MIT Art, Design and Technology University
MIT School of Computing, Pune

Department of Information Technology

Lab File

Practical – Full-Stack Development

Class - S.Y. (SEM-VII) SMAD

Name of the Practical Coordinator

Prof. Rohini Bhosale

Prepared By: Mustafa Bhewala

A.Y. 2025 – 2026 (SEM-VII)

Assignment-1: **Create a Static Homepage Layout Using Bootstrap**

- Aim:
  - Use Bootstrap to create a responsive homepage.

- Theory:

  - Mobile-first and responsive design: Bootstrap's CSS is designed to scale from small to large devices. Start with basic mobile styles, then layer on changes at larger breakpoints using responsive utilities (e.g., d-sm-flex, col-lg-4)

  - Container, rows, and columns: Use .container (fixed-width) or .container-fluid (full-width) to center and pad content. Inside, .row creates a horizontal group of columns, and .col-- creates proportional columns. This ensures a consistent structure across device sizes.

  - Spacing and typography: Use utility classes (mt-, mb-, py-, px-) to adjust spacing without custom CSS. Headings, display-* classes, and text-muted provide hierarchy and readability.

  - Hero/"jumbotron" patterns: In Bootstrap 5, jumbotron has been replaced with simple utility patterns. A hero section can be composed using a padded section (py-5), a light background (bg-light), centered text, and button groups. This flexibility avoids rigid component constraints.

  - Buttons and CTAs: Choose semantic CTA placements for key actions (e.g., Browse Courses, Sign Up). Styles like btn-primary and btn-outline-secondary create contrast and clear affordances. Keep accessible labels and adequate touch targets.

  - Images and responsiveness: Use img-fluid to ensure images adapt to their container. For backgrounds, combine responsive spacing and contrast-aware overlay text.

  - Accessibility (A11y): Ensure semantic HTML, appropriate heading order, sufficient color contrast, and keyboard-focusable controls. Include descriptive alt text when using images.

  - Performance: Prefer CDNs for Bootstrap and icons during development. When optimizing, consider self-hosting and subresource integrity (SRI), minification, and deferring non-critical scripts. Minimize large hero images to reduce LCP (Largest Contentful Paint).

  - Integration with EJS and partials: In OLP, share navigation and footers via EJS partials (e.g., include('partials/navbar')). Pass a page title via locals to keep consistent metadata and branding.

  - Visual hierarchy and layout rhythm: Group content into clear sections: hero, features/benefits, testimonials, and CTA. Limit the number of competing CTAs. Use consistent spacing rhythm (e.g., multiples of 8px) for visual harmony.

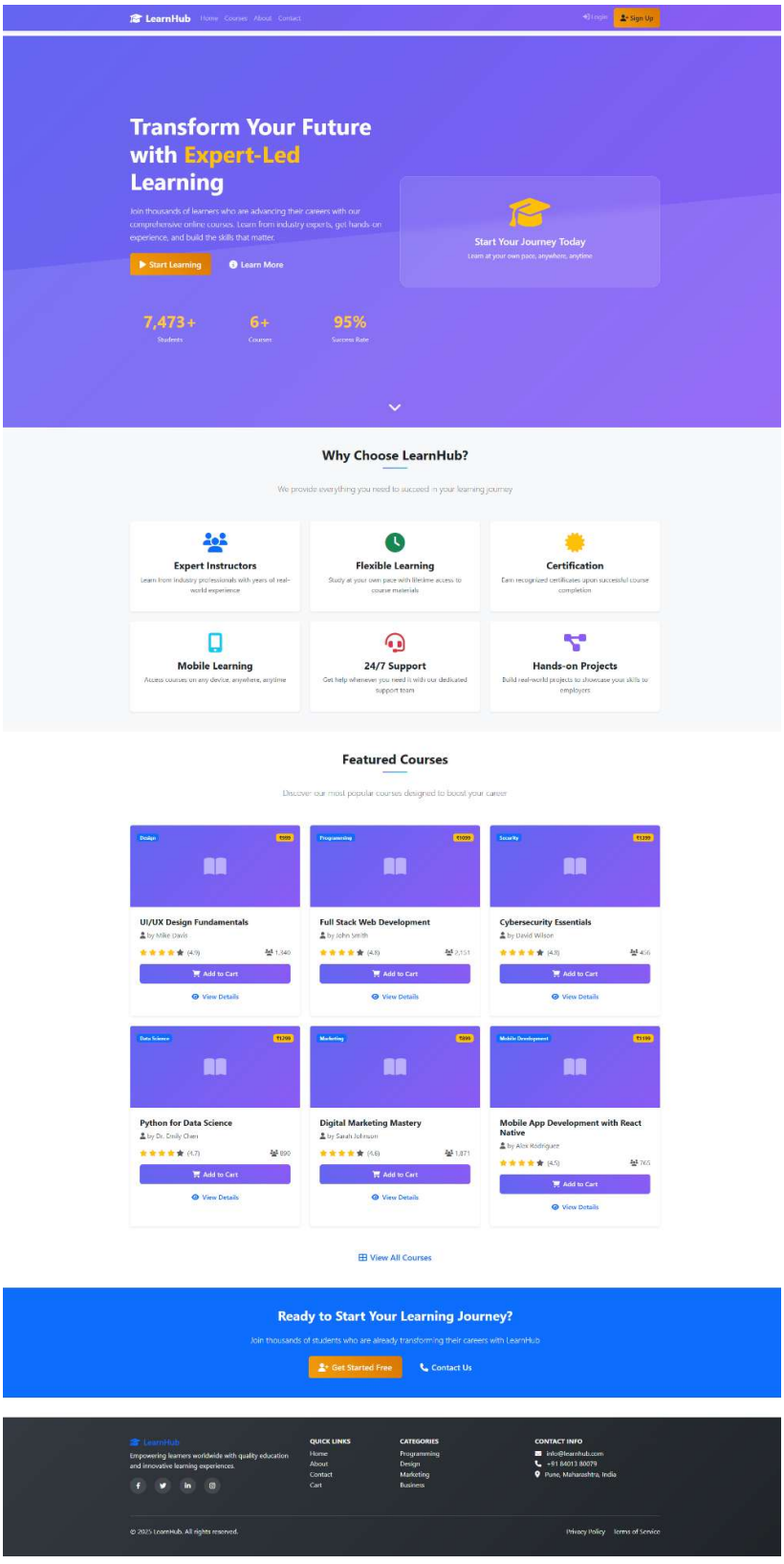
  - Theming and brand consistency: Bootstrap's CSS variables allow quick theme adjustments (primary color, font sizes). Keep a style.css for project-specific tweaks while relying primarily on utilities to reduce custom CSS complexity.

  - Outcome rationale:
    - A Bootstrap-driven homepage provides a predictable, accessible, and maintainable foundation that communicates the OLP's value proposition immediately. It balances development speed with design consistency, lays out a clear conversion path (browse courses, sign up), and establishes a visual system you can reuse across the platform.

- Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title><%= title || 'Online Learning Platform' %></title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
  <link rel="stylesheet" href="/css/style.css">
</head>
<body>
  <%- include('partials/navbar') %>

  <header class="bg-light py-5">
    <div class="container text-center">
      <h1 class="display-5 fw-bold">Learn Anything, Anytime</h1>
      <p class="lead text-muted">High-quality courses to boost your skills.</p>
      <div class="d-grid d-sm-flex gap-2 justify-content-center">
        <a href="/courses" class="btn btn-primary btn-lg">Browse Courses</a>
        <a href="/auth/register" class="btn btn-outline-secondary btn-lg">Get Started</a>
      </div>
    </div>
  </header>

  <section class="container py-5">
    <div class="row g-4">
      <div class="col-md-4">
        <div class="p-4 border rounded text-center h-100">
          <h3>Expert Instructors</h3>
          <p>Learn from professionals with real-world experience.</p>
        </div>
      </div>
      <div class="col-md-4">
        <div class="p-4 border rounded text-center h-100">
          <h3>Flexible Learning</h3>
          <p>Study at your own pace, on any device.</p>
        </div>
      </div>
      <div class="col-md-4">
        <div class="p-4 border rounded text-center h-100">
          <h3>Certificates</h3>
          <p>Earn certificates to showcase your skills.</p>
        </div>
      </div>
    </div>
  </section>

