Mohamed Amine Marzouk - med.amine.mzk@horizon-tech.tn

Jakarta EE Course - Final Project Descriptions

Level: L3 | Duration: 14 days (Dec 1-15, 2025)

PROJECT 1: Student Management Information System (SMIS)

Project Overview

Build a complete **Student Management Information System** for a university. The system allows administrators to manage students, courses, and enrollments. Students can view their profile, registered courses, and grades.

Real-World Context

This system is inspired by actual university management systems used by institutions like University of Sousse. It handles student records, course management, enrollment tracking, and grade management.

Technical Requirements

Backend Architecture

- **Servlets**: Authentication servlet for login/logout with form handling
- **Sessions**: Store user information and authentication state
- **JSF**: Create managed beans for Student, Course, and Enrollment management
- **EL/JSTL**: Display dynamic content in JSF pages (list students, courses)
- **REST API**: Expose endpoints for mobile app integration
  - GET /api/students/{id} - Get student details
  - POST /api/students - Create student
  - GET /api/courses - List all courses
  - POST /api/enrollments - Enroll student in course
- **Database (JPA)**: Student, Course, Enrollment entities with relationships

Database Schema

```
Student (id, name, email, enrollmentDate, major, gpa)
Course (id, name, instructor, credits, capacity)
Enrollment (id, student_id, course_id, grade, enrollmentDate)
```

Frontend Requirements

- **Login page** with session management (Servlet)
- **Dashboard** with student/course management (JSF pages)
- **Student list** displaying data with JSTL loops
- **Enrollment form** with validation
- **Grades display** page
- **Design**: Use Bootstrap 5 or Tailwind CSS
    - Responsive navigation bar
    - Card-based layout for student records
    - Responsive tables for course listings
    - Modal forms for enrollment

Functional Requirements

1. User authentication (username/password)
2. Admin can add/edit/delete students
3. Admin can add/edit/delete courses
4. Students can view their enrolled courses
5. Students can view their grades
6. REST API for third-party applications
7. Session timeout after 30 minutes of inactivity
8. Input validation for all forms

Grading Criteria (out of 20)

- **Servlet & Session Implementation** (3 points)
    - Proper login/logout with session management
    - Session timeout handling
- **JSF & Managed Beans** (3 points)
    - Proper JSF page structure
    - Correct bean scoping
    - Proper binding of components
- **EL/JSTL Usage** (2 points)
    - Dynamic data display with JSTL loops
    - EL expressions for data binding
- **REST API Implementation** (3 points)
    - Proper HTTP methods
    - JSON serialization
    - Error handling
- **Database Design & JPA** (3 points)
    - Correct entity relationships
    - Proper annotations

    - CRUD operations

- **UI/UX Design** (3 points)
    - Professional design with Bootstrap/Tailwind
    - Responsive layout
    - User-friendly interface

- **Code Quality & Documentation** (2 points)
    - Clean code
    - Meaningful comments
    - README file

Deliverables

- Complete source code on GitHub
- README.md with setup instructions
- Database SQL script
- Screenshots of all main pages
- REST API documentation
- 5-minute presentation

PROJECT 2: Online Library Management System

Project Overview

Create an **Online Library Management System** where users can browse books, search by title/author, manage their borrowing history, and receive notifications. Librarians can manage the book inventory.

Real-World Context

This system mirrors popular library management systems like those used by city libraries and university libraries worldwide for digital book catalogs and borrowing tracking.

