

# Direction — MERN Full-Stack Productivity Application

A full-stack web application built with the MERN stack (MongoDB, Express.js, React, Node.js) featuring JWT authentication, AI integration via Google Gemini API, and a gamification system.

**Developed by:** Housseem Eddine Kamel  
**Program:** Software Engineering - DS1

## Table of Contents

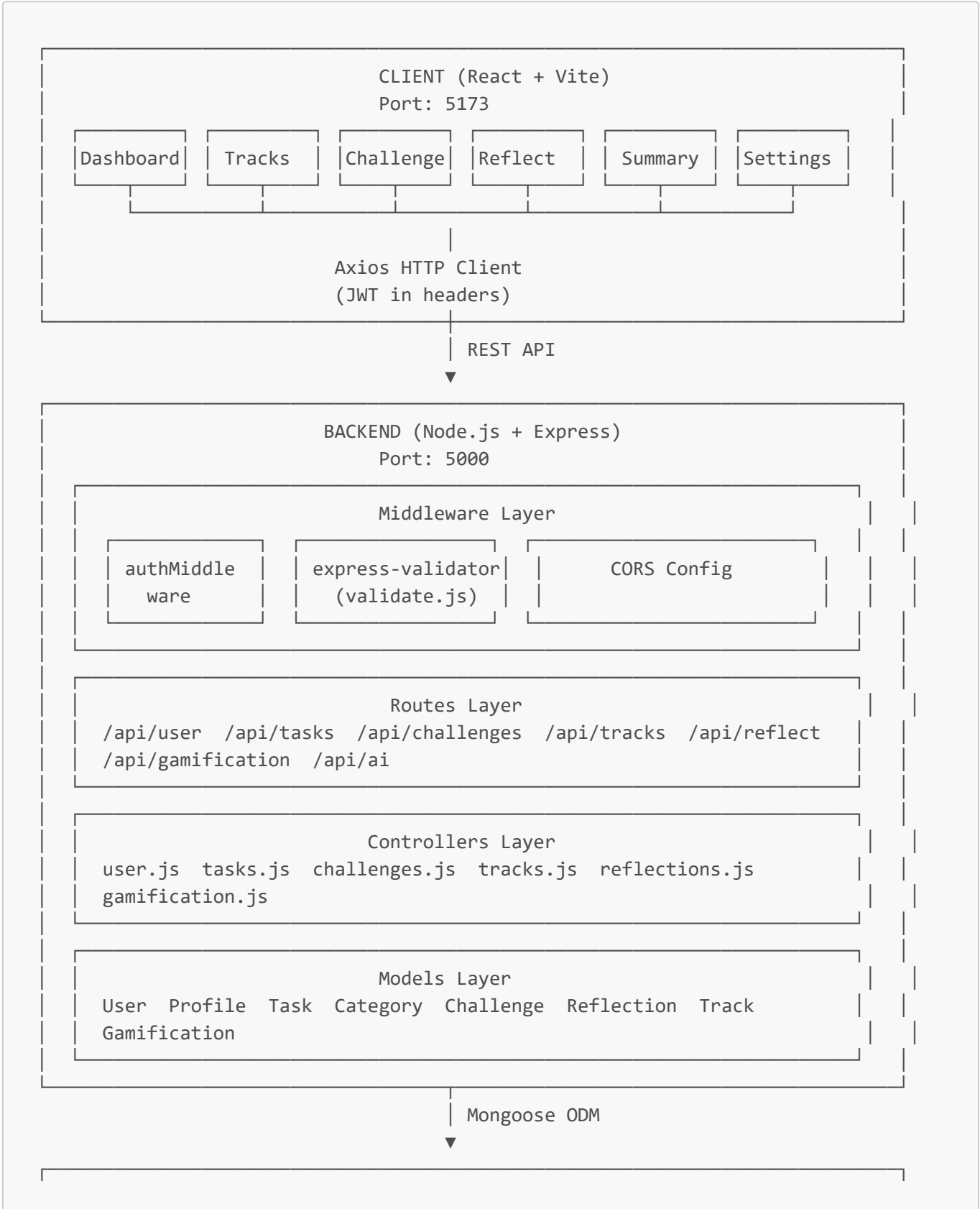
- 1. [Technical Requirements Compliance](#)
- 2. [System Architecture](#)
- 3. [Data Model & Entity Relationships](#)
- 4. [REST API Documentation](#)
- 5. [Security Implementation](#)
- 6. [AI Integration](#)
- 7. [Project Structure](#)
- 8. [Application Screenshots](#)
- 9. [Installation & Setup](#)