  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script
>
</body>
</html>
```

- Screenshot/Output:



- **Conclusion:** Built a responsive static homepage with Bootstrap components.

Assignment-2: **Design a Grid-Based Information Page**

- Aim:
    - Use Bootstrap Grid to layout course cards.
- Theory:
    - Grid anatomy: The grid is composed of containers, rows, and columns. Columns are flex-based and wrap automatically. Breakpoint-specific classes (col-sm-6, col-lg-4) control how many columns display per row at various viewport widths, ensuring a scalable layout from mobile to desktop.
    - Gutters and spacing: Bootstrap controls inter-column spacing via CSS variables and .g-* utilities. Use .g-3/.g-4 for consistent gaps between cards, improving readability and scannability.
    - Cards as content blocks: Cards combine images, headings, text, and actions within a unified visual box. With h-100 on cards inside equal-height columns, rows feel balanced despite variable content length.
    - Responsive images and media ratio: Ensure card images don't distort; use object-fit: cover and fixed-height containers when consistent aspect ratios are required. For purely responsive scaling, img-fluid is sufficient.
    - Content density and scannability: Keep titles concise, provide short descriptions, and surface a clear "View Details" action. Secondary metadata (price, rating, level, duration) should be visually subdued but consistent.
    - Progressive disclosure: A grid page should entice without overwhelming. The detail page (Assignment 7) handles full information; the grid surfaces the "why care" highlights.
    - Accessibility and semantics: Use ul/li semantically if representing a list, or section/article tags for content blocks. Provide alt text for images and ensure link targets are descriptive.
    - Empty and loading states: Handle the case where the dataset is empty (e.g., "No courses available") and display skeletons/spinners for slow loads to keep perceived performance high.
    - Pagination and filtering (forward-looking): Plan for pagination, sorting, and filters (category, level, price). Reserve UI real estate for these controls without compromising mobile usability. Apply server-side pagination in SSR to reduce payload size.
    - Consistent spacing scale: Harmonize margins and paddings across cards to avoid visual jitter. Keep actions aligned and consistent (e.g., primary action as the rightmost button).
    - SEO and metadata: Use meaningful headings (h1 for page title, h2/h3 for items if appropriate), unique titles, and descriptive text to improve discoverability.
- Code:

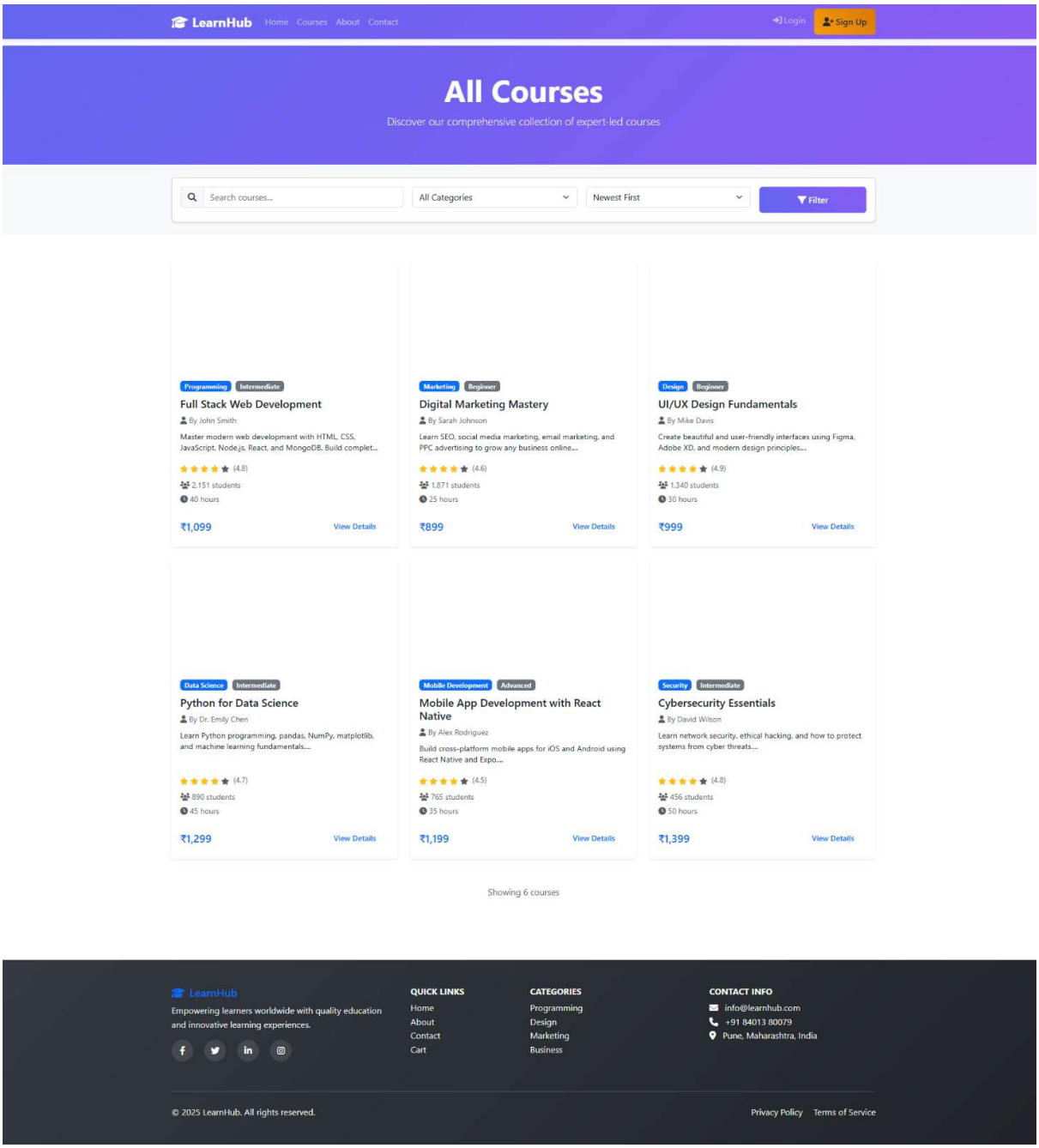
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
  <title><%= title || 'Courses' %></title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
</head>
<body>
  <%- include('partials/navbar') %>
  <main class="container py-5">
    <h1 class="mb-4">Featured Courses</h1>
    <div class="row g-4">
      <% courses.forEach(c => { %>
        <div class="col-12 col-sm-6 col-lg-4">
          <div class="card h-100">
            <img src="<%= c.image || 'https://via.placeholder.com/600x300' %>" class="card-
img-top" alt="<%= c.title %>">
            <div class="card-body">
              <h5 class="card-title"><%= c.title %></h5>
              <p class="card-text text-muted"><%= c.description %></p>
              <a href="/courses/<%= c.id %>" class="btn btn-primary">View Details</a>
            </div>
          </div>
        </div>
      <% }) %>
    </div>
  </main>
</body>
</html>
```

Route to render the grid with mock data:

```
// ...existing code...
const express = require('express');
const router = express.Router();

// In-memory demo data (replace later with DB)
const demoCourses = [
  { id: 1, title: 'JavaScript Fundamentals', description: 'Basics to advanced concepts.' },
  { id: 2, title: 'Node.js with Express', description: 'Build scalable backends.' },
  { id: 3, title: 'EJS Templating', description: 'Server-side rendering patterns.' },
];

router.get('/', (req, res) => {
  res.render('courses-list', { title: 'Courses', courses: demoCourses });
});

module.exports = router;
// ...existing code...
```

- Screenshot/Output:



- **Conclusion:** Created a multi-column responsive grid using Bootstrap.

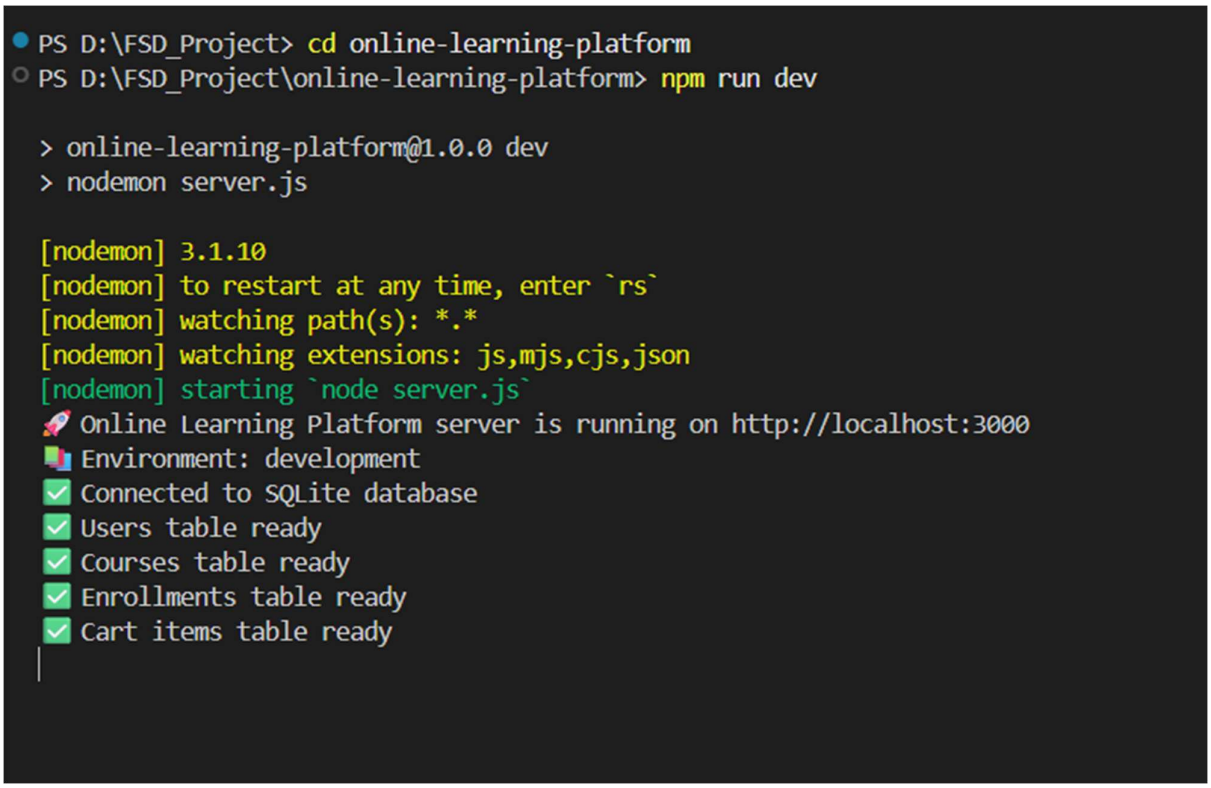
Assignment-3: S**et Up a Node.js Project with Express Server**

- Aim:
    - Initialize npm, install Express, run server with basic routes.
- Theory:
    - npm and package.json: npm init scaffolds package.json, the manifest that declares scripts, dependencies, and metadata. Scripts like start and dev standardize local workflows. Lockfiles ensure deterministic installs.
    - Dependencies vs devDependencies: Runtime libraries (express, ejs, express-session, connect-flash, sqlite3) belong in dependencies, while tooling (nodemon, eslint, prettier) typically belongs in devDependencies.
    - Express application lifecycle: An Express app is a function with a middleware pipeline. Requests traverse middleware and route handlers; responses terminate the chain. Error-handling middleware (four arguments) capture thrown or passed errors.
    - Routing: app.get/post/put/delete define route handlers. Using express.Router enables modular route files that map URL paths to controllers, keeping server.js lean.
    - Template engines and SSR: EJS renders server-side templates (views) with dynamic data. Views reside in a views directory; variables are passed via res.render. This pairs well with SEO needs and quick initial page loads.
    - Static assets and body parsing: express.static serves public assets; body-parser (or Express's built-in json/urlencoded parsers) decode request bodies for form submissions and JSON APIs.
    - Configuration and environment: Use process.env for PORT, DB paths, session secrets. Consider dotenv for local development and environment segregation (development, test, production).
    - Project structure: Separate concerns into config (database, middleware), routes (feature modules), views (EJS), public (CSS/JS/images), and models/data access layers when complexity grows.
    - Error handling and 404s: Centralize notFoundHandler and globalErrorHandler to provide consistent UX for missing pages and server errors. Log errors with sufficient context without leaking sensitive details.
    - Session and flash integration: express-session maintains per-user state; connect-flash provides single-use messages across redirects (e.g., login errors, success messages).
    - Security baselines: Set secure cookies when behind HTTPS in production, use Helmet for basic hardening, and validate inputs to prevent injection issues.
    - Outcome rationale:
        - A cleanly initialized Express project accelerates iteration, provides predictable server behavior, and establishes patterns (routing, views, sessions) that the rest of the OLP builds upon.
- Code:

```
cd D:\FSD_Project\online-learning-platform
npm init -y
npm install express ejs body-parser express-session connect-flash
npm start

const express = require('express');
const router = express.Router();
router.get('/', (_req, res) => res.render('about', { title: 'About' }));
module.exports = router;

const express = require('express');
const path = require('path');
const bodyParser = require('body-parser');
const session = require('express-session');
const flash = require('connect-flash');

// Import database
const database = require('./config/database');

// Import middleware
const { globalErrorHandler, notFoundHandler } = require('./config/middleware');

// Import route modules
const homeRoutes = require('./routes/home');
const aboutRoutes = require('./routes/about');
const contactRoutes = require('./routes/contact');
const cartRoutes = require('./routes/cart');
const authRoutes = require('./routes/auth');
const coursesRoutes = require('./routes/courses');

const app = express();
const PORT = process.env.PORT || 3000;

// Set view engine
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));

// Session configuration
app.use(session({
    secret: 'learning-platform-secret-key-2024',
    resave: false,
    saveUninitialized: false,
    cookie: {
        secure: false, // Set to true in production with HTTPS
        maxAge: 24 * 60 * 60 * 1000 // 24 hours
    }
}));

// Flash messages
app.use(flash());

// Global variables for templates
```

```
app.use((req, res, next) => {
    res.locals.user = req.session.user || null;
    res.locals.success_msg = req.flash('success_msg');
    res.locals.error_msg = req.flash('error_msg');
    res.locals.error = req.flash('error');
    next();
});

// Routes
app.use('/', homeRoutes);
app.use('/about', aboutRoutes);
app.use('/contact', contactRoutes);
app.use('/cart', cartRoutes);
app.use('/auth', authRoutes);
app.use('/courses', coursesRoutes);

// 404 handler
app.use(notFoundHandler);

// Global error handler
app.use(globalErrorHandler);

// Start server
app.listen(PORT, () => {
    console.log(`🚀 Online Learning Platform server is running on http://localhost:${PORT}`);
    console.log(`📚 Environment: ${process.env.NODE_ENV || 'development'}`);
});
```

- Screenshot/Output:



- **Conclusion:** Server set up and accessible across core pages.1

Assignment-4: **Implement Modular Routing and Middleware Logging**

- Aim:
  - Split routes and add a request logger middleware.