Technical Requirements

Backend Architecture

- **Servlets**: Book search servlet handling GET requests
- **Sessions**: Track user's borrowing history and preferences
- **JSF**: Admin panel for inventory management
- **EL/JSTL**: Display book lists, user history, availability status
- **REST API**: Mobile app and external integrations
    - GET /api/books - Search books (with pagination)
    - GET /api/books/{id} - Book details
    - POST /api/borrowing - Borrow a book
    - GET /api/borrowing/history - User borrowing history

    - POST /api/books/{id}/return - Return a book
- **Database (JPA)**: Book, User, BorrowingRecord entities

Database Schema

```
Book (id, title, author, isbn, publishDate, totalCopies, availableCopies,
category)
User (id, name, email, membershipDate, membershipType)
BorrowingRecord (id, user_id, book_id, borrowDate, dueDate, returnDate, status)
Notification (id, user_id, message, type, isRead, createdDate)
```

Frontend Requirements

- **Homepage** with featured books and search bar
- **Search results page** displaying books with pagination (JSTL)
- **Book details page** with availability status and borrow option
- **User dashboard** showing borrowed books and return dates (JSF)
- **Admin panel** for inventory management (JSF forms)
- **Notification center** for due date reminders
- **Design**: Modern design using Bootstrap 5 or Tailwind
    - Card-based book display
    - Search bar with filters
    - Timeline for borrowing history
    - Alert notifications

Functional Requirements

1. User registration and authentication
2. Browse and search books by title, author, ISBN, or category
3. View book details and availability
4. Borrow books (limit: 5 books maximum)
5. Return books
6. View borrowing history
7. Receive due date reminders via in-app notifications
8. Librarian can add/edit/delete books
9. Track overdue books
10. Responsive design for mobile and desktop

Grading Criteria (out of 20)

- **Servlet Implementation** (2 points)
    - Search servlet properly handling queries
    - Parameter validation
- **Session Management** (2 points)
    - Track user borrowing activity

    - Session security

- **JSF & Forms** (3 points)
    - Admin forms for book management
    - Form validation and error messages
- **EL/JSTL Dynamic Content** (2 points)
    - List rendering with JSTL
    - Conditional display of availability
- **REST API** (3 points)
    - Complete CRUD operations
    - Pagination implementation
    - Proper error responses
- **Database & JPA** (3 points)
    - Entity relationships
    - Queries for search functionality
- **UI/UX & Design** (2 points)
    - Professional appearance
    - Responsive and intuitive
- **Testing & Documentation** (2 points)
    - Test cases documented
    - API documentation

Deliverables

- Complete source code
- Database initialization script
- API documentation (Swagger/postman collection)
- UI screenshots
- README with features overview
- Video demo (3-5 minutes)

PROJECT 3: Restaurant Online Ordering System

Project Overview

Build an **Online Restaurant Ordering System** where customers can browse menu items, add to cart, place orders, and track delivery status. Restaurant staff can manage menu and process orders.

Real-World Context

Inspired by platforms like UberEats, Grab Food, and local restaurant delivery apps widely used in cities. This brings e-commerce concepts to the food service industry.

Technical Requirements

Backend Architecture

- **Servlets**: Shopping cart servlet managing Add/Remove/Update items
- **Sessions**: Shopping cart persistence across pages
- **JSF**: Admin panel for menu management and order processing
- **EL/JSTL**: Display dynamic menu items, order summary, order history
- **REST API**: Mobile app integration
    - GET /api/menu - Get all menu items with categories
    - GET /api/menu/{id} - Menu item details
    - POST /api/orders - Create new order
    - GET /api/orders/{id} - Order details and status
    - PUT /api/orders/{id}/status - Update order status
    - GET /api/orders/user/{userId} - User's order history
- **Database (JPA)**: MenuItem, Order, OrderItem, OrderStatus entities

Database Schema

```
MenuItem (id, name, description, price, category, image_url, availability)
Order (id, user_id, orderDate, totalAmount, status, deliveryAddress)
OrderItem (id, order_id, menuItem_id, quantity, price)
OrderStatus (id, order_id, status, timestamp, notes)
User (id, name, email, phone, address)
```

Frontend Requirements

- **Home page** with restaurant info and featured items
- **Menu page** with categories and filtering (JSTL loops)
- **Item detail modal** with add-to-cart functionality
- **Shopping cart** (Servlet-based) with quantity adjustment
- **Checkout page** with address and payment method
- **Order confirmation** page
- **Order tracking** showing real-time status (JSF)
- **Admin dashboard** for menu and order management
- **Design**: Modern, appetizing design with Bootstrap/Tailwind
    - High-quality food images in cards
    - Smooth cart transitions
    - Color-coded order status badges
    - Mobile-optimized checkout