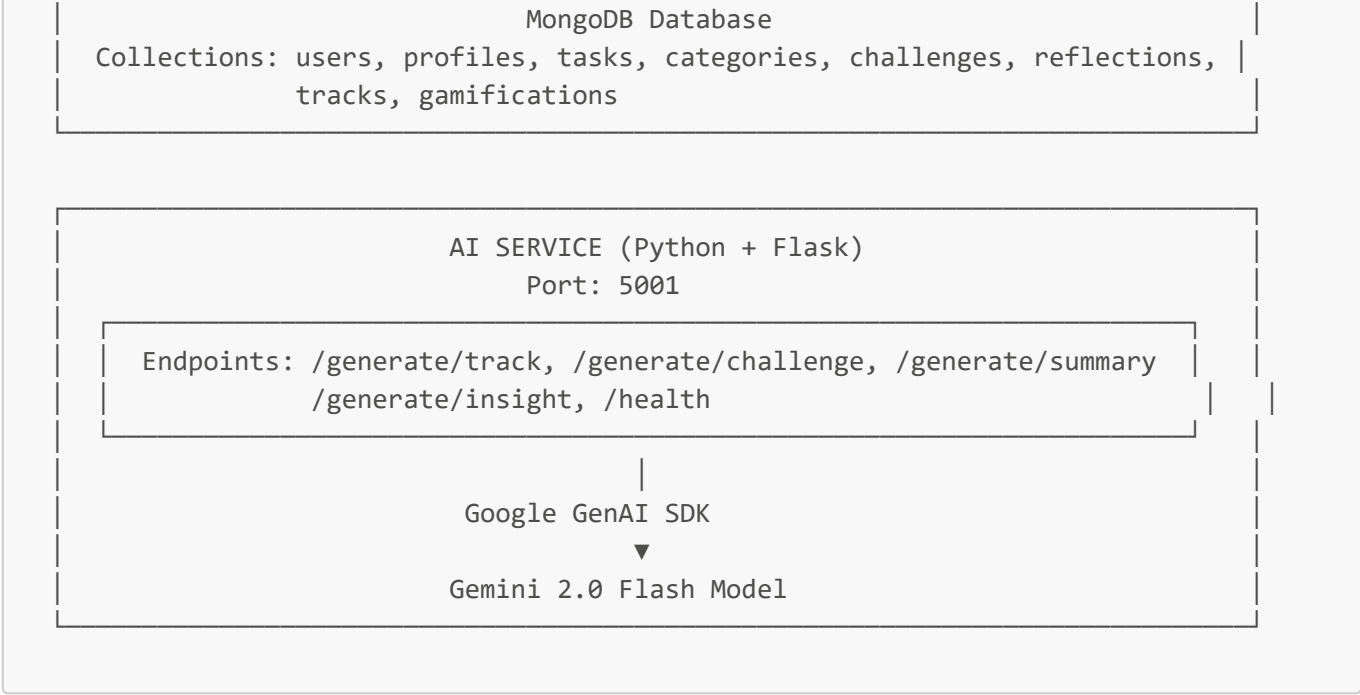
## Technical Requirements Compliance

Requirement	Implementation	Status
MERN Stack	MongoDB + Express.js + React + Node.js	✓
Minimum 5 Entities	8 Mongoose models (User, Profile, Task, Category, Challenge, Reflection, Track, Gamification)	✓
1-to-1 Relationship	User ↔ Profile	✓
1-to-Many Relationships	User → Tasks, User → Challenges, User → Reflections, User → Tracks, User → Categories	✓
Many-to-Many Relationship	Task ↔ Category (bidirectional references)	✓
Full CRUD for each entity	All entities have CREATE, READ, UPDATE, DELETE endpoints	✓
JWT Authentication	Access + Refresh token system with bcrypt password hashing	✓
Protected Routes	authMiddleware on all data routes	✓
Input Validation	express-validator on all endpoints	✓
Environment Variables	.env files for secrets (JWT_SECRET, MONGODB_URI, GEMINI_API_KEY)	✓

Requirement	Implementation	Status
CORS Configuration	Configured in <code>server.js</code>	✓
AI Feature (Gemini API)	Track generation, challenge generation, weekly summaries, reflection insights	✓