- Theory:
  - Separation of concerns: Splitting routes into modules (home, about, contact, auth, courses, cart) keeps server.js focused on orchestration. Each module can encapsulate its handlers, validation, and sub-middleware.

  - express.Router: A mini-app that supports its own middleware and routes. Mount routers at path prefixes (e.g., app.use('/courses', coursesRoutes)) to define clear ownership of URL spaces.

  - Middleware types:
    - Application-level middleware runs for every request (e.g., requestLogger).
    - Router-level middleware runs only for matching paths (e.g., auth checks for /cart).
    - Error-handling middleware has four parameters (err, req, res, next).

  - Request lifecycle and logging: A logger captures method, URL, status, response time, and timestamp. This aids incident analysis, performance tuning, and audit trails. Libraries like morgan simplify this, but custom loggers let you add app-specific context (e.g., user id from session).

  - Correlation IDs: Assign a unique request ID (e.g., via a header or generated UUID) to trace a request across logs and external services, invaluable in asynchronous, multi-service contexts.

  - Privacy and security: Avoid logging secrets or PII. Mask tokens and credentials. Ensure logs follow retention policies and comply with regulations.

  - Performance and overhead: Logging is I/O-heavy. Buffer logs or use asynchronous transports when volume grows. In development, console logging is fine; in production, use structured logs (JSON) and a collector (e.g., ELK, Azure App Insights).

  - Testing and diagnostics: With modular routes, you can unit test each router in isolation using supertest, verifying status codes and bodies without starting the full server.

  - Error propagation: next(err) forwards errors to centralized handlers, preventing duplicated try/catch and ensuring consistent error formatting.

  - Outcome rationale:
    - Modular routers reduce coupling and improve clarity. Request logging provides the telemetry necessary to maintain reliability and quickly diagnose issues in the OLP.

- Code:

```
// Authentication middleware
const requireAuth = (req, res, next) => {
   if (req.session.user) {
      next();
   } else {
      req.session.returnTo = req.originalUrl;
      req.flash('error_msg', 'Please log in to access this page');
      res.redirect('/auth/login');
```

```
    }
};

// Redirect if already logged in
const redirectIfLoggedIn = (req, res, next) => {
    if (req.session.user) {
        return res.redirect('/');
    }
    next();
};

// Check if user is admin
const requireAdmin = (req, res, next) => {
    if (req.session.user && req.session.user.role === 'admin') {
        next();
    } else {
        req.flash('error_msg', 'Access denied. Admin privileges required.');
        res.redirect('/');
    }
};

// Sanitize user input
const sanitizeInput = (str) => {
    if (typeof str !== 'string') return str;
    return str.trim().replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi, '');
};

// Format currency
const formatCurrency = (amount) => {
    return new Intl.NumberFormat('en-IN', {
        style: 'currency',
        currency: 'INR',
        minimumFractionDigits: 0,
        maximumFractionDigits: 0
    }).format(amount);
};

// Format date
const formatDate = (date) => {
    return new Date(date).toLocaleDateString('en-IN', {
        year: 'numeric',
        month: 'long',
        day: 'numeric'
    });
};

// Generate random ID
const generateId = () => {
    return Math.random().toString(36).substr(2, 9);
};

// Validate email
const isValidEmail = (email) => {
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return emailRegex.test(email);
};
```

```
// Validate password strength
const isStrongPassword = (password) => {
    // At least 6 characters, 1 uppercase, 1 lowercase, 1 number
    const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d@$!%*?&]{6,}$/;
    return passwordRegex.test(password);
};

// Error handler for async routes
const asyncHandler = (fn) => {
    return (req, res, next) => {
        Promise.resolve(fn(req, res, next)).catch(next);
    };
};

// Global error handler
const globalErrorHandler = (err, req, res, next) => {
    console.error('Global error:', err);

    // Database errors
    if (err.code === 'SQLITE_CONSTRAINT') {
        req.flash('error_msg', 'Data constraint violation. Please check your input.');
        return res.redirect('back');
    }

    // Validation errors
    if (err.name === 'ValidationError') {
        req.flash('error_msg', 'Please correct the validation errors and try again.');
        return res.redirect('back');
    }

    // Default error
    if (req.xhr) {
        res.status(500).json({ error: 'Internal server error' });
    } else {
        res.status(500).render('error', {
            title: '500 - Server Error',
            message: 'Something went wrong on our end.'
        });
    }
};

// 404 handler
const notFoundHandler = (req, res) => {
    res.status(404).render('404', {
        title: '404 - Page Not Found',
        message: 'The page you are looking for does not exist.'
    });
};

module.exports = {
    requireAuth,
    redirectIfLoggedIn,
    requireAdmin,
    sanitizeInput,
    formatCurrency,
    formatDate,
    generateId,
```

```
    isValidEmail,
    isStrongPassword,
    asyncHandler,
    globalErrorHandler,
    notFoundHandler
};

const express = require('express');
const bcrypt = require('bcryptjs');
const { body, validationResult } = require('express-validator');
const database = require('../config/database');
const router = express.Router();

// Middleware to redirect logged-in users
const redirectIfLoggedIn = (req, res, next) => {
    if (req.session.user) {
        return res.redirect('/');
    }
    next();
};

// Login page route
router.get('/login', redirectIfLoggedIn, (req, res) => {
    res.render('login', {
        title: 'Login - LearnHub'
    });
});

// Registration page route
router.get('/register', redirectIfLoggedIn, (req, res) => {
    res.render('register', {
        title: 'Register - LearnHub'
    });
});

// Handle registration form submission
router.post('/register', [
    // Validation rules
    body('name')
        .trim()
        .isLength({ min: 2, max: 50 })
        .withMessage('Name must be between 2 and 50 characters'),
    body('email')
        .isEmail()
        .normalizeEmail()
        .withMessage('Please enter a valid email address'),
    body('password')
        .isLength({ min: 6 })
        .withMessage('Password must be at least 6 characters long'),
    body('confirmPassword')
        .custom((value, { req }) => {
            if (value !== req.body.password) {
                throw new Error('Passwords do not match');
            }
            return true;
        })
], async (req, res) => {
```