Functional Requirements

1. Browse menu by category

2. Search menu items
3. Add items to cart with quantity
4. View cart with total calculation
5. Remove items from cart
6. Place order with delivery address
7. View order history
8. Track order status (Pending → Preparing → Ready → Out for Delivery → Delivered)
9. Admin add/edit/delete menu items
10. Admin view and process orders
11. Session-based cart persistence

Grading Criteria (out of 20)

- **Servlet & Cart Management** (3 points)
    - Add/remove/update cart items
    - Session persistence
    - Calculation accuracy
- **JSF Implementation** (3 points)
    - Admin menu management forms
    - Order status update interface
    - Form validation
- **EL/JSTL** (2 points)
    - Dynamic menu display
    - Order history rendering
    - Conditional status display
- **REST API** (3 points)
    - Complete menu endpoints
    - Order creation and tracking
    - Proper status codes
- **Database Design** (2 points)
    - Entity relationships
    - Order and item relationships
- **UI/UX Design** (3 points)
    - Professional restaurant branding
    - Appetizing presentation
    - Smooth user flow
- **Code Quality & Documentation** (2 points)
    - Clean code structure

    - Comments and README

- **Testing** (1 point)
    - Tested functionality documentation

Deliverables

- Complete source code
- Database script with sample data
- REST API documentation
- Screenshots of all pages
- Admin guide
- README with setup instructions

PROJECT 4: Hospital Appointment Booking System

Project Overview

Develop a **Hospital Appointment Booking System** where patients can book appointments with doctors, view appointment history, receive reminders, and doctors can manage their schedules.

Real-World Context

Used by modern healthcare systems globally. Examples include Practo, Doctolib, and hospital-specific systems for appointment scheduling and patient management.

Technical Requirements

Backend Architecture

- **Servlets**: Appointment booking servlet handling form submissions
- **Sessions**: Patient profile and appointment preferences
- **JSF**: Doctor's dashboard, appointment management interface
- **EL/JSTL**: Display available time slots, appointment lists
- **REST API**: Patient mobile app integration
    - GET /api/doctors - List all doctors with specialties
    - GET /api/doctors/{id}/availability - Available time slots
    - POST /api/appointments - Book appointment
    - GET /api/appointments/patient/{id} - Patient's appointments
    - GET /api/appointments/doctor/{id} - Doctor's appointments
    - PUT /api/appointments/{id} - Reschedule appointment
    - DELETE /api/appointments/{id} - Cancel appointment
- **Database (JPA)**: Patient, Doctor, Appointment, Specialty entities

Database Schema

```
Patient (id, name, email, phone, dateOfBirth, address, medicalHistory)
Doctor (id, name, specialization, email, phone, experience, maxAppointmentsPerDay)
Appointment (id, patient_id, doctor_id, appointmentDate, appointmentTime,
             status, notes, createdDate)
Specialty (id, name, description)
Appointment_Status: SCHEDULED, COMPLETED, CANCELLED, NO_SHOW
```

Frontend Requirements

- **Home page** with doctor search and quick booking
- **Doctor listing** page with filters by specialty (JSTL)
- **Appointment booking** form with date/time picker (Servlet)
- **Appointment confirmation** page
- **Patient dashboard** showing upcoming and past appointments (JSF)
- **Doctor dashboard** showing daily schedule (JSF)
- **Appointment management** interface (reschedule, cancel)
- **Email reminder** notification display
- **Design**: Healthcare-themed with Bootstrap/Tailwind
    - Clean, trust-inducing color scheme
    - Calendar widget for date selection
    - Doctor profile cards
    - Status badges for appointments

