CPAN 212 - Movies Application PRD

Product Requirements Document

Project: Movies Management System
Course: CPAN 212 – Modern Web Technologies
Team Size: 3 Members (Gustavo, Amy, and Yagna)
Weight: 30% of Final Grade
Deployment Target: Heroku

---

1. PROJECT OVERVIEW

1.1 Product Vision

A web-based movie management system where users can register, login, and manage a collection of movies with full CRUD operations and user authentication.

1.2 Core Features

- User registration and authentication
- Movie CRUD operations (Create, Read, Update, Delete)
- User-specific access controls
- Form validation and error handling
- Responsive web interface using Pug templates

1.3 Technical Stack

- **Backend**: Node.js + Express.js
- **Database**: MongoDB + Mongoose ODM
- **Frontend**: Pug templating engine
- **Authentication**: Session-based
- **Deployment**: Heroku

---

2. TEAM ORGANIZATION & WORK STREAMS

2.0 Team Member Assignments

- **Developer A**: Gustavo
- **Developer B**: Amy
- **Developer C**: Yagna

2.1 Work Stream Distribution

Team Member	Primary Responsibility	Key Deliverables

Team Member	Primary Responsibility	Key Deliverables
Gustavo (Developer A)	Backend Core & Database	Express setup, Mongoose models, Movie CRUD APIs
Amy (Developer B)	Authentication System	User registration, login/logout, access controls
Yagna (Developer C)	Frontend & Deployment	Pug templates, forms, validation, Heroku deployment

3. WORK STREAM A: BACKEND CORE & DATABASE

Assigned to: Gustavo (Developer A)
Dependencies: None (can start immediately)
Estimated Time: 8-10 hours

3.1 Deliverables

3.1.1 Express Application Setup

```
// Requirements:
- Initialize Express.js application
- Configure middleware (body-parser, express-session, etc.)
- Set up Pug as view engine
- Configure static file serving
- Error handling middleware
```

3.1.2 Database Connection & Models

```
// Mongoose Connection
- MongoDB connection setup
- Connection error handling
- Environment-based configuration

// Movie Model Schema
{
  name: { type: String, required: true },
  description: { type: String, required: true },
  year: { type: Number, required: true, min: 1900, max: 2030 },
  genres: [{ type: String }],
  rating: { type: Number, min: 0, max: 10 },
  createdBy: { type: ObjectId, ref: 'User' },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now }
}
```

3.1.3 Movie Routes (movies.js)

```
// Route Structure:
GET    /movies              - List all movies
GET    /movies/new          - Show add movie form
POST   /movies              - Create new movie
GET    /movies/:id          - Show movie details
GET    /movies/:id/edit     - Show edit form
PUT    /movies/:id          - Update movie
DELETE /movies/:id          - Delete movie
```

3.1.4 Movie Controller Logic

- Input validation for movie data
- Database operations (CRUD)
- Error handling and response formatting
- Pagination for movie listings

3.2 Success Criteria

- [ ] Express app runs without errors
- [ ] MongoDB connection established
- [ ] Movie model properly defined
- [ ] All movie routes respond correctly
- [ ] Basic CRUD operations work
- [ ] Proper error handling implemented

---

4. WORK STREAM B: AUTHENTICATION SYSTEM

Assigned to: Amy (Developer B)
Dependencies: Basic Express setup from Gustavo (Developer A)
Estimated Time: 8-10 hours

4.1 Deliverables

4.1.1 User Model & Schema

```
// User Schema
{
  username: { type: String, required: true, unique: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true }, // hashed
  createdAt: { type: Date, default: Date.now }
}
```

4.1.2 Authentication Routes

```
// Route Structure:
GET  /auth/register     - Show registration form
POST /auth/register     - Process registration
GET  /auth/login        - Show login form
POST /auth/login        - Process login
POST /auth/logout       - Process logout
```

4.1.3 Authentication Middleware

```
// Middleware Functions:
- requireAuth()         - Protect routes requiring login
- requireOwnership()    - Ensure user owns the resource
- hashPassword()        - Password hashing utility
- validateUser()        - User input validation
```

4.1.4 Session Management

- Configure express-session
- Session-based authentication
- User session persistence
- Logout functionality

4.1.5 Access Control Implementation

```
// Protection Rules:
- Add movie: Logged in users only
- Edit movie: Movie owner only
- Delete movie: Movie owner only
- View movies: Public access
```

4.2 Validation Requirements

```
// Registration Validation:
- Username: 3-20 chars, alphanumeric
- Email: Valid email format
- Password: 8+ chars, complexity rules
- Confirm password match

// Login Validation:
- Required: username/email + password
- Account existence check
- Password verification
```

4.3 Success Criteria

- [ ] User registration works with validation
- [ ] User login/logout functionality
- [ ] Password hashing implemented
- [ ] Session management working
- [ ] Access controls properly enforced
- [ ] Error messages for auth failures

---

5. WORK STREAM C: FRONTEND & DEPLOYMENT

Assigned to: Yagna (Developer C)
Dependencies: Backend routes from Gustavo (Developer A) & Amy (Developer B)
Estimated Time: 10-12 hours

5.1 Deliverables

5.1.1 Pug Template Structure

```
// Template Hierarchy:
layout.pug              - Base layout with navigation
index.pug               - Homepage/movie listing
movies/
  ├── show.pug          - Movie details view
  ├── new.pug           - Add movie form
  ├── edit.pug          - Edit movie form
auth/
  ├── register.pug      - Registration form
  ├── login.pug         - Login form
partials/
  ├── header.pug        - Navigation component
  ├── footer.pug        - Footer component
  └── movie-card.pug    - Movie display component
```