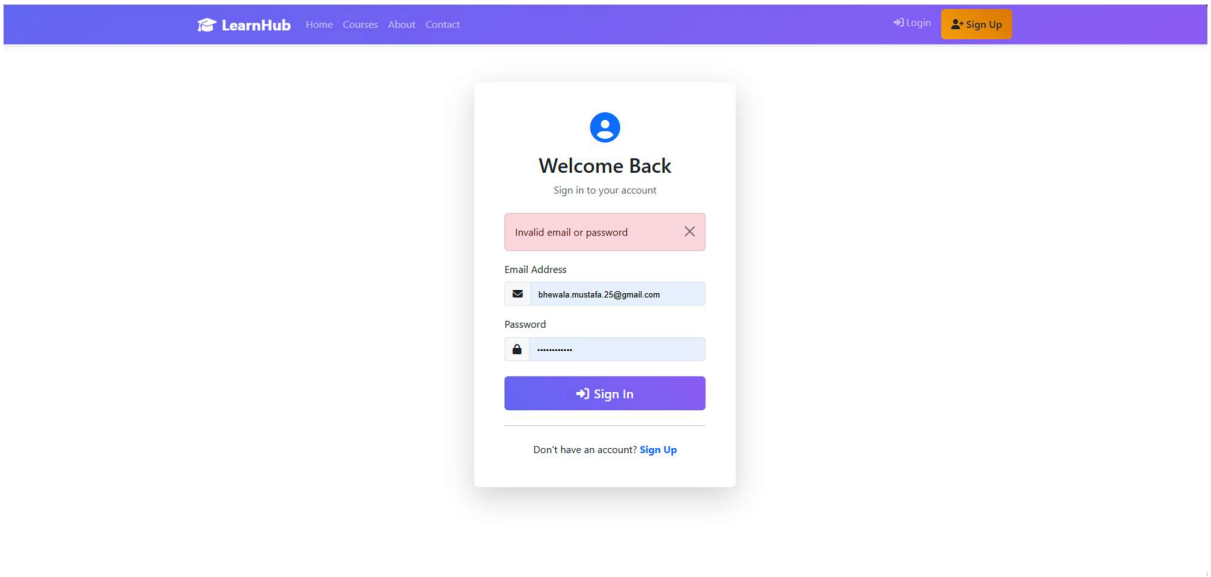
```
    const errors = validationResult(req);

    if (!errors.isEmpty()) {
        const errorMessages = errors.array().map(error => error.msg);
        req.flash('error_msg', errorMessages.join('. '));
        return res.redirect('/auth/register');
    }

    const { name, email, password } = req.body;

    try {
        // Check if user already exists
        database.getDb().get('SELECT * FROM users WHERE email = ?', [email], async
(err, existingUser) => {
            if (err) {
                console.error('Database error:', err);
                req.flash('error_msg', 'Something went wrong. Please try again.');
                return res.redirect('/auth/register');
            }

            if (existingUser) {
                req.flash('error_msg', 'User with this email already exists');
                return res.redirect('/auth/register');
            }

            try {
                // Hash password
                const saltRounds = 12;
                const hashedPassword = await bcrypt.hash(password, saltRounds);

                // Insert new user
                database.getDb().run(
                    'INSERT INTO users (name, email, password) VALUES (?, ?, ?)',
                    [name, email, hashedPassword],
                    function(err) {
                        if (err) {
                            console.error('Error creating user:', err);
                            req.flash('error_msg', 'Error creating account. Please try again.');
                            return res.redirect('/auth/register');
                        }

                        req.flash('success_msg', 'Registration successful! You can now log in.');
                        res.redirect('/auth/login');
                    }
                );
            } catch (hashError) {
                console.error('Error hashing password:', hashError);
                req.flash('error_msg', 'Error creating account. Please try again.');
                res.redirect('/auth/register');
            }
        });
    } catch (error) {
        console.error('Registration error:', error);
        req.flash('error_msg', 'Something went wrong. Please try again.');
        res.redirect('/auth/register');
    }
});
```

```
// Handle login form submission
router.post('/login', [
    // Validation rules
    body('email')
        .isEmail()
        .normalizeEmail()
        .withMessage('Please enter a valid email address'),
    body('password')
        .notEmpty()
        .withMessage('Password is required')
], (req, res) => {
    const errors = validationResult(req);

    if (!errors.isEmpty()) {
        const errorMessages = errors.array().map(error => error.msg);
        req.flash('error_msg', errorMessages.join('. '));
        return res.redirect('/auth/login');
    }

    const { email, password } = req.body;

    // Find user by email
    database.getDb().get('SELECT * FROM users WHERE email = ?', [email], async (err,
user) => {
        if (err) {
            console.error('Database error:', err);
            req.flash('error_msg', 'Something went wrong. Please try again.');
            return res.redirect('/auth/login');
        }

        if (!user) {
            req.flash('error_msg', 'Invalid email or password');
            return res.redirect('/auth/login');
        }

        try {
            // Compare password
            const isMatch = await bcrypt.compare(password, user.password);

            if (!isMatch) {
                req.flash('error_msg', 'Invalid email or password');
                return res.redirect('/auth/login');
            }

            // Create session
            req.session.user = {
                id: user.id,
                name: user.name,
                email: user.email,
                role: user.role,
                avatar: user.avatar
            };

            req.flash('success_msg', `Welcome back, ${user.name}!`);

            // Redirect to the page they were trying to access or home
```

```
            const redirectTo = req.session.returnTo || '/';
            delete req.session.returnTo;
            res.redirect(redirectTo);

        } catch (compareError) {
            console.error('Error comparing password:', compareError);
            req.flash('error_msg', 'Something went wrong. Please try again.');
            res.redirect('/auth/login');
        }
    });
});

// Logout route
router.get('/logout', (req, res) => {
    if (req.session.user) {
        req.session.destroy((err) => {
            if (err) {
                console.error('Error destroying session:', err);
                req.flash('error_msg', 'Error logging out');
                return res.redirect('/');
            }

            res.clearCookie('connect.sid');
            res.redirect('/auth/login');
        });
    } else {
        res.redirect('/');
    }
});

// Profile page
router.get('/profile', (req, res) => {
    if (!req.session.user) {
        req.flash('error_msg', 'Please log in to view your profile');
        return res.redirect('/auth/login');
    }

    database.getDb().get('SELECT * FROM users WHERE id = ?', [req.session.user.id],
(err, user) => {
        if (err) {
            console.error('Error fetching user:', err);
            req.flash('error_msg', 'Error loading profile');
            return res.redirect('/');
        }

        res.render('profile', {
            title: 'My Profile - LearnHub',
            user
        });
    });
});

// Update profile
router.post('/profile', [
    body('name')
        .trim()
        .isLength({ min: 2, max: 50 })
```

```
        .withMessage('Name must be between 2 and 50 characters')
], (req, res) => {
    if (!req.session.user) {
        req.flash('error_msg', 'Please log in to update your profile');
        return res.redirect('/auth/login');
    }

    const errors = validationResult(req);

    if (!errors.isEmpty()) {
        const errorMessages = errors.array().map(error => error.msg);
        req.flash('error_msg', errorMessages.join('. '));
        return res.redirect('/auth/profile');
    }

    const { name } = req.body;
    const userId = req.session.user.id;

    database.getDb().run(
        'UPDATE users SET name = ?, updated_at = CURRENT_TIMESTAMP WHERE id
= ?',
        [name, userId],
        function(err) {
            if (err) {
                console.error('Error updating profile:', err);
                req.flash('error_msg', 'Error updating profile');
                return res.redirect('/auth/profile');
            }

            // Update session
            req.session.user.name = name;
            req.flash('success_msg', 'Profile updated successfully!');
            res.redirect('/auth/profile');
        }
    );
});

module.exports = router;
```

- Screenshot/Output:



- **Conclusion:** Cleaner routing and auditable request logs.

Assignment-5: **Create a Data Submission Form and Handle POST Requests**

- Aim:
  - Build an HTML form and process POST data.

- Theory:
  - HTTP methods and semantics: GET retrieves resources; POST creates or submits data; PUT/PATCH update; DELETE removes. For forms, POST is the standard to avoid exposing data in URLs and to respect idempotency expectations.

  - Body parsing: Use urlencoded parser for HTML forms and json parser for JSON APIs. Ensure correct content-type (application/x-www-form-urlencoded for standard forms).

  - Validation and sanitization: Validate presence, type, and format (e.g., email). Sanitize to prevent injection (strip HTML, allowlists). Apply both client-side (HTML5 attributes, lightweight JS) and server-side validation (source of truth).

  - CSRF protection: For authenticated users, protect POST endpoints with CSRF tokens to prevent cross-site request forgery. Many frameworks offer middleware for this.

  - Post/Redirect/Get (PRG) pattern: After successful POST, redirect to a confirmation page with a flash message. This prevents duplicate submissions on refresh and offers better UX.

  - Error feedback and accessibility: On validation errors, re-render the form with entered values preserved and specific error messages. Use ARIA attributes (aria-describedby) and clear labels for screen readers.

  - Flash messaging: connect-flash integrates with express-session to display ephemeral success/error messages after redirects, as used in the OLP's login view.

  - Rate limiting and spam mitigation: For public forms, throttle requests (e.g., via express-rate-limit), add honeypots or CAPTCHAs if necessary.

  - Data persistence: Start by echoing values back to the user; later, persist to SQLite using parameterized statements. Ensure you store only required data and comply with privacy policies.

  - Security considerations: Limit allowed fields (avoid mass assignment), enforce size limits, and validate file uploads if present (content-type and size checks).

    - Outcome rationale:
      A robust form-handling flow improves reliability and user trust, reduces support overhead, and integrates cleanly with session-backed UX patterns used throughout the OLP.

- Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title><%= title || 'Feedback' %></title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>
  <%- include('partials/navbar') %>
```

```
  <div class="container py-5">
    <h1>Feedback</h1>
    <form action="/contact/feedback" method="POST" class="mt-4">
      <div class="mb-3">
        <label class="form-label">Name</label>
        <input name="name" class="form-control" required />
      </div>
      <div class="mb-3">
        <label class="form-label">Email</label>
        <input type="email" name="email" class="form-control" required />
      </div>
      <div class="mb-3">
        <label class="form-label">Message</label>
        <textarea name="message" class="form-control" rows="4" required></textarea>
      </div>
      <button class="btn btn-primary">Submit</button>
    </form>
  </div>
</body>
</html>
```

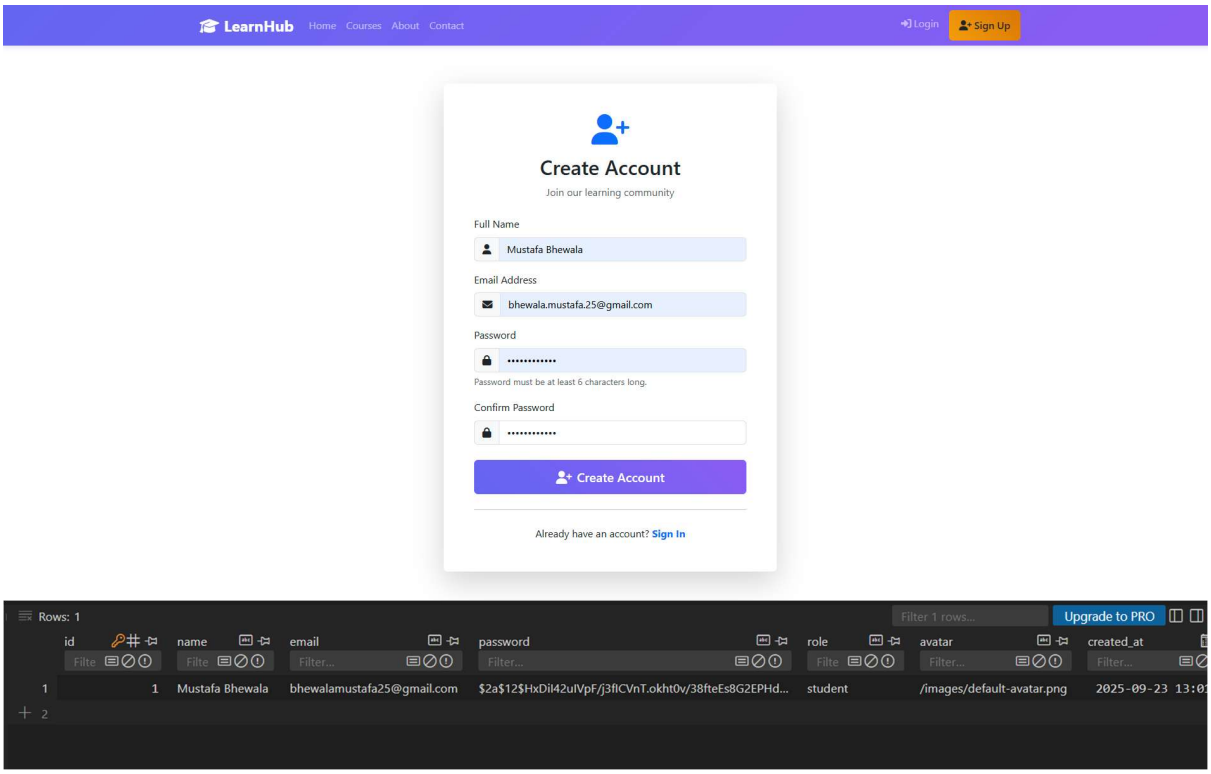
Confirmation view:

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="UTF-8"><meta name="viewport" content="width=device-width,
initial-scale=1.0">
<title>Feedback Received</title></head>
<body>
  <%- include('partials/navbar') %>
  <div class="container py-5">
    <h1>Thanks, <%= data.name %>!</h1>
    <p>We received your message:</p>
    <pre><%= data.message %></pre>
  </div>
</body>
</html>
```

Route:

```
// ...existing code...
const express = require('express');
const router = express.Router();

router.get('/', (_req, res) => res.render('feedback', { title: 'Contact / Feedback' }));

router.post('/feedback', (req, res) => {
  const { name, email, message } = req.body;
  // TODO: Persist to DB or send email
  res.render('feedback-confirm', { data: { name, email, message } });
});

module.exports = router;
// ...existing code...
```

- Screenshot/Output:



- **Conclusion:** Implemented POST handling and dynamic confirmation rendering.

Assignment-6: **Serve Dynamic Content Using Server-Side Rendering**

- Aim:
  - Loop over data and render to a view.