Functional Requirements

1. Patient registration and login
2. Browse doctors by specialty
3. View doctor profiles and experience
4. Check doctor availability
5. Book appointments with date/time selection
6. Cancel/reschedule appointments
7. View appointment history
8. Receive appointment reminders
9. Doctor can view their daily schedule
10. Doctor can mark appointments as completed
11. Hospital admin can manage doctors and specialties

Grading Criteria (out of 20)

- **Servlet for Booking** (3 points)
    - Form handling and validation
    - Availability checking
- **Session Management** (2 points)
    - Patient profile persistence
    - Appointment preferences storage

- **JSF Interface** (3 points)
    - Doctor and patient dashboards
    - Proper component binding
    - Real-time updates
- **EL/JSTL Implementation** (2 points)
    - Availability display
    - Appointment list rendering
    - Conditional status display
- **REST API** (3 points)
    - Doctor and appointment endpoints
    - Availability checking endpoint
    - Reschedule/cancel functionality
- **Database Design** (2 points)
    - Proper entity relationships
    - Date/time handling
- **UI/UX Design** (2 points)
    - Healthcare-appropriate design
    - Intuitive booking flow
- **Code Quality** (1 point)
    - Clean and documented code
- **Testing** (1 point)
    - Appointment conflict checking

Deliverables

- Complete source code
- Database schema and initialization script
- REST API documentation
- Screenshots of booking flow and dashboards
- Admin guide for managing doctors
- README with features and setup

PROJECT 5: E-Commerce Shopping Platform

Project Overview

Build a complete **E-Commerce Shopping Platform** with product catalog, shopping cart, order management, inventory tracking, and payment status tracking.

Real-World Context

Inspired by Shopify stores, Woocommerce websites, and small e-commerce businesses. Covers the complete cycle from browsing to purchase.

Technical Requirements

Backend Architecture

- **Servlets**: Product search servlet with filtering and pagination
- **Sessions**: Shopping cart and user preferences
- **JSF**: Admin product management, order management dashboard
- **EL/JSTL**: Product listings, category filtering, cart items display
- **REST API**: Mobile app and third-party integrations
  - GET /api/products - List products with filtering and pagination
  - GET /api/products/{id} - Product details with reviews
  - GET /api/categories - Product categories
  - POST /api/cart - Add item to cart
  - DELETE /api/cart/{itemId} - Remove from cart
  - POST /api/orders - Create order from cart
  - GET /api/orders/{id} - Order details
  - GET /api/reviews - Product reviews
- **Database (JPA)**: Product, Category, Cart, Order, OrderItem, Review entities

Database Schema

```
Product (id, name, description, price, stock, sku, category_id, image_url, rating)
Category (id, name, description)
Cart (id, user_id, createdDate, lastModified)
CartItem (id, cart_id, product_id, quantity, price)
Order (id, user_id, orderDate, totalAmount, status, shippingAddress)
OrderItem (id, order_id, product_id, quantity, price)
Review (id, product_id, user_id, rating, comment, createdDate)
```

Frontend Requirements

- **Homepage** with featured products and categories
- **Product listing page** with filtering and sorting (JSTL)
- **Product detail page** with reviews and specifications
- **Shopping cart** with quantity adjustment (Session-based)
- **Checkout page** with order summary
- **Order confirmation** and tracking page
- **User account** with order history (JSF)
- **Admin dashboard** for inventory and order management

- **Design**: Modern e-commerce design with Bootstrap/Tailwind
    - Professional product cards with images
    - Star ratings display
    - Smooth cart interactions
    - Color-coded order statuses

Functional Requirements

1. Browse products by category
2. Search and filter products (price, rating, etc.)
3. View product details with images and reviews
4. Add products to cart with quantity
5. Apply discount codes (optional)
6. Checkout with address entry
7. Order confirmation with order number
8. View order history and tracking
9. Write and view product reviews
10. Admin add/edit/delete products
11. Admin view and process orders
12. Inventory management and low stock alerts

Grading Criteria (out of 20)