5.1.2 Form Implementation

```
// Movie Form Features:
- Movie name input with validation
- Description textarea
- Year number input (1900-2030)
- Genres multi-select/checkboxes
- Rating slider/number input (0-10)
- Form validation with error display
- CSRF protection

// Auth Form Features:
- Registration form with validation
- Login form with error handling
- Client-side validation
- Server-side error display
```

5.1.3 Frontend JavaScript

```
// Client-side Features:
- Delete confirmation dialogs
- Form validation
- Dynamic genre selection
- Rating input enhancement
- Success/error message handling
```

5.1.4 Styling & UX

```
// CSS Requirements:
- Responsive design (mobile-friendly)
- Clean, modern interface
- Form styling and validation states
- Navigation menu
- Movie card layouts
- Error/success message styling
```

5.1.5 Heroku Deployment

```
// Deployment Checklist:
- Procfile configuration
- Environment variables setup
- MongoDB Atlas connection
- Build scripts optimization
- Error logging configuration
- Production-ready settings
```

5.2 Error Handling & Validation

```
// Error Display Requirements:
- Field-level validation errors
- Form submission errors
- Authentication error messages
- Success confirmation messages
- 404/500 error pages
```

5.3 Success Criteria

- [ ] All Pug templates render correctly
- [ ] Forms work with proper validation
- [ ] Responsive design implemented
- [ ] JavaScript functionality working
- [ ] Error messages display properly
- [ ] Successfully deployed to Heroku
- [ ] Application accessible via URL

---

6. INTEGRATION & TESTING PLAN

6.1 Integration Timeline

```
Week 1: Individual development
Week 2: Integration testing
Week 3: Final testing & deployment
```

6.2 Integration Points

1. **A→B**: User authentication with movie ownership
2. **A→C**: Movie routes with Pug templates
3. **B→C**: Auth routes with forms
4. **All**: Complete workflow testing

6.3 Testing Checklist

- [ ] User registration flow
- [ ] Login/logout functionality
- [ ] Add movie (authenticated users)
- [ ] View movie details
- [ ] Edit movie (owners only)
- [ ] Delete movie (owners only)
- [ ] Access control validation
- [ ] Form validation and errors
- [ ] Responsive design testing
- [ ] Production deployment

---

7. TECHNICAL SPECIFICATIONS

7.1 Environment Setup

```
// Required Dependencies:
{
  "express": "^4.18.0",
  "mongoose": "^6.0.0",
  "pug": "^3.0.0",
  "express-session": "^1.17.0",
  "bcrypt": "^5.0.0",
  "express-validator": "^6.14.0",
  "method-override": "^3.0.0",
  "dotenv": "^16.0.0"
}
```

7.2 File Structure

```
movies-app/
├── app.js                # Main application file
├── package.json          # Dependencies
├── Procfile             # Heroku configuration
├── .env                 # Environment variables
├── routes/
│   ├── movies.js        # Movie routes
│   └── auth.js          # Authentication routes
├── models/
│   ├── Movie.js         # Movie model
│   └── User.js          # User model
├── middleware/
│   └── auth.js          # Authentication middleware
├── views/               # Pug templates (see template structure)
├── public/
│   ├── css/
│   ├── js/
│   └── images/
└── controllers/         # Route controllers (optional)
```

7.3 Database Schema

```
// Collections:
users: {
  _id, username, email, password, createdAt
}

movies: {
  _id, name, description, year, genres[], rating,
  createdBy (ref: users._id), createdAt, updatedAt
}
```

---

8. GRADING RUBRIC ALIGNMENT

Requirement	Points	Primary Owner	Validation Criteria

Requirement	Points	Primary Owner	Validation Criteria
Express app with Pug and Mongoose	20	Gustavo (Developer A)	App runs, templates render, DB connected
Add, edit, and delete movie	20	Gustavo (Developer A) + Yagna (Developer C)	Full CRUD operations working
Login and Logout	20	Amy (Developer B)	Auth system functional
User Registration and Route restriction	20	Amy (Developer B)	Registration works, access controls enforced
Heroku Deployment	20	Yagna (Developer C)	App accessible via Heroku URL

9. RISK MITIGATION

9.1 Technical Risks

- **Database Connection**: Use MongoDB Atlas for reliability
- **Session Management**: Implement proper session configuration
- **Deployment Issues**: Test deployment early and often

9.2 Team Coordination

- **Daily Standups**: Brief progress updates
- **Shared Repository**: Git with feature branches
- **Integration Testing**: Regular code merges

9.3 Timeline Buffers

- **Buffer Time**: 2-3 days for unexpected issues
- **Final Testing**: Dedicated testing phase
- **Deployment Buffer**: Early deployment for troubleshooting

10. DELIVERABLE SCHEDULE

Week	Gustavo (Developer A)	Amy (Developer B)	Yagna (Developer C)
Week 1	Express setup, Movie model, Basic routes	User model, Auth routes, Middleware	Template structure, Basic forms
Week 2	Movie CRUD completion, Testing	Access controls, Validation	Form completion, Styling, JS
Week 3	Integration support, Bug fixes	Integration testing, Security	Deployment, Final testing

Total Estimated Hours: 26-32 hours (8-11 hours per person)
Complexity Level: Intermediate

Success Metrics: All rubric requirements met, deployed application functional