- Code:

```
// Main JavaScript file for Online Learning Platform
document.addEventListener('DOMContentLoaded', function() {

    // Smooth scrolling for anchor links
    document.querySelectorAll('a[href^="#"]').forEach(anchor => {
        anchor.addEventListener('click', function (e) {
            e.preventDefault();
            const target = document.querySelector(this.getAttribute('href'));
            if (target) {
                target.scrollIntoView({
                    behavior: 'smooth',
                    block: 'start'
                });
            }
        });
    });

    // Navbar scroll effect
    window.addEventListener('scroll', function() {
        const navbar = document.querySelector('.navbar');
        if (window.scrollY > 50) {
            navbar.classList.add('scrolled');
        } else {
            navbar.classList.remove('scrolled');
        }
    });

    // Add to cart functionality (placeholder)
    document.querySelectorAll('.btn:contains("Add to Cart")').forEach(button => {
        button.addEventListener('click', function(e) {
            e.preventDefault();

            // Add loading state
            const originalText = this.innerHTML;
            this.innerHTML = '<i class="fas fa-spinner fa-spin me-2"></i>Adding...';
            this.disabled = true;

            // Simulate API call
            setTimeout(() => {
                this.innerHTML = '<i class="fas fa-check me-2"></i>Added!';
                this.classList.remove('btn-primary');
                this.classList.add('btn-success');

                // Update cart badge
                updateCartBadge();

                // Reset button after 2 seconds
                setTimeout(() => {
                    this.innerHTML = originalText;
                    this.classList.remove('btn-success');
                    this.classList.add('btn-primary');
```

```
                this.disabled = false;
            }, 2000);
        }, 1000);
    });
});

// Update cart badge
function updateCartBadge() {
    const badge = document.querySelector('.navbar .badge');
    if (badge) {
        let currentCount = parseInt(badge.textContent) || 0;
        badge.textContent = currentCount + 1;

        // Add animation
        badge.classList.add('animate__animated', 'animate__pulse');
        setTimeout(() => {
            badge.classList.remove('animate__animated', 'animate__pulse');
        }, 1000);
    }
}

// Course card hover effects
document.querySelectorAll('.course-card').forEach(card => {
    card.addEventListener('mouseenter', function() {
        this.style.transform = 'translateY(-5px)';
    });

    card.addEventListener('mouseleave', function() {
        this.style.transform = 'translateY(0)';
    });
});

// Feature card animations on scroll
const observerOptions = {
    threshold: 0.1,
    rootMargin: '0px 0px -50px 0px'
};

const observer = new IntersectionObserver(function(entries) {
    entries.forEach(entry => {
        if (entry.isIntersecting) {
            entry.target.style.opacity = '1';
            entry.target.style.transform = 'translateY(0)';
        }
    });
}, observerOptions);

// Observe feature cards
document.querySelectorAll('.feature-card, .course-card').forEach(card => {
    card.style.opacity = '0';
    card.style.transform = 'translateY(30px)';
    card.style.transition = 'opacity 0.6s ease, transform 0.6s ease';
    observer.observe(card);
});

// Form validation (for future forms)
function validateForm(form) {
```

```
        let isValid = true;
        const inputs = form.querySelectorAll('input[required], textarea[required],
select[required]');

        inputs.forEach(input => {
            if (!input.value.trim()) {
                isValid = false;
                input.classList.add('is-invalid');
            } else {
                input.classList.remove('is-invalid');
            }
        });

        return isValid;
    }

    // Mobile menu toggle enhancement
    const navbarToggler = document.querySelector('.navbar-toggler');
    const navbarCollapse = document.querySelector('.navbar-collapse');

    if (navbarToggler && navbarCollapse) {
        navbarToggler.addEventListener('click', function() {
            const isExpanded = this.getAttribute('aria-expanded') === 'true';

            if (!isExpanded) {
                navbarCollapse.style.maxHeight = navbarCollapse.scrollHeight + 'px';
            } else {
                navbarCollapse.style.maxHeight = '0px';
            }
        });
    }

    // Close mobile menu when clicking outside
    document.addEventListener('click', function(e) {
        const navbar = document.querySelector('.navbar');
        const isClickInsideNav = navbar.contains(e.target);
        const navbarCollapse = document.querySelector('.navbar-collapse');

        if (!isClickInsideNav && navbarCollapse.classList.contains('show')) {
            const navbarToggler = document.querySelector('.navbar-toggler');
            navbarToggler.click();
        }
    });

    // Back to top button
    const backToTopButton = document.createElement('button');
    backToTopButton.innerHTML = '<i class="fas fa-arrow-up"></i>';
    backToTopButton.className = 'btn btn-primary position-fixed';
    backToTopButton.style.cssText = `
        bottom: 20px;
        right: 20px;
        z-index: 1000;
        border-radius: 50%;
        width: 50px;
        height: 50px;
        display: none;
        box-shadow: 0 4px 12px rgba(0, 123, 255, 0.3);
```

```
    `;

    document.body.appendChild(backToTopButton);

    // Show/hide back to top button
    window.addEventListener('scroll', function() {
        if (window.scrollY > 300) {
            backToTopButton.style.display = 'block';
        } else {
            backToTopButton.style.display = 'none';
        }
    });

    // Back to top functionality
    backToTopButton.addEventListener('click', function() {
        window.scrollTo({
            top: 0,
            behavior: 'smooth'
        });
    });

    // Search functionality (placeholder)
    const searchInput = document.querySelector('input[type="search"]');
    if (searchInput) {
        searchInput.addEventListener('input', function() {
            const query = this.value.toLowerCase();
            // Implement search logic here
            console.log('Searching for:', query);
        });
    }

    // Toast notifications
    function showToast(message, type = 'info') {
        const toastContainer = document.querySelector('.toast-container') ||
createToastContainer();

        const toast = document.createElement('div');
        toast.className = `toast align-items-center text-white bg-${type} border-0`;
        toast.setAttribute('role', 'alert');
        toast.innerHTML = `
            <div class="d-flex">
                <div class="toast-body">
                    ${message}
                </div>
                <button type="button" class="btn-close btn-close-white me-2 m-auto" data-bs-
dismiss="toast"></button>
            </div>
        `;

        toastContainer.appendChild(toast);

        const bsToast = new bootstrap.Toast(toast);
        bsToast.show();

        // Remove toast element after it's hidden
        toast.addEventListener('hidden.bs.toast', function() {
            toast.remove();
```

```
    });
}

function createToastContainer() {
    const container = document.createElement('div');
    container.className = 'toast-container position-fixed bottom-0 end-0 p-3';
    document.body.appendChild(container);
    return container;
}

// Make showToast globally available
window.showToast = showToast;

// Performance optimization: Lazy load images
if ('IntersectionObserver' in window) {
    const imageObserver = new IntersectionObserver((entries, observer) => {
        entries.forEach(entry => {
            if (entry.isIntersecting) {
                const img = entry.target;
                img.src = img.dataset.src;
                img.classList.remove('lazy');
                imageObserver.unobserve(img);
            }
        });
    });

    document.querySelectorAll('img[data-src]').forEach(img => {
        imageObserver.observe(img);
    });
}

// Error handling for images
document.querySelectorAll('img').forEach(img => {
    img.addEventListener('error', function() {
        this.src = 'data:image/svg+xml,<svg xmlns="http://www.w3.org/2000/svg"
width="200" height="150" viewBox="0 0 200 150"><rect width="200" height="150"
fill="%23f8f9fa"/><text x="50%" y="50%" text-anchor="middle" dy="0.3em"
fill="%236c757d">Image not found</text></svg>';
    });
});

console.log('LearnHub platform initialized successfully!');
});

// Utility functions
const Utils = {
    // Format currency
    formatCurrency: function(amount, currency = 'USD') {
        return new Intl.NumberFormat('en-US', {
            style: 'currency',
            currency: currency
        }).format(amount);
    },

    // Format date
    formatDate: function(date, options = {}) {
        const defaultOptions = {
```

```
            year: 'numeric',
            month: 'long',
            day: 'numeric'
        };
        return new Intl.DateTimeFormat('en-US', {...defaultOptions, ...options}).format(new
Date(date));
    },

    // Debounce function
    debounce: function(func, wait, immediate) {
        let timeout;
        return function executedFunction() {
            const context = this;
            const args = arguments;
            const later = function() {
                timeout = null;
                if (!immediate) func.apply(context, args);
            };
            const callNow = immediate && !timeout;
            clearTimeout(timeout);
            timeout = setTimeout(later, wait);
            if (callNow) func.apply(context, args);
        };
    },

    // Throttle function
    throttle: function(func, limit) {
        let inThrottle;
        return function() {
            const args = arguments;
            const context = this;
            if (!inThrottle) {
                func.apply(context, args);
                inThrottle = true;
                setTimeout(() => inThrottle = false, limit);
            }
        };
    }
};

// Make Utils globally available
window.Utils = Utils;
```