- **Servlet Search & Filtering** (2 points)
    - Product search functionality
    - Pagination implementation
- **Session-Based Cart** (3 points)
    - Add/remove/update items
    - Cart persistence
    - Total calculation
- **JSF Admin Interface** (3 points)
    - Product management forms
    - Order management interface
    - Form validation
- **EL/JSTL Dynamic Content** (2 points)
    - Product listings with JSTL loops
    - Category filtering display
    - Review rendering
- **REST API** (3 points)
    - Complete product endpoints
    - Cart and order management

    - Reviews endpoint

- **Database Design** (2 points)
    - Entity relationships
    - Stock management
- **UI/UX Design** (2 points)
    - Professional appearance
    - Responsive and intuitive
    - Product image handling
- **Code Quality & Documentation** (1 point)
    - Clean code and comments

Deliverables

- Complete source code
- Database script with sample products
- REST API documentation
- Admin guide
- Screenshots showing all features
- README with deployment instructions

PROJECT 6: Blog Platform with Comments

Project Overview

Create a **Blog Platform** where users can create, publish, and manage blog posts, readers can comment on posts, and administrators can moderate content.

Real-World Context

Similar to Medium, Dev.to, Wordpress.com blogs. Covers content management, user engagement, and community moderation.

Technical Requirements

Backend Architecture

- **Servlets**: Blog post submission and search servlet
- **Sessions**: Author profile and draft storage
- **JSF**: Post editor interface, admin moderation panel
- **EL/JSTL**: Display blog posts, comments, author info
- **REST API**: External readers and syndication
    - GET /api/posts - List posts (published, with pagination)
    - GET /api/posts/{id} - Post with comments

    - POST /api/posts - Create post
    - PUT /api/posts/{id} - Update post
    - DELETE /api/posts/{id} - Delete post
    - POST /api/posts/{id}/comments - Add comment
    - GET /api/categories - Blog categories
- **Database (JPA)**: Post, Comment, Author, Category entities

Database Schema

```
Author (id, name, email, bio, image_url, joinDate)
Post (id, author_id, title, content, publishDate, status, category_id, views)
Comment (id, post_id, author_id, content, createdDate, status)
Category (id, name, description)
Tag (id, name)
Post_Tag (post_id, tag_id)
```

Frontend Requirements

- **Homepage** with latest posts and featured posts
- **Post listing page** with category filters (JSTL)
- **Post detail page** with comments section
- **Comment form** for readers
- **Author dashboard** to create/edit/publish posts (JSF)
- **Post editor** with rich text editor
- **Category browsing** page
- **Search functionality** for posts
- **Admin moderation panel** for comments
- **Design**: Blog-style with Bootstrap/Tailwind
    - Clean typography
    - Author profile cards
    - Comments thread display
    - Reading time estimate

Functional Requirements

1. User registration (readers and authors)
2. Author create and publish posts
3. Save drafts before publishing
4. Readers browse posts by category
5. Search posts by title and content
6. View post comments
7. Add comments to posts
8. Author can delete/edit their posts
9. Admin moderate comments (approve/reject)
10. Track post views
11. Categorize posts with tags

12. Reading time calculation

Grading Criteria (out of 20)

- **Servlet Implementation** (2 points)
    - Post submission handling
    - Search functionality
- **Session Management** (2 points)
    - Author session persistence
    - Draft storage
- **JSF Editor Interface** (3 points)
    - Post editor form
    - Admin moderation interface
    - Form validation
- **EL/JSTL Content Display** (2 points)
    - Post listing with filters
    - Comments rendering
    - Category display
- **REST API** (3 points)
    - Post CRUD endpoints
    - Comments endpoint
    - Search functionality
- **Database Design** (2 points)
    - Entity relationships
    - Tag system implementation
- **UI/UX Design** (2 points)
    - Professional blog appearance
    - Good typography and readability
    - Responsive design
- **Code Quality** (1 point)
    - Clean code and documentation
- **Testing** (1 point)
    - Comment moderation workflow

Deliverables

- Complete source code
- Database initialization script
- Rich text editor integration guide
- API documentation
- Screenshots of all features
- README with setup and usage instructions

PROJECT 7: Project Management & Task Tracking System

Project Overview

Build a **Project Management System** where teams can create projects, manage tasks, assign work, track progress, and collaborate with comments and status updates.

