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**: Vercel

---

1 · Project Overview

1.1 Product Vision

A web-based movie management system where users can register, log in, 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

Layer	Technology
Backend	Node.js + Express.js
Database	MongoDB + Mongoose ODM
Frontend	Pug templating engine
Authentication	Session-based
Deployment	Vercel

---

2 · Team Organization & Work Streams

2.0 Team Member Assignments

Developer	Identifier
Gustavo	Developer A
Amy	Developer B

Developer	Identifier
Yagna	Developer C

2.1 Work-Stream Distribution

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

3 · Work Stream A – Backend Core & Database

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

3.1 Deliverables

3.1.1 Express Application Setup

```
- 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 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 (`routes/movies.js`)

Method	Path	Purpose
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 Controller Logic

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

3.2 Success Criteria

- Express app runs without errors
- MongoDB connection established
- Movie model defined correctly
- All movie routes respond as expected
- Full CRUD operations functional
- Proper error handling in place

---

4 · Work Stream B – Authentication System

Assigned to: Amy (Developer B)
Dependencies: Basic Express setup from Work-Stream A
Estimated Time: 8–10 h

4.1 Deliverables

4.1.1 User Model & 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

Method	Path	Purpose
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

```
- 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

```
- 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

- Username: 3–20 chars, alphanumeric
- Email: valid email format
- Password: ≥ 8 chars, complexity rules
- Confirm-password match

Login

- Requires username/email **and** password
- Account existence check
- Password verification

4.3 Success Criteria

- User registration with validation
- Login / logout functional
- Password hashing implemented
- Sessions persisted correctly
- Access controls enforced
- Proper error messages for auth failures

---

5 · Work Stream C – Frontend & UI/UX

Assigned to: Yagna (Developer C)
Dependencies: Backend routes from Work-Streams A & B
Estimated Time: 10–12 h

5.1 Deliverables

5.1.1 Pug Template Structure

```
layout.pug              # Base layout (navigation)
index.pug               # Homepage / movie listing
movies/
  ├── show.pug          # Movie details
  ├── 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 card
```

5.1.2 Form Implementation

```
Movie Forms
- Name input (validation)
- Description textarea
- Year number (1900–2030)
- Genres multi-select / checkboxes
- Rating slider / number (0–10)
- CSRF protection
- Client & server-side validation
```

```
Auth Forms
- Registration with validation
- Login with error handling
```

5.1.3 Front-End JavaScript

- Delete confirmation dialogs
- Form validation logic
- Dynamic genre selection
- Rating input enhancement
- Success / error-message handling

5.1.4 Styling & UX

- Responsive design (mobile-friendly)
- Clean, modern interface
- Styled forms & validation states
- Navigation menu
- Movie card layouts
- Consistent message styling

5.1.5 UI/UX Enhancements

```
- Interactive elements and animations
- User experience optimization
- Cross-browser compatibility
- Performance optimization
- Accessibility improvements
- Visual design consistency
```

5.2 Error Handling & Validation

- Field-level validation errors
- Form submission errors
- Authentication error messages

- Success confirmation messages
- Custom 404 / 500 pages

5.3 Success Criteria

- All Pug templates render correctly
- Forms validate client- & server-side
- Responsive design verified
- Client-side JS features operational
- Error messages display properly
- UI/UX optimized and user-friendly

---

6 · Integration & Testing Plan

6.1 Integration Timeline

Week	Focus
1	Individual development
2	Integration testing
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
- Add movie (authenticated)
- View movie details
- Edit movie (owners only)
- Delete movie (owners only)
- Access-control validation
- Form validation & errors
- Responsive design
- Production deployment

---

7 · Technical Specifications

7.1 Environment 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 Project File Structure

