Xforce-devthon Backend Documentation

Project Overview

Xforce-devthon is a gamified learning platform for Sri Lankan Advanced Level students. The platform includes features such as user authentication, subject-based learning modules, interactive quizzes, discussion forums, resource library, rewards system, and profile management.

Technology Stack

- **Runtime Environment**: Node.js
- **Web Framework**: Express.js
- **Database**: MongoDB
- **ODM**: Mongoose
- **Authentication**: JWT (planned)
- **File Storage**: Multer (planned)
- **Deployment**: Not yet determined

Project Structure

Copy

```
xforce-devthon-backend/
├── config/             # Configuration files
│   ├── db.js           # MongoDB connection setup
│   └── config.js       # App configuration
├── controllers/        # Route controllers
│   ├── authController.js
│   ├── userController.js
│   ├── subjectController.js
│   ├── quizController.js
│   ├── forumController.js
│   ├── resourceController.js
│   └── rewardController.js
├── middleware/         # Express middleware
│   ├── auth.js         # Authentication middleware
│   ├── errorHandler.js # Error handling middleware
│   └── validation.js   # Input validation
├── models/             # MongoDB schema models
│   ├── userModel.js    # User schema
│   ├── subjectModel.js # Subject schema
│   ├── quizModel.js    # Quiz schema
│   └── [other models]  # To be implemented
├── routes/             # API routes
│   ├── authRoutes.js   # Authentication routes
│   ├── userRoutes.js   # User routes
│   ├── subjectRoutes.js# Subject routes
│   ├── quizRoutes.js   # Quiz routes
│   ├── forumRoutes.js  # Forum routes
│   ├── resourceRoutes.js# Resource routes
│   └── rewardRoutes.js # Reward routes
├── utils/              # Utility functions
```

```
│   ├── logger.js       # Logging utility
│   ├── helpers.js      # Helper functions
│   └── validators.js   # Input validation helpers
├── .env                # Environment variables (not tracked by Git)
├── .env.example        # Example environment variables
├── server.js           # Entry point for the application
├── package.json        # Project dependencies and scripts
└── README.md           # Project documentation
```

Database Models

User Model

javascript

Copy

```javascript
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Please provide your name'],
    trim: true
  },
  email: {
    type: String,
    required: [true, 'Please provide your email'],
    unique: true,
    lowercase: true
  },
  password: {
    type: String,
    required: [true, 'Please provide a password'],
    minlength: 8,
    select: false
  },
  subjects: [{
    type: mongoose.Schema.ObjectId,
    ref: 'Subject'
  }],
  level: {
    type: Number,
    default: 1
  },
  xp: {
    type: Number,
    default: 0
  },
```

```
  reputation: {
    type: Number,
    default: 0
  },
  joinedDate: {
    type: Date,
    default: Date.now
  },
  lastActive: {
    type: Date,
    default: Date.now
  },
  streak: {
    type: Number,
    default: 0
  },
  achievements: [{
    type: mongoose.Schema.ObjectId,
    ref: 'Achievement'
  }],
  points: {
    type: Number,
    default: 0
  },
  role: {
    type: String,
    enum: ['user', 'admin', 'moderator'],
    default: 'user'
  }
}, {
  timestamps: true,
  toJSON: { virtuals: true },
```

```
  toObject: { virtuals: true }
});
```

Subject Model

javascript

Copy

```javascript
// Topic Schema (embedded in Subject)
const topicSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Topic must have a name'],
    trim: true
  },
  description: {
    type: String,
    trim: true
  },
  mastery: {
    type: String,
    enum: ['low', 'medium', 'high'],
    default: 'low'
  },
  order: {
    type: Number,
    required: true
  },
  resources: [{
    type: mongoose.Schema.ObjectId,
    ref: 'Resource'
  }]
}, {
  _id: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});
```

```
// Subject Schema
const subjectSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Subject must have a name'],
    unique: true,
    trim: true
  },
  description: {
    type: String,
    required: [true, 'Subject must have a description']
  },
  color: {
    type: String,
    default: '#3498db'
  },
  gradientFrom: {
    type: String,
    default: '#3498db'
  },
  gradientTo: {
    type: String,
    default: '#2980b9'
  },
  icon: {
    type: String,
    default: 'book'
  },
  topics: [topicSchema],
  isActive: {
    type: Boolean,
    default: true
  }
```

```
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});
```