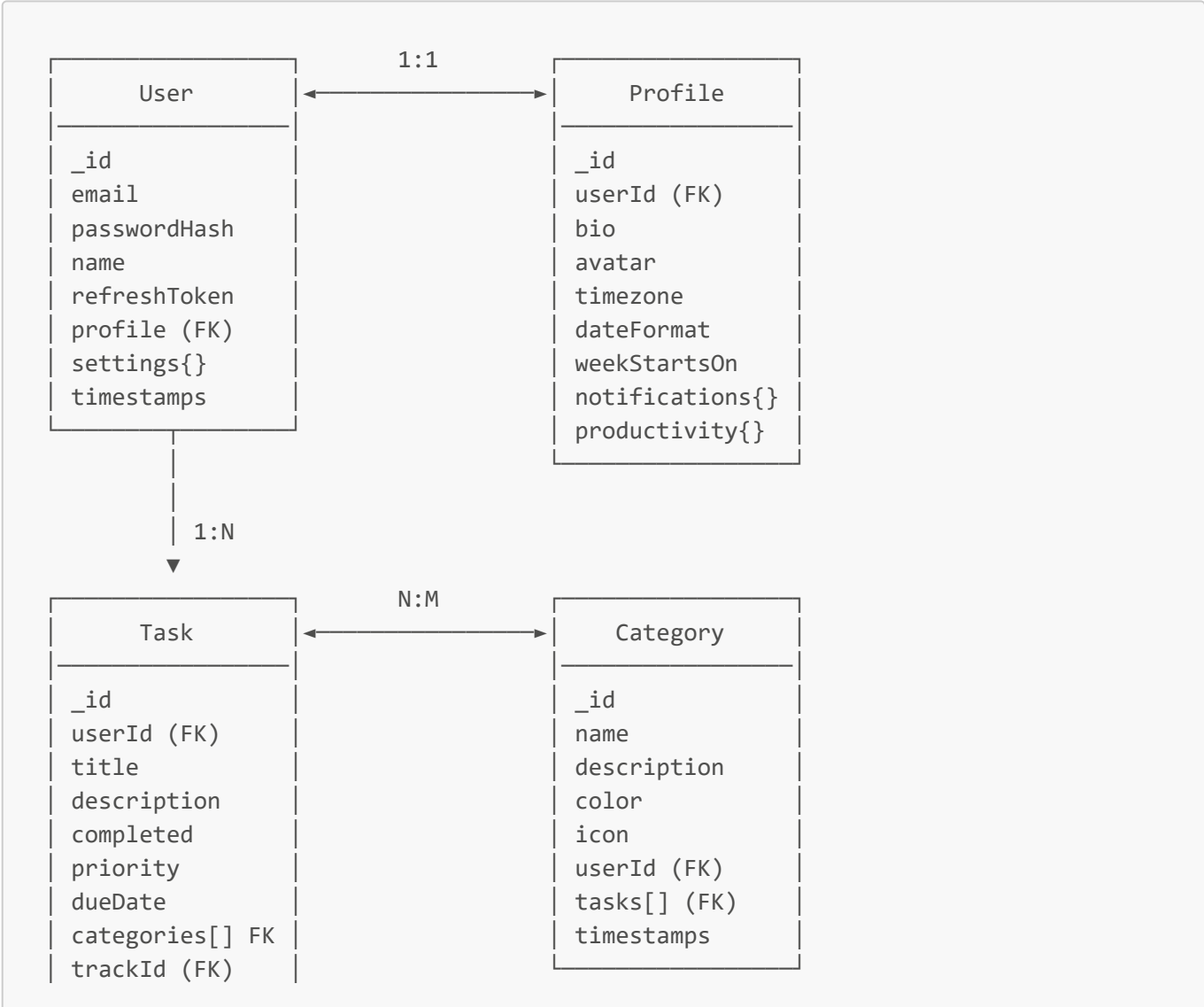
## System Architecture



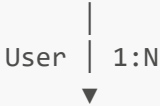


Data Model & Entity Relationships

Entity-Relationship Diagram



trackLevel
timestamps



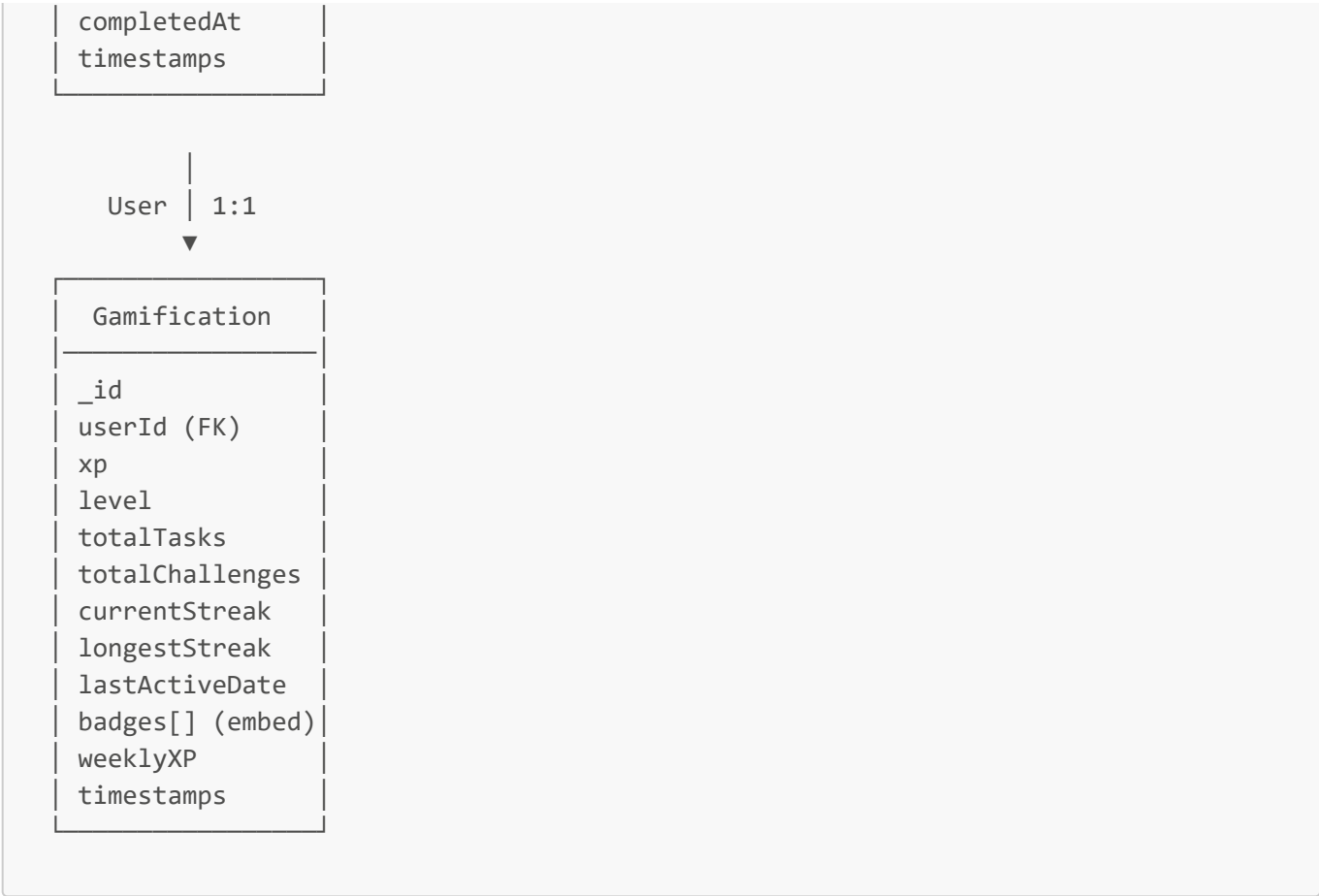
Challenge
<u>_id</u>
userId (FK)
title
description
difficulty
category
status
weekOf
isTimed
durationMinutes
startedAt
completedAt
xpAwarded
timestamps



Reflection
<u>_id</u>
userId (FK)
text
mood
forDate
timestamps



Track
<u>_id</u>
userId (FK)
name
description
status
currentLevel
targetLevel
levels[] (embed)



Relationship Summary

Relationship Type	Entities	Implementation
1-to-1	User ↔ Profile	<code>User.profile</code> references <code>Profile._id</code> ; <code>Profile.userId</code> references <code>User._id</code> with <code>unique: true</code>
1-to-1	User ↔ Gamification	<code>Gamification.userId</code> references <code>User._id</code> with <code>unique: true</code>
1-to-Many	User → Task	<code>Task.userId</code> references <code>User._id</code>
1-to-Many	User → Challenge	<code>Challenge.userId</code> references <code>User._id</code>
1-to-Many	User → Reflection	<code>Reflection.userId</code> references <code>User._id</code>
1-to-Many	User → Track	<code>Track.userId</code> references <code>User._id</code>