```
movies-app/
├── app.js                    # Main application
├── package.json              # Dependencies
├── package-lock.json         # Dependency lock file
├── Procfile                  # Heroku config
├── vercel.json               # Vercel deployment config
├── seed-movies.js            # Database seeding script
├── README.md                 # Project documentation
├── DEMO_GUIDE.md             # Demo instructions
├── DEPLOYMENT.md             # Deployment guide
├── DETAILED_DEMO_SCRIPT.md   # Detailed demo script
├── Git.md                    # Git workflow guide
├── Git.pdf                   # Git guide PDF
├── Project/
│   └── doc/
│       └── movies_prd.md     # This PRD document
├── routes/
│   ├── index.js              # Home route
│   ├── movies.js             # Movie routes
│   └── auth.js               # Authentication routes
├── models/
│   ├── Movie.js              # Movie schema
│   └── User.js               # User schema
├── middleware/
│   ├── auth.js               # Authentication middleware
│   └── flash.js              # Flash message middleware
├── views/                    # Pug templates
│   ├── layout.pug            # Base layout
│   ├── index.pug             # Homepage
│   ├── auth/
│   │   ├── login.pug         # Login form
│   │   └── register.pug      # Registration form
│   ├── movies/
│   │   ├── index.pug         # Movie listing
│   │   ├── show.pug          # Movie details
│   │   ├── new.pug           # Add movie form
│   │   └── edit.pug          # Edit movie form
```

```
│   ├── error/
│   │   ├── 404.pug           # 404 error page
│   │   └── 500.pug           # 500 error page
│   └── partials/
│       ├── header.pug        # Navigation header
│       ├── footer.pug        # Footer component
│       ├── flash.pug         # Flash messages
│       └── delete-modal.pug  # Delete confirmation modal
├── public/
│   ├── css/
│   │   └── style.css         # Main stylesheet
│   ├── js/
│   │   └── main.js           # Client-side JavaScript
│   └── favicon.ico           # Site favicon
└── node_modules/             # NPM dependencies
```

7.3 Database Schema

```
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
Express app with Pug & Mongoose	20	Gustavo (Dev A)	App runs, templates render, DB connected
Add, edit & delete movie	20	Gustavo + Yagna	Full CRUD operations working
Login & Logout	20	Amy + Yagna	Auth system functional
User registration & route restriction	20	Amy (Dev B)	Registration works, access controls enforced
Vercel deployment	20	Gustavo (Dev A)	App accessible via Vercel URL

---

9 · Risk Mitigation

9.1 Technical Risks

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

9.2 Team Coordination

- Daily stand-ups
- Shared Git repository with feature branches
- Regular integration tests

9.3 Timeline Buffers

- 2–3 h buffer for unexpected issues
- Dedicated final-testing phase
- Early deployment for troubleshooting

---

10 · Deliverable Schedule

Week	Gustavo (Dev A)	Amy (Dev B)	Yagna (Dev C)
1	Express setup, Movie model, basic routes	User model, auth routes, middleware	Template structure, basic forms
2	Movie CRUD completion, testing	Access controls, validation	Form completion, styling, JS
3	Integration support, bug fixes, deployment	Integration testing, security	UI/UX polish & final testing

Total Estimated Hours: 26–32 h (≈ 8–11 h per person)
Complexity Level: Intermediate
Success Metrics: All rubric requirements met, deployed application functional

---

11 · Git Repository & Branch Management

11.1 Repository Setup

The project code is hosted on GitHub. To get started, clone the repository using:

```
git clone https://github.com/Guipini/movies-app.git
cd movies-app
npm install
```

11.2 Branch Strategy

Branch Type	Naming Convention	Purpose
main	`main`	Production-ready code
develop	`develop`	Integration branch for features
feature	`feature/name`	New features development
bugfix	`bugfix/name`	Bug fixes

11.3 Developer Workflow

1. **Clone the repository** (first time only)

```
git clone https://github.com/Guipini/movies-app.git
cd movies-app
```

2. **Create a new feature branch**

```
git checkout develop
git pull origin develop
git checkout -b feature/your-feature-name
```

3. **Make changes and commit**

```
git add .
git commit -m "Descriptive message about your changes"
```

4. **Push your branch to GitHub**

```
git push -u origin feature/your-feature-name
```

5. **Create a Pull Request**
    - Go to the repository on GitHub
    - Click "Compare & pull request"
    - Set the base branch to `develop`
    - Add a description of your changes
    - Request review from team members

11.4 Code Review Process

- All code changes require at least one review
- Reviewers should check for:

    - Functionality (meets requirements)
    - Code quality and style
    - Test coverage
    - Documentation

11.5 Merging Guidelines

- Feature branches merge into `develop`
- `develop` merges into `main` for releases
- Use squash merging to keep history clean
- Delete branches after merging

11.6 Conflict Resolution

If you encounter merge conflicts:

```
git checkout develop
git pull origin develop
git checkout your-branch
git merge develop
# Resolve conflicts in your editor
git add .
git commit -m "Resolved merge conflicts"
git push origin your-branch
```