Quiz Model

javascript

Copy

```javascript
// Option Schema (for multiple choice questions)
const optionSchema = new mongoose.Schema({
  text: {
    type: String,
    required: true
  },
  isCorrect: {
    type: Boolean,
    default: false
  }
}, { _id: false });

// Question Schema
const questionSchema = new mongoose.Schema({
  text: {
    type: String,
    required: [true, 'Question must have text']
  },
  options: [optionSchema],
  correctAnswer: {
    type: String
  },
  explanation: {
    type: String
  },
  difficulty: {
    type: String,
    enum: ['easy', 'medium', 'hard'],
    default: 'medium'
  },
```

```
  points: {
    type: Number,
    default: 10
  },
  isTrueFalse: {
    type: Boolean,
    default: false
  },
  isFillBlank: {
    type: Boolean,
    default: false
  }
});

// Quiz Schema
const quizSchema = new mongoose.Schema({
  title: {
    type: String,
    required: [true, 'Quiz must have a title'],
    trim: true
  },
  description: {
    type: String,
    trim: true
  },
  subject: {
    type: mongoose.Schema.ObjectId,
    ref: 'Subject',
    required: [true, 'Quiz must belong to a subject']
  },
  topic: {
    type: mongoose.Schema.ObjectId,
    ref: 'Subject.topics'
```

```
  },
  difficulty: {
    type: String,
    enum: ['easy', 'medium', 'hard'],
    default: 'medium'
  },
  timeLimit: {
    type: Number,
    default: 30
  },
  questions: [questionSchema],
  isPublished: {
    type: Boolean,
    default: false
  },
  passScore: {
    type: Number,
    default: 70
  },
  createdBy: {
    type: mongoose.Schema.ObjectId,
    ref: 'User'
  },
  attempts: {
    type: Number,
    default: 0
  },
  rating: {
    type: Number,
    default: 0,
    min: 0,
    max: 5
  }
```

```
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});
```

API Routes

Authentication Routes

- POST `/api/auth/register` - Register a new user
- POST `/api/auth/login` - Authenticate user
- GET `/api/auth/me` - Get current user

User Routes

- GET `/api/users/:id` - Get user profile
- PATCH `/api/users/:id` - Update user profile
- GET `/api/users/:id/progress` - Get user progress

Subject Routes

- GET `/api/subjects` - Get all subjects
- GET `/api/subjects/:id` - Get subject by ID
- GET `/api/subjects/:id/topics` - Get topics for a subject
- POST `/api/subjects/test` - Create a test subject (for testing)

Quiz Routes

- GET `/api/quizzes` - Get all quizzes

- GET `/api/quizzes/:id` - Get quiz by ID
- POST `/api/quizzes/:id/attempts` - Submit quiz attempt

Forum Routes

- GET `/api/forum/categories` - Get all forum categories
- GET `/api/forum/topics` - Get all forum topics
- GET `/api/forum/topics/:id` - Get topic by ID
- POST `/api/forum/topics` - Create a new topic

Resource Routes

- GET `/api/resources` - Get all resources
- GET `/api/resources/:id` - Get resource by ID
- GET `/api/resources/:id/download` - Download a resource

Reward Routes

- GET `/api/rewards` - Get all rewards
- POST `/api/rewards/:id/redeem` - Redeem a reward
- GET `/api/rewards/user/:id` - Get user's redeemed rewards

MongoDB Connection

The application connects to MongoDB Atlas with the following connection string format:

Copy

```
mongodb+srv://username:password@clustername.xxxxx.mongodb.net/xforce?retryWrites=true&w=ma
```

Configuration is stored in `.env` file (not tracked by git).

Current Implementation Status

Completed

- Project structure setup
- MongoDB connection configuration
- Basic route templates
- Core data models (User, Subject, Quiz)
- Error handling middleware
- Test endpoint for database verification

Pending

- Authentication implementation
- Controller implementation for all routes
- File upload functionality
- Advanced validation
- Advanced error handling
- Testing

Next Steps

1. Implement controllers for each route:
    - Start with subjectController.js
    - Then quizController.js
    - Then userController.js

2. Implement authentication:
    - JWT token generation and verification
    - Password hashing
    - Protected routes
3. Implement validation middleware:
    - Input validation for all routes
    - Error messages
4. Set up file uploads for resources:
    - Configure Multer
    - Set up storage options

MongoDB Atlas Details

- Cluster: Xforce Cluster
- Database: xforce
- Connection: mongodb+srv://ranawakaramr22 @xforcecluster.akdvacy.mongodb.net/xforce?retryWrites=true&w=majority

GitHub Repository

https://github.com/mehara-rothila/Xforce-devthon-backend.git