project (team-based) - A comprehensive final project (team-based)

7. Team-Based Learning and Mentoring

Students complete major assignments and projects in **teams of two**. This structure supports:

- Division of responsibilities
- Collaborative problem solving
- Exposure to professional development practices

Mentoring is provided through instructor and teaching assistant office hours. Team dynamics are monitored, and interventions are applied when necessary to support successful collaboration.

8. Technology Stack (Examples)

Specific technologies may vary by institution. Example categories include:

- Frontend: Modern JavaScript frameworks
- Backend: Server-side JavaScript platforms
- Databases: Cloud-hosted document or relational databases
- Deployment: Cloud-based hosting services

These examples illustrate contemporary practices rather than prescriptive requirements.

9. Adaptation Guidelines

Instructors adopting this learning plan are encouraged to:

- Adjust pacing to fit local calendars
- Substitute equivalent technologies
- Scale mentoring support based on enrollment
- Modify assessment weights while preserving structure

The core pedagogical principles—learning by doing, incremental integration, teamwork, and authenticity—should remain intact.

10. Notes on Reproducibility

This learning plan is intended as a **template** rather than a fixed syllabus. Detailed weekly schedules, assignment descriptions, and implementation artifacts can be derived locally while preserving the conceptual structure presented here.