Real-World Context

Inspired by Asana, Jira, Monday.com. Used by software development teams and project management offices worldwide for task tracking and team coordination.

Technical Requirements

Backend Architecture

- **Servlets**: Task creation servlet handling form data
- **Sessions**: User workspace and preferences
- **JSF**: Project dashboard, task assignment interface
- **EL/JSTL**: Display projects, tasks, progress bars
- **REST API**: Team collaboration tools and mobile access
  - GET /api/projects - User's projects
  - POST /api/projects - Create project
  - GET /api/projects/{id}/tasks - Project tasks
  - POST /api/tasks - Create task
  - PUT /api/tasks/{id} - Update task status/assignee
  - GET /api/tasks/{id}/comments - Task comments
  - POST /api/tasks/{id}/comments - Add comment
  - GET /api/dashboard - Project progress overview
- **Database (JPA)**: Project, Task, User, TaskComment, ProjectMember entities

Database Schema

```
Project (id, name, description, owner_id, createdDate, dueDate, status)
ProjectMember (id, project_id, user_id, role)
Task (id, project_id, title, description, status, priority,
      assignee_id, dueDate, createdDate, completionDate)
```

```
TaskComment (id, task_id, author_id, content, createdDate)
User (id, name, email, role, avatar_url, joinDate)
```

Frontend Requirements

- **Homepage/Dashboard** with project overview and recent tasks
- **Project page** with task list and progress (JSF)
- **Task creation form** (Servlet) with assignment
- **Task detail page** with comments and status updates
- **Team member management** for projects
- **Progress tracking** with visual progress bars (JSTL)
- **Kanban/List view** for tasks
- **Notification panel** for task updates
- **Admin panel** for user and project management
- **Design**: Professional project management UI with Bootstrap/Tailwind
  - Kanban board style
  - Color-coded priority levels
  - Progress visualization
  - Team avatar display

Functional Requirements

1. User registration and team creation
2. Create and manage projects
3. Add team members to projects
4. Create tasks within projects
5. Assign tasks to team members
6. Update task status (To Do → In Progress → Done)
7. Set task priorities (Low, Medium, High)
8. Add comments to tasks
9. View project progress with percentages
10. Set task deadlines
11. Track task completion
12. Send notifications for task assignments
13. View team members' workload

Grading Criteria (out of 20)

- **Servlet Task Management** (2 points)
  - Task creation form handling
  - Data validation
- **Session Management** (2 points)
  - User workspace persistence
  - Project context maintenance

- **JSF Dashboard & Forms** (3 points)
    - Project dashboard interface
    - Task assignment forms
    - Real-time updates
- **EL/JSTL Dynamic Display** (2 points)
    - Task list rendering
    - Progress bar display
    - Status color coding
- **REST API** (3 points)
    - Complete project/task endpoints
    - Comments functionality
    - Dashboard statistics
- **Database Design** (2 points)
    - Entity relationships
    - User roles and permissions
- **UI/UX Design** (2 points)
    - Professional Kanban interface
    - Intuitive task management
    - Team collaboration features
- **Code Quality** (1 point)
    - Clean, documented code
- **Testing** (1 point)
    - Task workflow testing

Deliverables

- Complete source code
- Database schema and sample data
- REST API documentation with examples
- Screenshots of all interfaces
- Team collaboration guide
- README with deployment and usage instructions

GRADING RUBRIC (UNIFIED FOR ALL PROJECTS)

Total: 20 Points

Technical Implementation (12 points)