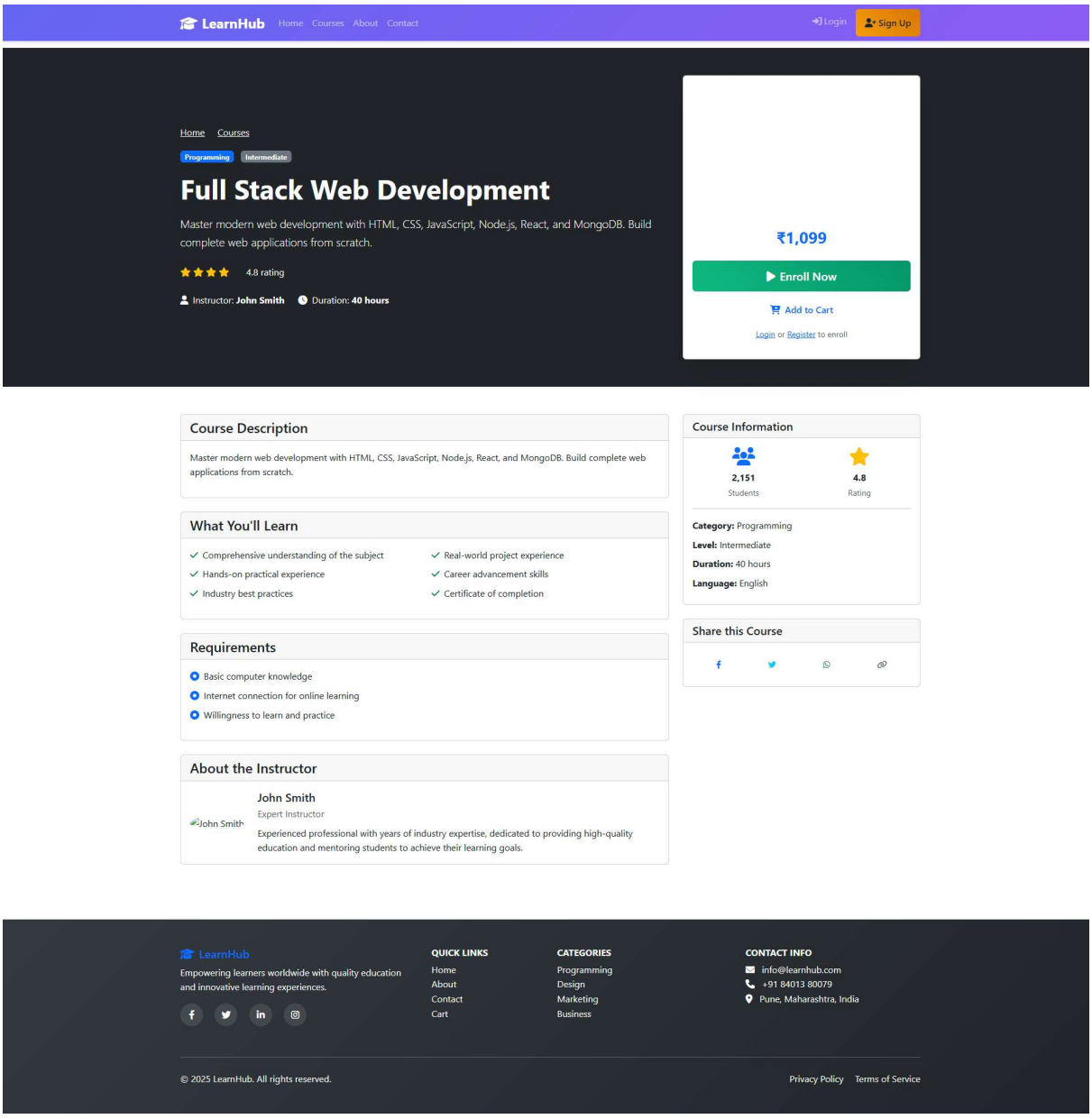
Server.JS:

```
const express = require('express');
const path = require('path');
const bodyParser = require('body-parser');
const session = require('express-session');
const flash = require('connect-flash');

// Import database
const database = require('./config/database');

// Import middleware
const { globalErrorHandler, notFoundHandler } = require('./config/middleware');
```

```
// Import route modules
const homeRoutes = require('./routes/home');
const aboutRoutes = require('./routes/about');
const contactRoutes = require('./routes/contact');
const cartRoutes = require('./routes/cart');
const authRoutes = require('./routes/auth');
const coursesRoutes = require('./routes/courses');

const app = express();
const PORT = process.env.PORT || 3000;

// Set view engine
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

// Middleware
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'public')));

// Session configuration
app.use(session({
    secret: 'learning-platform-secret-key-2024',
    resave: false,
    saveUninitialized: false,
    cookie: {
        secure: false, // Set to true in production with HTTPS
        maxAge: 24 * 60 * 60 * 1000 // 24 hours
    }
}));

// Flash messages
app.use(flash());

// Global variables for templates
app.use((req, res, next) => {
    res.locals.user = req.session.user || null;
    res.locals.success_msg = req.flash('success_msg');
    res.locals.error_msg = req.flash('error_msg');
    res.locals.error = req.flash('error');
    next();
});

// Routes
app.use('/', homeRoutes);
app.use('/about', aboutRoutes);
app.use('/contact', contactRoutes);
app.use('/cart', cartRoutes);
app.use('/auth', authRoutes);
app.use('/courses', coursesRoutes);

// 404 handler
app.use(notFoundHandler);

// Global error handler
app.use(globalErrorHandler);
```

```
// Start server
app.listen(PORT, () => {
   console.log(`🚀 Online Learning Platform server is running on
http://localhost:${PORT}`);
   console.log(`📚 Environment: ${process.env.NODE_ENV || 'development'}`);
});
```

- Screenshot/Output:



- **Conclusion:** Implemented POST handling and dynamic confirmation rendering.

Assignment-7: **Create Dynamic Detail Pages Using URL Parameters**

- Aim:
    - Use Bootstrap Grid to layout course cards.
- Theory:
    - Route parameters vs query strings: Route params (e.g., :id) identify a specific resource and map cleanly to RESTful patterns. Query strings express filtering, sorting, or pagination for collections.
    - Express params: Express parses params and provides them via req.params. Validate params (type, range) and normalize them (cast to number if necessary) before queries.
    - Resource lookup and 404s: If no resource matches, return a 404 and render a friendly page with navigation options. Do not leak internal details in error pages.
    - Canonical URLs and SEO: Use stable, human-readable URLs. Optionally include slugs (/courses/123-ejs-templating) for clarity. Set canonical links to avoid duplicate content issues.
    - Data hydration: Fetch all necessary fields to render the page (title, description, instructor, rating, price). Defer heavy or optional data (e.g., reviews) or load progressively.
    - Breadcrumbs and navigation: Provide “Back to Courses” and related links to support exploration and reduce pogo-sticking.
    - Caching and conditional requests: For popular courses, consider caching details or using ETags/Last-Modified headers to reduce server load.
    - Security: Ensure authorization on restricted resources (e.g., purchased-only content). Avoid predictable enumeration by enforcing access checks.
    - Analytics: Capture page views and engagement to inform recommendations and content strategy (privacy-compliant).
    - Outcome rationale:
        - Parametrized detail routes align URLs to resources, improve findability, and deliver focused information architecture for the OLP.
- Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title><%= title %></title>

  <!-- Bootstrap CSS -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">

  <!-- Font Awesome -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
```

```
awesome/6.4.0/css/all.min.css">

    <!-- Custom CSS -->
    <link rel="stylesheet" href="/css/style.css">
</head>
<body>
    <!-- Navigation -->
    <%- include('partials/navbar') %>

    <!-- Page Header -->
    <section class="bg-gradient-primary text-white py-5">
        <div class="container">
            <div class="row">
                <div class="col-lg-12 text-center">
                    <h1 class="display-4 fw-bold">All Courses</h1>
                    <p class="lead">Discover our comprehensive collection of expert-led
courses</p>
                </div>
            </div>
        </div>
    </section>

    <!-- Filters and Search -->
    <section class="py-4 bg-light">
        <div class="container">
            <div class="row">
                <div class="col-lg-12">
                    <div class="card shadow-sm">
                        <div class="card-body">
                            <form method="GET" action="/courses" class="row g-3">
                                <!-- Search -->
                                <div class="col-md-4">
                                    <div class="input-group">
                                        <span class="input-group-text">
                                            <i class="fas fa-search"></i>
                                        </span>
                                        <input type="text" class="form-control" name="search"
                                               placeholder="Search courses..." value="<%= currentSearch
%>">
                                    </div>
                                </div>

                                <!-- Category Filter -->
                                <div class="col-md-3">
                                    <select class="form-select" name="category">
                                        <option value="all" <%= currentCategory === 'all' ? 'selected' : ''
%>>All Categories</option>
                                        <% categories.forEach(category => { %>
                                            <option value="<%= category.category %>"
                                                    <%= currentCategory === category.category ? 'selected' :
'' %>>
                                                <%= category.category %>
                                            </option>
                                        <% }); %>
                                    </select>
                                </div>
```

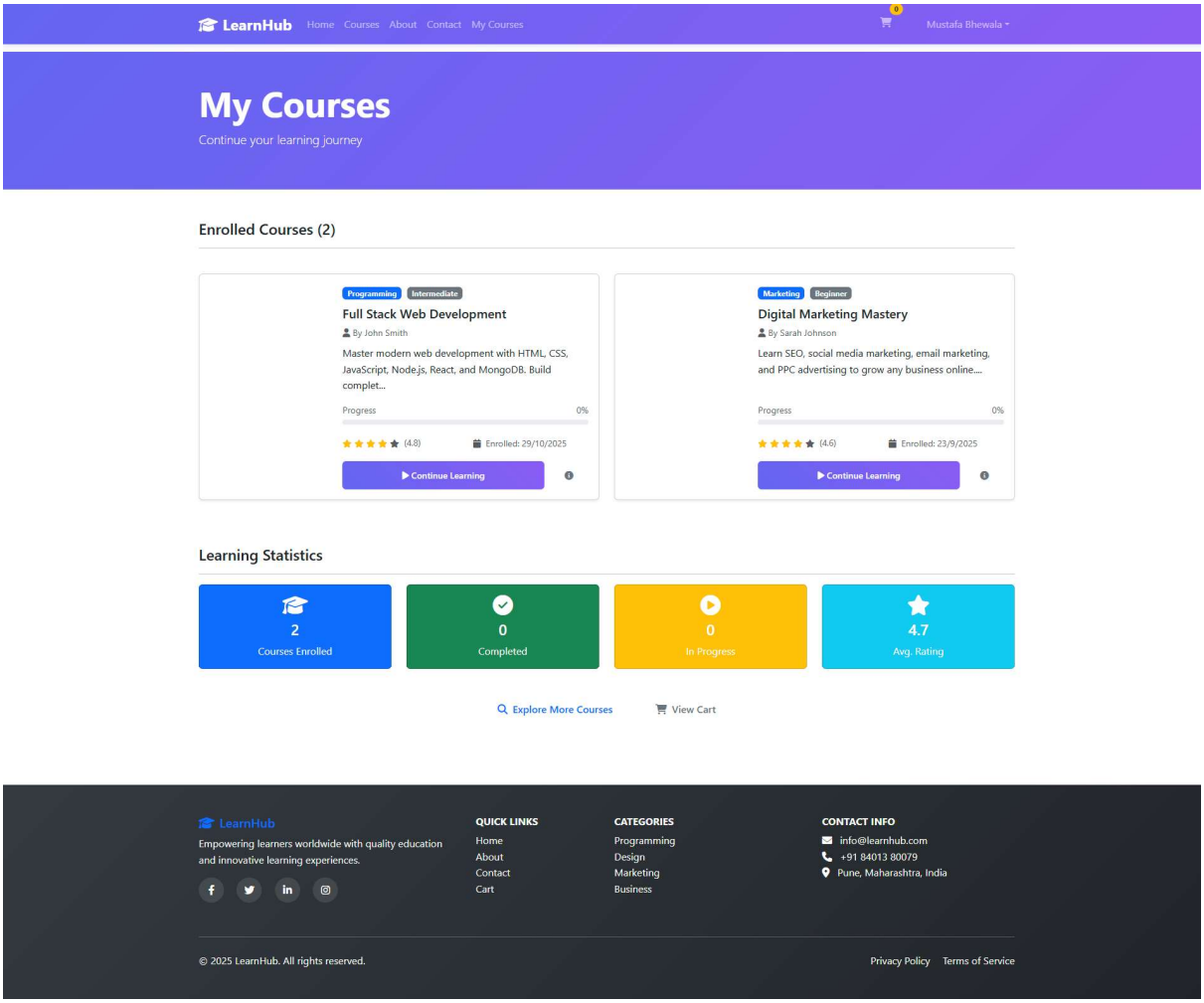
```
                        <!-- Sort -->
                        <div class="col-md-3">
                            <select class="form-select" name="sort">
                                <option value="newest" <%= currentSort === 'newest' ? 'selected'
: '' %>>Newest First</option>
                                <option value="price_low" <%= currentSort === 'price_low' ?
'selected' : '' %>>Price: Low to High</option>
                                <option value="price_high" <%= currentSort === 'price_high' ?
'selected' : '' %>>Price: High to Low</option>
                                <option value="rating" <%= currentSort === 'rating' ? 'selected' :
'' %>>Highest Rated</option>
                                <option value="students" <%= currentSort === 'students' ?
'selected' : '' %>>Most Popular</option>
                            </select>
                        </div>

                        <!-- Filter Button -->
                        <div class="col-md-2">
                            <button type="submit" class="btn btn-primary w-100">
                                <i class="fas fa-filter me-1"></i>Filter
                            </button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </section>