1-to-Many	User → Category	<code>Category.userId</code> references <code>User._id</code>
1-to-Many	Track → Task	<code>Task.trackId</code> references <code>Track._id</code>
Many-to-Many	Task ↔ Category	<code>Task.categories[]</code> references <code>Category._id</code> ; <code>Category.tasks[]</code> references <code>Task._id</code>

## Schema Details

### User Schema

```
{
  email: { type: String, required: true, unique: true, lowercase: true },
  passwordHash: { type: String, required: true },
  name: { type: String, required: true },
  refreshToken: { type: String, default: null },
  profile: { type: ObjectId, ref: "Profile" },
  settings: {
    gamificationEnabled: { type: Boolean, default: true },
    challengeFrequency: { type: String, enum: ["daily", "weekly", "off"] }
  }
}
```

### Profile Schema

```
{
  userId: { type: ObjectId, ref: "User", required: true, unique: true },
  bio: { type: String, maxlength: 500 },
  avatar: { type: String },
  timezone: { type: String, default: "UTC" },
  dateFormat: { type: String, enum: ["MM/DD/YYYY", "DD/MM/YYYY", "YYYY-MM-DD"] },
  weekStartsOn: { type: String, enum: ["sunday", "monday"] },
  notifications: { email: Boolean, push: Boolean, dailyReminder: Boolean,
  weeklyDigest: Boolean },
  productivity: { dailyGoal: Number, focusSessionMinutes: Number, breakMinutes:
  Number }
}
```

