# RIMSS E-Commerce Platform - Technical Specification

## 1. Technical Requirements

### 1.1 Frontend Requirements

* **Framework**: React 18.2.0 with TypeScript
* **State Management**: React Context API
* **Routing**: React Router v6
* **Styling**: Tailwind CSS v3.3
* **Build Tool**: Create React App
* **Package Manager**: npm
* **Minimum Browser Versions**:
  + Chrome 80+
  + Firefox 75+
  + Safari 13+
  + Edge 80+

### 1.2 Backend Requirements

* **Runtime**: Node.js 16+
* **Framework**: Express.js 4.18
* **Database**: MongoDB 5.0+
* **Cache**: Redis 6.0+
* **Authentication**: JWT
* **API Format**: RESTful
* **Documentation**: OpenAPI/Swagger

### 1.3 Development Requirements

* **Version Control**: Git
* **Code Quality**: ESLint, Prettier
* **Testing**: Jest, React Testing Library
* **CI/CD**: GitHub Actions
* **Containerization**: Docker
* **API Testing**: Postman/Insomnia

## 2. System Architecture

### 2.1 Frontend Architecture

* Single Page Application (SPA)
* Component-based architecture
* Responsive design with mobile-first approach
* Progressive Web App (PWA) capabilities
* Client-side routing
* Local storage for cart persistence
* Service worker for offline capabilities

### 2.2 Backend Architecture

* RESTful API architecture
* Microservices-oriented design
* JWT-based authentication
* Role-based access control
* Rate limiting and request throttling
* Error handling middleware
* Logging and monitoring

### 2.3 Database Architecture

* Document-based MongoDB schema
* Indexing for performance optimization
* Data validation at schema level
* Soft deletion implementation
* Audit trails for critical operations
* Data backup and recovery strategy

## 3. Security Implementation

### 3.1 Authentication

* JWT-based token authentication
* Refresh token mechanism
* Password hashing with bcrypt
* Session management
* Two-factor authentication (planned)

### 3.2 Authorization

* Role-based access control (RBAC)
* Permission-based actions
* API endpoint protection
* Resource-level access control

### 3.3 Data Security

* HTTPS/SSL encryption
* XSS protection
* CSRF protection
* Input validation
* SQL injection prevention
* Rate limiting

## 4. Performance Optimization

### 4.1 Frontend Optimization

* Code splitting
* Lazy loading of components
* Image optimization
* Bundle size optimization
* Caching strategies
* Performance monitoring

### 4.2 Backend Optimization

* Database indexing
* Query optimization
* Caching layer
* Connection pooling
* Load balancing
* Request/Response compression

## 5. API Specifications

### 5.1 Authentication API

POST /api/auth/register  
Request:  
{  
 firstName: string;  
 lastName: string;  
 email: string;  
 password: string;  
}  
Response:  
{  
 user: User;  
 token: string;  
}  
  
POST /api/auth/login  
Request:  
{  
 email: string;  
 password: string;  
}  
Response:  
{  
 user: User;  
 token: string;  
}

### 5.2 Product API

GET /api/products  
Query Parameters:  
{  
 category?: string;  
 search?: string;  
 minPrice?: number;  
 maxPrice?: number;  
 sort?: string;  
 page?: number;  
 limit?: number;  
}  
Response:  
{  
 products: Product[];  
 total: number;  
 page: number;  
 pages: number;  
}

## 6. Data Models

### 6.1 User Model

interface User {  
 \_id: ObjectId;  
 firstName: string;  
 lastName: string;  
 email: string;  
 password: string;  
 role: UserRole;  
 isActive: boolean;  
 lastLogin: Date;  
 createdAt: Date;  
 updatedAt: Date;  
}

### 6.2 Product Model

interface Product {  
 \_id: ObjectId;  
 name: string;  
 description: string;  
 price: number;  
 category: string;  
 images: string[];  
 stock: number;  
 isActive: boolean;  
 createdAt: Date;  
 updatedAt: Date;  
}

## 7. Testing Strategy

### 7.1 Unit Testing

* Component testing
* Service testing
* Utility function testing
* Model testing
* Controller testing

### 7.2 Integration Testing

* API endpoint testing
* Database integration testing
* Authentication flow testing
* Payment flow testing

### 7.3 End-to-End Testing

* User journey testing
* Critical path testing
* Cross-browser testing
* Mobile responsiveness testing

## 8. Deployment Strategy

### 8.1 Development Environment

* Local development setup
* Development database
* Mock services
* Hot reloading

### 8.2 Staging Environment

* Staging server setup
* Test database
* Integration testing
* Performance testing

### 8.3 Production Environment

* Production server configuration
* Load balancing
* Database clustering
* Monitoring setup
* Backup strategy

## 9. Monitoring and Logging

### 9.1 Application Monitoring

* Error tracking
* Performance metrics
* User analytics
* System health monitoring

### 9.2 Logging Strategy

* Error logging
* Access logging
* Audit logging
* Performance logging

## 10. Maintenance and Updates

### 10.1 Regular Maintenance

* Security updates
* Dependency updates
* Performance optimization
* Bug fixes

### 10.2 Feature Updates

* Feature planning
* Development cycles
* Testing procedures
* Deployment strategy