    <!-- Courses Grid -->
    <section class="py-5">
        <div class="container">
            <% if (courses.length === 0) { %>
                <div class="row">
                    <div class="col-12 text-center">
                        <div class="py-5">
                            <i class="fas fa-search fa-5x text-muted mb-3"></i>
                            <h3 class="text-muted">No courses found</h3>
                            <p class="text-muted">Try adjusting your search criteria or browse all
courses.</p>
                            <a href="/courses" class="btn btn-primary">View All Courses</a>
                        </div>
                    </div>
                </div>
            <% } else { %>
                <div class="row">
                    <% courses.forEach(course => { %>
                        <div class="col-lg-4 col-md-6 mb-4">
                            <div class="card course-card h-100 shadow-sm">
                                <img src="<%= course.image %>" class="card-img-top course-image"
                                    alt="<%= course.title %>" onerror="this.src='/images/default-
course.jpg'">

                                <div class="card-body d-flex flex-column">
                                    <div class="mb-2">
                                        <span class="badge bg-primary"><%= course.category
```

```
%></span>
                        <span class="badge bg-secondary ms-1"><%= course.level
%></span>
                    </div>

                    <h5 class="card-title course-title">
                        <a href="/courses/<%= course.id %>" class="text-decoration-
none text-dark">
                            <%= course.title %>
                        </a>
                    </h5>

                    <p class="card-text text-muted small mb-2">
                        <i class="fas fa-user me-1"></i>By <%= course.instructor %>
                    </p>

                    <p class="card-text course-description">
                        <%= course.description ? course.description.substring(0, 100) +
'...' : 'No description available' %>
                    </p>

                    <div class="mt-auto">
                        <div class="row align-items-center mb-3">
                            <div class="col">
                                <div class="d-flex align-items-center mb-1">
                                    <div class="rating-stars me-2">
                                        <% for (let i = 1; i <= 5; i++) { %>
                                            <i class="fas fa-star <%= i <=
Math.floor(course.rating) ? 'text-warning' : 'text-muted' %>"></i>
                                        <% } %>
                                    </div>
                                    <small class="text-muted">(<%= course.rating
%>)</small>
                                </div>
                                <small class="text-muted">
                                    <i class="fas fa-users me-1"></i><%=
course.students.toLocaleString() %> students
                                </small>
                                <% if (course.duration) { %>
                                    <br><small class="text-muted">
                                        <i class="fas fa-clock me-1"></i><%= course.duration
%>
                                    </small>
                                <% } %>
                            </div>
                        </div>

                        <div class="d-flex justify-content-between align-items-center">
                            <h5 class="text-primary mb-0">₹<%=
course.price.toLocaleString() %></h5>
                            <a href="/courses/<%= course.id %>" class="btn btn-outline-
primary btn-sm">
                                View Details
                            </a>
                        </div>
                    </div>
                </div>
```

```
                    </div>
                </div>
            <% }); %>
        </div>

        <!-- Results Info -->
        <div class="row mt-4">
            <div class="col-12 text-center">
                <p class="text-muted">
                    Showing <%= courses.length %> course<%= courses.length !== 1 ? 's' : ''
%>
                    <% if (currentSearch) { %>
                        for "<%= currentSearch %>"
                    <% } %>
                    <% if (currentCategory !== 'all') { %>
                        in <%= currentCategory %>
                    <% } %>
                </p>
            </div>
        </div>
    <% } %>
</div>
</section>

<!-- Footer -->
<%- include('partials/footer') %>

<!-- Bootstrap JS -->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script
>

<style>
    .course-card {
        transition: transform 0.2s ease-in-out;
    }

    .course-card:hover {
        transform: translateY(-5px);
    }

    .course-image {
        height: 200px;
        object-fit: cover;
    }

    .course-title a:hover {
        color: var(--bs-primary) !important;
    }

    .course-description {
        font-size: 0.9rem;
        line-height: 1.4;
    }

    .rating-stars i {
        font-size: 0.8rem;
```

```
        }
    </style>
</body>
</html>
```

- Screenshot/Output:



- **Conclusion:** Implemented URL parameterized pages.

Assignment-8: **Build a Reusable Layout Using EJS Partials**

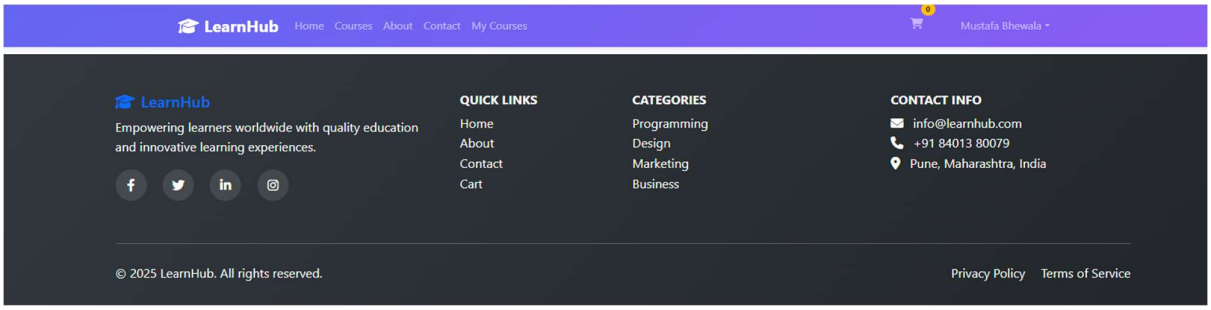
- Aim:
    - Use partials to avoid duplication.
- Theory:
    - DRY templating: Extract common structures (doctype, head, navbar, footer, scripts) into partials and include them in all views. This allows global updates (e.g., a new nav link) with a single change.
    - Layout strategies with EJS: Although EJS lacks native template inheritance, you can simulate layouts via header/footer partials or adopt helpers like ejs-mate for yield sections. Keep it simple initially with include-based composition.
    - Passing locals: Provide per-page variables (title, description) to partials. Establish defaults in server middleware (res.locals) so views don't have to repeat boilerplate.
    - Asset management: Centralize CSS and JS includes within partials. Use cache-busting (query hashes) if you pipeline assets. For Bootstrap via CDN, consider SRI and defer noncritical scripts.
    - Navigation and state: Navbar can reflect login state (res.locals.user). Keep conditionals in partials minimal; compute flags in controllers or middleware for clarity.
    - Accessibility and SEO: Ensure consistent landmarks (header, nav, main, footer). Provide meta tags for viewport, description, and social previews. Maintain heading hierarchy across pages.
    - Theming and design tokens: Define color palette and spacing rules in CSS variables. Keep component-specific styles modular to avoid regressions.
    - Testing and maintainability: Snapshot-test rendered views with representative data to catch unintended changes. Document partial usage and expected locals to onboard new contributors.
    - Error and special pages: Reuse layout on 404 and error pages to maintain brand consistency while clearly signaling status.
    - Outcome rationale:
        - A partial-driven layout system ensures the OLP's UI remains coherent and easy to evolve, minimizing duplication and improving developer productivity.
- Code:

Header and footer partials:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8"><meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title><%= title || 'Online Learning Platform' %></title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
  <link rel="stylesheet" href="/css/style.css">
</head>
```

```
<body>
  <%- include('partials/navbar') %>

  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>

Refactor a page to use them:
<%- include('partials/header', { title: 'About' }) %>
<main class="container py-5">
  <h1>About the Platform</h1>
  <p>Our mission is to democratize education.</p>
</main>
<%- include('partials/footer') %>
```

- Screenshot/Output:



- **Conclusion:** Achieved DRY layout with reusable partials.

Assignment-9: **Connect Node.js to MySQL and Perform Basic CRUD**

- Aim:
  - Connect to MySQL and implement Create/Read/Update/Delete for courses.

- Theory:

  - Relational modeling: Define normalized tables (users, courses, enrollments, cart_items) with primary keys, foreign keys, and constraints (UNIQUE on emails, composite UNIQUE for enrollments). Balance normalization with practical denormalization for read performance (e.g., cached counts).

  - SQLite vs MySQL: SQLite stores data in a single file, requires no server, and suits development or low to moderate concurrency. MySQL scales better for high concurrency and offers advanced features (e.g., replication). SQL dialects vary slightly, but core CRUD is portable.

  - Connection management: sqlite3 opens a single connection to the DB file. For MySQL, you'd use a connection pool. Ensure graceful startup (log connection) and shutdown (close DB).

  - Schema initialization: Create tables at startup if not exists, as seen in OLP's config/database.js. For production, adopt migrations (e.g., knex, umzug) to version and manage schema changes predictably.

  - CRUD operations:
    - Create: INSERT with parameterized statements to prevent SQL injection.
    - Read: SELECT specific columns; avoid SELECT * in production. Add WHERE, ORDER BY, and LIMIT.
    - Update: UPDATE with WHERE to target rows; update timestamps (updated_at).
    - Delete: DELETE with WHERE; consider soft deletes (deleted_at) for recoverability.

  - Parameterized queries and security: Always use placeholders (?) to avoid SQL injection. Validate and sanitize inputs before hitting the DB.

  - Transactions: Group related operations (e.g., enrollments affecting counts) into transactions for atomicity and consistency, especially when multiple tables are involved.

  - Indexing: Add indexes on frequently queried fields (e.g., courses.title, courses.category) to improve performance. In SQLite, be selective to avoid write overhead.

  - Sample data and seeding: Seed data accelerates development and testing. Ensure idempotency (check counts before inserting) to avoid duplicates.

  - Error handling: Surface user-friendly messages while logging technical details. Map database constraint violations (e.g., UNIQUE) to meaningful UX feedback (e.g., "Email already registered").

  - Data access patterns: Encapsulate DB logic in a data layer (repositories) to decouple routes from SQL and ease future DB swaps (SQLite to MySQL/PostgreSQL).

  - Outcome rationale:
    - A disciplined approach to relational data enables persistent, consistent course, user, and enrollment records. SQLite supports quick development; the same CRUD patterns scale to server-based RDBMS when needed.