### Task Schema

```
{
  userId: { type: ObjectId, ref: "User", required: true },
  title: { type: String, required: true, maxlength: 200 },
  description: { type: String, maxlength: 1000 },
  completed: { type: Boolean, default: false },
  priority: { type: String, enum: ["low", "medium", "high"] },
  dueDate: { type: Date },
  categories: [{ type: ObjectId, ref: "Category" }], // Many-to-Many
  trackId: { type: ObjectId, ref: "Track" },
  trackLevel: { type: Number }
}
```

## Category Schema

```
{
  name: { type: String, required: true, maxlength: 50 },
  description: { type: String, maxlength: 200 },
  color: { type: String, match: /^[0-9A-Fa-f]{6}$/ },
  icon: { type: String, maxlength: 50 },
  userId: { type: ObjectId, ref: "User", required: true },
  tasks: [{ type: ObjectId, ref: "Task" }] // Many-to-Many (reverse)
}
```

## Challenge Schema

```
{
  userId: { type: ObjectId, ref: "User" },
  title: { type: String },
  description: { type: String },
  difficulty: { type: String, enum: ["easy", "medium", "hard"] },
  category: { type: String, enum: ["productivity", "wellness", "learning",
    "social", "creativity", "fitness"] },
  status: { type: String, enum: ["pending", "accepted", "active", "completed",
    "skipped", "expired"] },
  weekOf: { type: Date },
  isTimed: { type: Boolean },
  durationMinutes: { type: Number },
  startedAt: { type: Date },
  completedAt: { type: Date },
  xpAwarded: { type: Number }
}
```

## Reflection Schema

```
{
  userId: { type: ObjectId, ref: "User" },
  text: { type: String },
  mood: { type: String }, // "energized", "balanced", "stretched", "depleted"
  forDate: { type: Date }
}
```

## Track Schema

```
{
  userId: { type: ObjectId, ref: "User" },
  name: { type: String },
}
```

```
description: { type: String },
status: { type: String, enum: ["active", "completed", "archived"] },
currentLevel: { type: Number },
targetLevel: { type: Number, default: 5 },
levels: [{ // Embedded subdocument
  title: String,
  description: String,
  focusGoal: String,
  levelNumber: Number,
  completed: Boolean,
  completedAt: Date
}],
completedAt: { type: Date }
}
```

Gamification Schema

```
{
  userId: { type: ObjectId, ref: "User", unique: true },
  xp: { type: Number, default: 0 },
  level: { type: Number, default: 1 },
  totalTasksCompleted: { type: Number },
  totalChallengesCompleted: { type: Number },
  currentStreak: { type: Number },
  longestStreak: { type: Number },
  lastActiveDate: { type: Date },
  badges: [{ // Embedded subdocument
    id: String,
    name: String,
    description: String,
    icon: String,
    earnedAt: Date,
    category: { type: String, enum: ["tasks", "streaks", "challenges",
    "milestones", "special"] }
  }],
  weeklyXP: { type: Number },
  weeklyTasksCompleted: { type: Number }
}
```

REST API Documentation

Authentication Routes (/api/user)

Method	Endpoint	Description	Auth Required
POST	/api/user/register	Create new user account	No
POST	/api/user/login	Authenticate and receive tokens	No

Method	Endpoint	Description	Auth Required
POST	<a href="#">/api/user/logout</a>	Invalidate refresh token	Yes
POST	<a href="#">/api/user/refresh</a>	Get new access token	No (requires refresh token)
GET	<a href="#">/api/user/me</a>	Get current user data	Yes
PATCH	<a href="#">/api/user/me</a>	Update user settings	Yes
DELETE	<a href="#">/api/user/me</a>	Delete user account	Yes

### Task Routes ([/api/tasks](#))

Method	Endpoint	Description	Auth Required
GET	<a href="#">/api/tasks</a>	Get all user tasks	Yes
GET	<a href="#">/api/tasks/:id</a>	Get single task	Yes
POST	<a href="#">/api/tasks</a>	Create new task	Yes
PATCH	<a href="#">/api/tasks/:id</a>	Update task	Yes
DELETE	<a href="#">/api/tasks/:id</a>	Delete task	Yes

### Challenge Routes ([/api/challenges](#))

Method	Endpoint	Description	Auth Required
GET	<a href="#">/api/challenges</a>	Get all challenges	Yes
GET	<a href="#">/api/challenges/active</a>	Get current active challenge	Yes
GET	<a href="#">/api/challenges/meta</a>	Get challenge metadata	Yes
GET	<a href="#">/api/challenges/:id</a>	Get single challenge	Yes
POST	<a href="#">/api/challenges</a>	Create new challenge	Yes
PATCH	<a href="#">/api/challenges/:id</a>	Update challenge	Yes
PATCH	<a href="#">/api/challenges/:id/start</a>	Start timed challenge	Yes
PATCH	<a href="#">/api/challenges/:id/complete</a>	Complete challenge	Yes
PATCH	<a href="#">/api/challenges/:id/skip</a>	Skip challenge	Yes
DELETE	<a href="#">/api/challenges/:id</a>	Delete challenge	Yes