- **Servlet & Session Management (2-3 points)**
  - Proper servlet lifecycle and request handling
  - Session creation, management, and timeout
  - Cookie handling where applicable
- **JSF & Managed Beans (2-3 points)**
  - Correct page navigation flow
  - Proper bean scoping (@RequestScoped, @SessionScoped, @ApplicationScoped)
  - Component binding and validation
  - Form submission handling
- **EL & JSTL (1-2 points)**
  - Proper use of Expression Language for dynamic values
  - JSTL tags for iteration and conditionals
  - No hardcoded values
- **REST API (2-3 points)**
  - HTTP methods used correctly
  - JSON serialization/deserialization
  - Proper status codes
  - Error handling with meaningful messages
- **Database & JPA (2-3 points)**
  - Correct entity mapping with annotations
  - Proper relationships (OneToMany, ManyToOne, etc.)
  - Efficient queries
  - Transaction management

User Interface & Design (5 points)

- **UI Framework Usage (2 points)**
  - Bootstrap or Tailwind CSS properly utilized
  - Responsive design for mobile and desktop
  - Consistent styling throughout
- **User Experience (2 points)**
  - Intuitive navigation
  - Clear visual hierarchy
  - Appropriate use of colors and typography
  - Form validation with user-friendly error messages
- **Design Aesthetics (1 point)**

- Professional appearance
- Attention to detail
- Brand consistency

Code Quality & Documentation (2 points)

- **Code Standards (1 point)**
  - Proper naming conventions
  - Code organization and structure
  - No code duplication
  - Exception handling
- **Documentation (1 point)**
  - README file with setup instructions
  - Code comments for complex logic
  - API documentation (if REST API implemented)
  - Database schema explanation

Testing & Presentation (1 point)

- **Functionality Testing (0.5 points)**
  - All features work as described
  - No bugs or errors encountered
  - Edge cases handled
- **Presentation (0.5 points)**
  - Clear explanation of project
  - Demonstration of all features
  - Professional presentation

SUBMISSION REQUIREMENTS

For Each Project, Deliver:

1. **Source Code**
   - GitHub repository with complete code
   - .gitignore file configured
   - Organized folder structure
2. **Database Files**
   - SQL script for schema creation
   - Sample data insertion script
   - Database diagram/documentation

3. **Documentation**
    - README.md file (500-1000 words)
    - Setup and installation guide
    - Features overview
    - User guide with screenshots
    - API documentation (for REST APIs)
4. **Screenshots/Media**
    - At least 8-10 screenshots showing all major features
    - Screenshots should include login, main features, admin panels
5. **Presentation**
    - 5-7 minute presentation
    - Live demo of the application
    - Q&A from instructors

Submission Timeline

- **December 1, 2025**: Project assignment and kickoff
- **December 15, 2025**: Final presentation and submission deadline
- **Presentations**: 10-15 minutes per project (including Q&A)

TECHNICAL STACK RECOMMENDATIONS

Backend

- **Server**: WildFly 26+, Payara, or TomEE
- **Framework**: Jakarta EE 10+
- **Build Tool**: Maven or Gradle
- **Database**: PostgreSQL, MySQL, or H2 (for development)

Frontend

- **Bootstrap 5**: `https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/...`
- **Tailwind CSS**: `npm install -D tailwindcss`
- **CDN jQuery** (if needed for interactivity)

Development Tools

- **IDE**: VS Code, IntelliJ IDEA, or Eclipse
- **Git**: GitHub, GitLab, or Bitbucket
- **API Testing**: Postman or Insomnia
- **Database Tool**: DBeaver or MySQL Workbench

IMPORTANT NOTES FOR STUDENTS

1. **Code Quality Matters**: Well-structured, commented code will score higher
2. **Responsive Design**: Ensure your application works on mobile and desktop
3. **Security**: Always validate user input and use prepared statements
4. **Testing**: Test your application thoroughly before submission
5. **Documentation**: Clear documentation makes evaluation easier
6. **Time Management**: Start early, work incrementally
7. **Git Commits**: Make meaningful commits throughout the project
8. **Ask for Help**: Don't hesitate to ask instructors if you're stuck

---

Good Luck with Your Projects! 🚀