- Code:

```
CREATE TABLE IF NOT EXISTS courses (
  id INT AUTO_INCREMENT PRIMARY KEY,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  image VARCHAR(512),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

const sqlite3 = require('sqlite3').verbose();
const path = require('path');

// Database connection
const DB_PATH = path.join(__dirname, '..', 'data', 'learning_platform.db');

class Database {
    constructor() {
        this.db = new sqlite3.Database(DB_PATH, (err) => {
            if (err) {
                console.error('Error opening database:', err.message);
            } else {
                console.log('✅ Connected to SQLite database');
                this.initializeTables();
            }
        });
    }

    initializeTables() {
        // Users table
        this.db.run(`
            CREATE TABLE IF NOT EXISTS users (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                email TEXT UNIQUE NOT NULL,
                password TEXT NOT NULL,
                role TEXT DEFAULT 'student',
                avatar TEXT DEFAULT '/images/default-avatar.png',
                created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
                updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
            )
        `, (err) => {
            if (err) {
                console.error('Error creating users table:', err);
            } else {
                console.log('✅ Users table ready');
            }
        });

        // Courses table
        this.db.run(`
            CREATE TABLE IF NOT EXISTS courses (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                title TEXT NOT NULL,
                description TEXT,
                instructor TEXT NOT NULL,
```

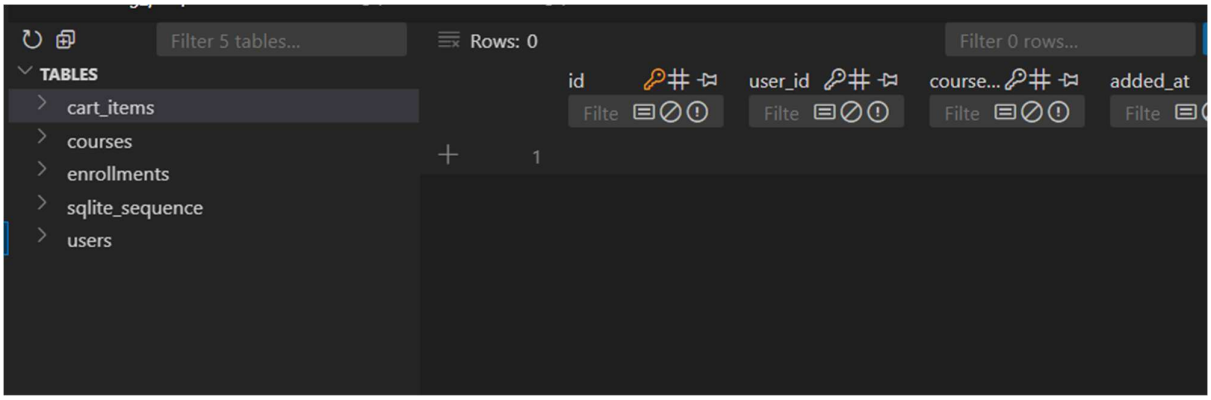
```
            category TEXT NOT NULL,
            price DECIMAL(10,2) NOT NULL,
            rating DECIMAL(3,2) DEFAULT 0,
            students INTEGER DEFAULT 0,
            image TEXT DEFAULT '/images/default-course.jpg',
            duration TEXT,
            level TEXT DEFAULT 'Beginner',
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            updated_at DATETIME DEFAULT CURRENT_TIMESTAMP
        )
    `, (err) => {
        if (err) {
            console.error('Error creating courses table:', err);
        } else {
            console.log('✅ Courses table ready');
            // Insert sample courses after table creation
            this.insertSampleData();
        }
    });

    // Enrollments table
    this.db.run(`
        CREATE TABLE IF NOT EXISTS enrollments (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            course_id INTEGER NOT NULL,
            enrolled_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            progress DECIMAL(5,2) DEFAULT 0,
            FOREIGN KEY (user_id) REFERENCES users (id),
            FOREIGN KEY (course_id) REFERENCES courses (id),
            UNIQUE(user_id, course_id)
        )
    `, (err) => {
        if (err) {
            console.error('Error creating enrollments table:', err);
        } else {
            console.log('✅ Enrollments table ready');
        }
    });

    // Cart items table
    this.db.run(`
        CREATE TABLE IF NOT EXISTS cart_items (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            course_id INTEGER NOT NULL,
            added_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (user_id) REFERENCES users (id),
            FOREIGN KEY (course_id) REFERENCES courses (id),
            UNIQUE(user_id, course_id)
        )
    `, (err) => {
        if (err) {
            console.error('Error creating cart_items table:', err);
        } else {
            console.log('✅ Cart items table ready');
```

```
            }
        });
    }

    insertSampleData() {
        const sampleCourses = [
            {
                title: 'Full Stack Web Development',
                description: 'Master modern web development with HTML, CSS, JavaScript,
Node.js, React, and MongoDB. Build complete web applications from scratch.',
                instructor: 'John Smith',
                category: 'Programming',
                price: 1099.00,
                rating: 4.8,
                students: 2150,
                image: '/images/course-1.jpg',
                duration: '40 hours',
                level: 'Intermediate'
            },
            {
                title: 'Digital Marketing Mastery',
                description: 'Learn SEO, social media marketing, email marketing, and PPC
advertising to grow any business online.',
                instructor: 'Sarah Johnson',
                category: 'Marketing',
                price: 899.00,
                rating: 4.6,
                students: 1870,
                image: '/images/course-2.jpg',
                duration: '25 hours',
                level: 'Beginner'
            },
            {
                title: 'UI/UX Design Fundamentals',
                description: 'Create beautiful and user-friendly interfaces using Figma, Adobe
XD, and modern design principles.',
                instructor: 'Mike Davis',
                category: 'Design',
                price: 999.00,
                rating: 4.9,
                students: 1340,
                image: '/images/course-3.jpg',
                duration: '30 hours',
                level: 'Beginner'
            },
            {
                title: 'Python for Data Science',
                description: 'Learn Python programming, pandas, NumPy, matplotlib, and
machine learning fundamentals.',
                instructor: 'Dr. Emily Chen',
                category: 'Data Science',
                price: 1299.00,
                rating: 4.7,
                students: 890,
                image: '/images/course-4.jpg',
                duration: '45 hours',
                level: 'Intermediate'
```

```
        },
        {
            title: 'Mobile App Development with React Native',
            description: 'Build cross-platform mobile apps for iOS and Android using React
Native and Expo.',
            instructor: 'Alex Rodriguez',
            category: 'Mobile Development',
            price: 1199.00,
            rating: 4.5,
            students: 765,
            image: '/images/course-5.jpg',
            duration: '35 hours',
            level: 'Advanced'
        },
        {
            title: 'Cybersecurity Essentials',
            description: 'Learn network security, ethical hacking, and how to protect systems
from cyber threats.',
            instructor: 'David Wilson',
            category: 'Security',
            price: 1399.00,
            rating: 4.8,
            students: 456,
            image: '/images/course-6.jpg',
            duration: '50 hours',
            level: 'Intermediate'
        }
    ];

    // Check if courses already exist
    this.db.get("SELECT COUNT(*) as count FROM courses", (err, row) => {
        if (err) {
            console.error('Error checking courses:', err);
            return;
        }

        if (row.count === 0) {
            const stmt = this.db.prepare(`
                INSERT INTO courses (title, description, instructor, category, price, rating,
students, image, duration, level)
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            `);

            sampleCourses.forEach(course => {
                stmt.run([
                    course.title,
                    course.description,
                    course.instructor,
                    course.category,
                    course.price,
                    course.rating,
                    course.students,
                    course.image,
                    course.duration,
                    course.level
                ]);
            });
```

```
                stmt.finalize();
                console.log('✅ Sample courses inserted successfully');
            }
        });
    }

    getDb() {
        return this.db;
    }

    close() {
        this.db.close((err) => {
            if (err) {
                console.error('Error closing database:', err.message);
            } else {
                console.log('Database connection closed');
            }
        });
    }
}

module.exports = new Database();
```

- Screenshot/Output:



- **Conclusion: Completed DB integration and basic CRUD.**

Assignment-10: **Develop RESTful APIs for Data Access and Manipulation**

- Aim:
    - Build REST endpoints returning JSON.
- Theory:
    - Resource modeling and URIs: Map nouns to endpoints (e.g., /api/courses, /api/courses/:id). Keep names plural for collections. Use nested routes judiciously (e.g., /api/users/:id/enrollments).
    - HTTP methods and idempotency: GET (safe, idempotent), POST (create, not idempotent), PUT (replace, idempotent), PATCH (partial update, idempotent by definition of target), DELETE (idempotent). Choose methods that align with behavior to meet client expectations and caching intermediaries.
    - Status codes: Use 200/201/204 for success; 400 for validation errors; 401/403 for auth/authorization issues; 404 for missing resources; 409 for conflicts (e.g., duplicate enrollments); 422 for semantic errors; 500 for unexpected server errors.
    - Request/response shape: Accept and return JSON with clear property names. Wrap responses in consistent envelopes (e.g., { data: ... }) and errors in { error: { code, message, details } }.
    - Validation and schema: Enforce request body schemas (e.g., title: string, price: number). Use libraries like Joi/Zod for server-side validation. Return actionable error messages with field-level details.
    - Pagination, filtering, sorting: For list endpoints, support limit, offset/page, sort, and filter parameters (e.g., category). Return metadata (total, page) to help clients build UIs.
    - Authentication and authorization: For protected operations (create/update/delete courses, enrollments), require authentication (session, JWT). Enforce role-based access (admin vs student). Secure secrets and store hashed passwords (bcrypt) outside of API scopes.
    - CORS and security: Configure CORS for trusted origins. Rate-limit sensitive endpoints. Use HTTPS in production. Avoid exposing internal IDs where not necessary or consider opaque IDs.
    - Caching and ETags: For GET endpoints, support ETag/If-None-Match to reduce bandwidth. Cache public data (course lists) with conservative TTLs.
    - Idempotency keys: For POST operations that might be retried by clients, support Idempotency-Key headers to prevent duplicate writes.
    - Observability: Log request IDs, latencies, and error rates. Add structured logs to aid debugging and performance tracking.
    - Documentation and testing: Provide an OpenAPI/Swagger spec for discoverability. Test endpoints with Postman and automated tests (supertest) to guard against regressions.
    - Outcome rationale:
        - A well-designed REST API gives the OLP a stable contract for clients, supports growth to multiple frontends, and simplifies integration and testing while

maintaining security and performance.

- Code:

```
const express = require('express');
const router = express.Router();
const db = require('../config/database');

// GET /api/courses
router.get('/', async (_req, res, next) => {
  try {
    const [rows] = await db.query('SELECT id, title, description, image FROM courses
ORDER BY id DESC');
    res.json({ data: rows });
  } catch (e) { next(e); }
});

// GET /api/courses/:id
router.get('/:id', async (req, res, next) => {
  try {
    const [rows] = await db.query('SELECT id, title, description, image FROM courses
WHERE id=?', [req.params.id]);
    if (!rows.length) return res.status(404).json({ error: 'Not found' });
    res.json({ data: rows[0] });
  } catch (e) { next(e); }
});

// POST /api/courses
router.post('/', async (req, res, next) => {
  try {
    const { title, description, image } = req.body;
    const [r] = await db.query('INSERT INTO courses (title, description, image) VALUES
(?, ?, ?)', [title, description, image]);
    res.status(201).json({ id: r.insertId });
  } catch (e) { next(e); }
});

// PUT /api/courses/:id
router.put('/:id', async (req, res, next) => {
  try {
    const { title, description, image } = req.body;
    await db.query('UPDATE courses SET title=?, description=?, image=? WHERE id=?',
[title, description, image, req.params.id]);
    res.json({ updated: true });
  } catch (e) { next(e); }
});

// DELETE /api/courses/:id
router.delete('/:id', async (req, res, next) => {
  try {
    await db.query('DELETE FROM courses WHERE id=?', [req.params.id]);
    res.status(204).end();
  } catch (e) { next(e); }
});

module.exports = router;
```

- Screenshot/Output:

- **Conclusion:** Exposed RESTful endpoints for integration/testing.