### Track Routes ([/api/tracks](#))

Method	Endpoint	Description	Auth Required
GET	<a href="#">/api/tracks</a>	Get all tracks	Yes
GET	<a href="#">/api/tracks/:id</a>	Get single track	Yes

Method	Endpoint	Description	Auth Required
POST	/api/tracks	Create new track	Yes
PATCH	/api/tracks/:id	Update track	Yes
PATCH	/api/tracks/:id/complete	Complete track	Yes
DELETE	/api/tracks/:id	Delete track	Yes

Reflection Routes (/api/reflections)

Method	Endpoint	Description	Auth Required
GET	/api/reflections	Get all reflections	Yes
GET	/api/reflections/:id	Get single reflection	Yes
POST	/api/reflections	Create new reflection	Yes
PATCH	/api/reflections/:id	Update reflection	Yes
DELETE	/api/reflections/:id	Delete reflection	Yes

Gamification Routes (/api/gamification)

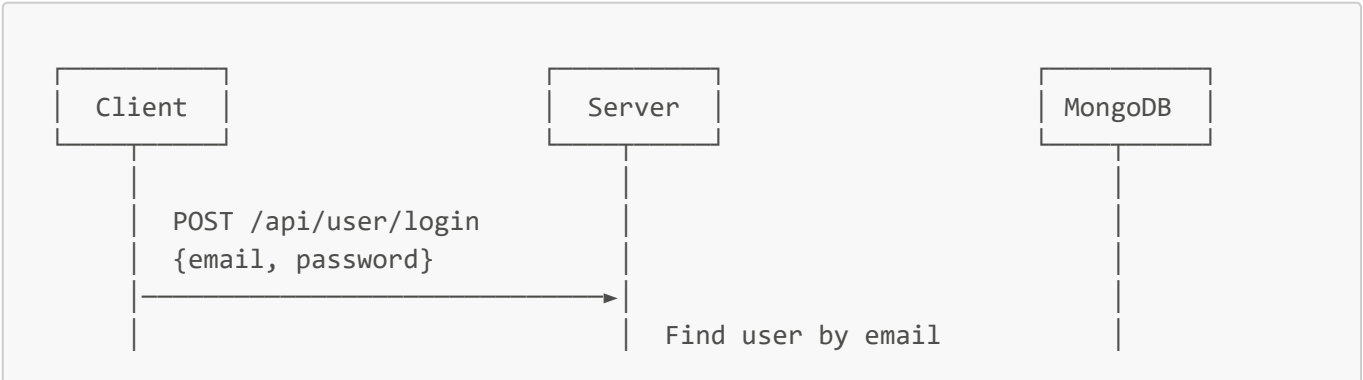
Method	Endpoint	Description	Auth Required
GET	/api/gamification/stats	Get user XP, level, badges	Yes
GET	/api/gamification/activity	Get activity history	Yes

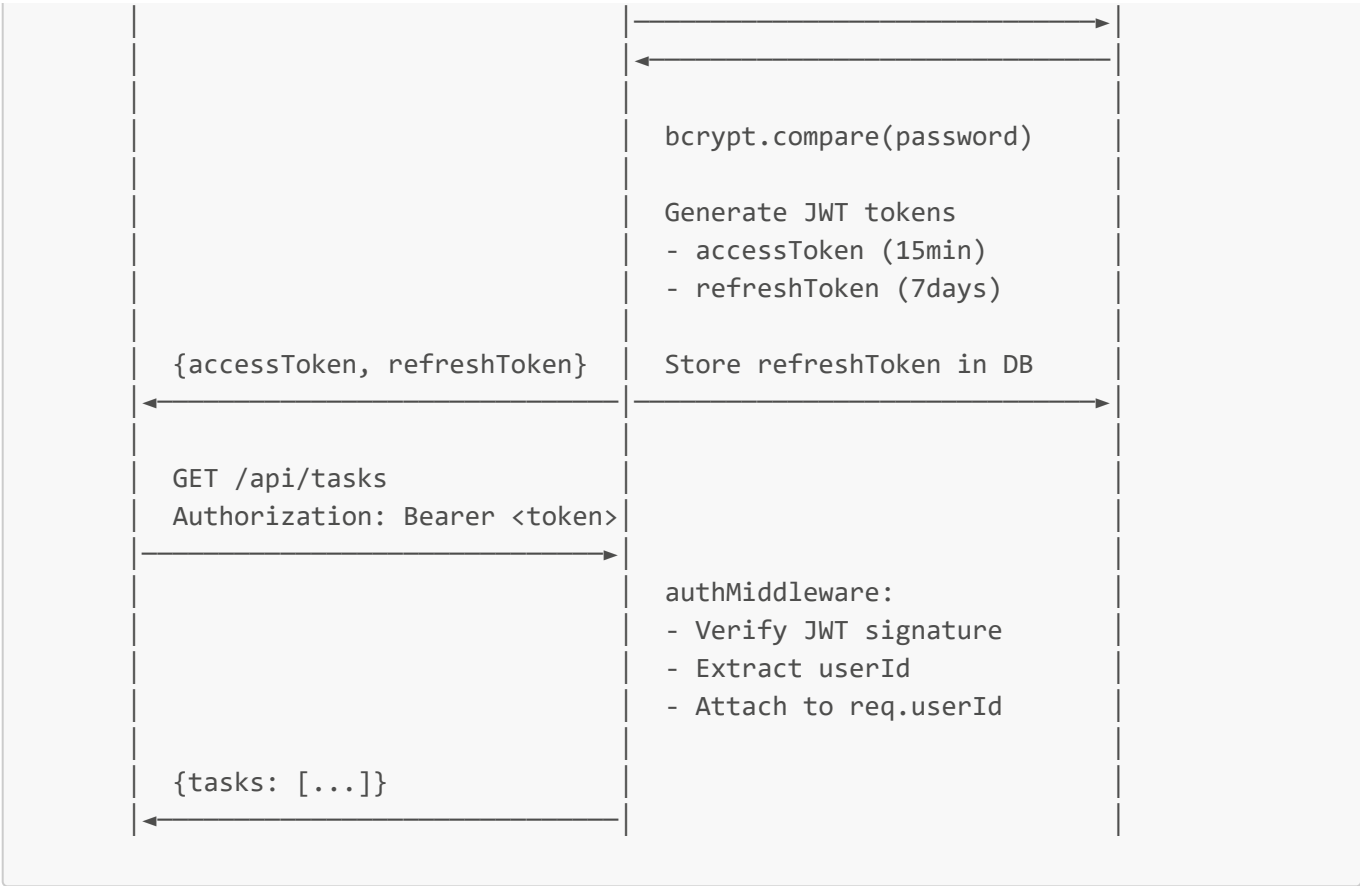
AI Routes (/api/ai)

Method	Endpoint	Description	Auth Required
POST	/api/ai/proxy	Forwards track-generation requests to the AI microservice	Yes

Security Implementation

JWT Authentication Flow





Security Features

Feature	Implementation
Password Hashing	bcrypt with salt rounds = 12
Access Token	JWT, expires in 15 minutes
Refresh Token	JWT, expires in 7 days, stored in DB
Token Refresh	POST <code>/api/user/refresh</code> with refresh token
Protected Routes	<code>authMiddleware</code> validates JWT on every request
Input Validation	express-validator on all POST/PATCH endpoints
CORS	Configured to allow frontend origin
Environment Variables	Secrets stored in <code>.env</code> files

Validation Middleware

All input is validated using `express-validator`:

```
// Example: Task creation validation
validateCreateTask = [
  body("title").trim().notEmpty().isLength({ max: 200 }),
  body("description").optional().trim().isLength({ max: 1000 }),
  body("priority").optional().isIn(["low", "medium", "high"]),
  body("categories").optional().isArray(),
```

```
body("categories.*").optional().isMongoId(),
  handleValidationErrors,
];
```

## AI Integration

### AI Service Architecture

The AI service is a separate Flask microservice that interfaces with Google's Gemini API.

Endpoint	Purpose	Input	Output
/generate/track	Generate track level content	user_name, track_theme, current_level	level_title, level_description, focus_goal
/generate/challenge	Generate personalized challenge	user_name, difficulty, category	title, description
/generate/summary	Generate weekly summary	tasks_completed, reflections, mood_trend	narrative summary
/generate/insight	Generate reflection insight	reflection_text, mood	personalized insight

### AI Service Configuration

```
# ai_service/app.py
model = genai.GenerativeModel("gemini-2.0-flash")
generation_config = {
    "temperature": 0.8,
    "top_p": 0.95,
    "max_output_tokens": 1024
}
```

## Project Structure

```
Project Direction/
├── client/                                # React Frontend
│   ├── src/
│   │   ├── api/                          # API client functions
│   │   │   ├── tasks.js
│   │   │   ├── challenges.js
│   │   │   ├── tracks.js
│   │   │   └── reflections.js
```

```

├── gamification.js
├── ai.js
├── components/                # Reusable UI components
│   ├── CosmicLoader.jsx
│   ├── RankCard.jsx
│   ├── ActivityHeatmap.jsx
│   ├── AchievementsModule.jsx
│   ├── Sidebar.jsx
│   └── ProtectedRoute.jsx
├── context/                   # React Context
│   └── AuthContext.jsx        # Authentication state
├── hooks/                     # Custom hooks
│   └── useAI.js
├── pages/                     # Page components
│   ├── DashboardPage.jsx      # Main task view
│   ├── TracksPage.jsx         # Track management
│   ├── ChallengePage.jsx      # Challenge system
│   ├── ReflectionPage.jsx     # Daily reflections
│   ├── SummaryPage.jsx        # Weekly AI summary
│   ├── SettingsPage.jsx       # User settings
│   ├── LoginPage.jsx
│   └── RegisterPage.jsx
├── styles/                    # CSS files
│   ├── variables.css          # CSS custom properties
│   ├── components.css         # Component styles
│   └── pages.css              # Page-specific styles
├── App.jsx                    # Main app with routing
├── main.jsx                   # Entry point
├── package.json
├── vite.config.js

├── server/                    # Node.js Backend
│   ├── controllers/           # Route handlers
│   │   ├── user.js            # Auth + user CRUD
│   │   ├── tasks.js           # Task CRUD + gamification
│   │   ├── challenges.js       # Challenge CRUD
│   │   ├── tracks.js           # Track CRUD
│   │   ├── reflections.js      # Reflection CRUD
│   │   └── gamification.js     # XP/stats endpoints
│   ├── middleware/
│   │   ├── auth.js            # JWT verification
│   │   └── validate.js        # express-validator rules
│   ├── models/                # Mongoose schemas
│   │   ├── User.js
│   │   ├── Profile.js
│   │   ├── Task.js
│   │   ├── Category.js
│   │   ├── Challenge.js
│   │   ├── Reflection.js
│   │   ├── Track.js
│   │   └── Gamification.js
│   ├── routes/                # Express routers
│   │   ├── user.js
│   │   └── tasks.js

```

```
├── challenges.js
├── tracks.js
├── reflections.js
├── gamification.js
├── ai.js
├── server.js                # Express app entry
├── package.json
├── .env.example



├── ai_service/              # Python AI Microservice
│   ├── app.py               # Flask app + Gemini integration
│   ├── requirements.txt     # Python dependencies
│   └── .env.example

├── Screenshots/            # Application screenshots
│   ├── 0-Register.png
│   ├── 1-Login.png
│   ├── 2-Dashboard.png
│   ├── 3-Tracks.png
│   ├── 4-Challenges.png
│   ├── 5-Reflections.png
│   ├── 6-Summary.png
│   └── 7-Settings.png





└── README.md                # This file
```



## Application Screenshots

### Authentication

Registration	Login
 Register	 Login
User registration with email, password, and name validation	JWT-based login with access and refresh tokens

### Main Application

Dashboard	Tracks
 Dashboard	 Tracks
Task management with gamification stats, XP tracking, and activity heatmap	AI-generated productivity tracks with level progression
Challenges	Reflections
 Challenges	 Reflections
Timed challenges with category selection and circular timer	Daily mood tracking and reflection journaling

Weekly Summary	Settings
 Summary	 Settings
AI-generated weekly performance analysis	User preferences and gamification settings

# Installation & Setup

## Prerequisites

- Node.js v18+
- Python 3.9+
- MongoDB (local or Atlas)
- Google Gemini API key

## 1. Clone Repository

```
git clone <repository-url>
cd "Project Direction"
```

## 2. Backend Setup

```
cd server
npm install

# Create .env file
cp .env.example .env
# Edit .env with your values:
# MONGODB_URI=mongodb://localhost:27017/direction
# JWT_SECRET=your-secret-key
# JWT_REFRESH_SECRET=your-refresh-secret

npm run dev
```

## 3. AI Service Setup

```
cd ai_service
python -m venv .venv
source .venv/bin/activate # Windows: .venv\Scripts\activate
pip install -r requirements.txt

# Create .env file
cp .env.example .env
# Edit .env with your values:
# GEMINI_API_KEY=your-gemini-api-key
```

```
python app.py
```

4. Frontend Setup

```
cd client
npm install

# Create .env file (optional, defaults work for local dev)
# VITE_API_BASE_URL=http://localhost:5000

npm run dev
```

5. Access Application

- Frontend: http://localhost:5173
- Backend API: http://localhost:5000
- AI Service: http://localhost:5001

Tech Stack

Layer	Technology	Version
Frontend	React	19.x
Build Tool	Vite	6.x
Routing	React Router	7.x
HTTP Client	Axios	1.x
Backend	Node.js	18.x
Framework	Express	4.x
Database	MongoDB	7.x
ODM	Mongoose	8.x
Authentication	jsonwebtoken	9.x
Password Hashing	bcryptjs	2.x
Validation	express-validator	7.x
AI Service	Flask	3.x
AI Model	Google Gemini	2.0 Flash