

Backend Implementation Guide - Journal App

This guide provides a comprehensive checklist for implementing the backend using **Node.js**, **Express**, and **MongoDB** for your daily journaling application.

Table of Contents

1. [Project Setup](#)
 2. [Database Schema Design](#)
 3. [Authentication APIs](#)
 4. [Journal APIs](#)
 5. [Category APIs](#)
 6. [Prompt APIs](#)
 7. [Security Implementation](#)
 8. [Frontend Integration](#)
-

1. Project Setup

Initialize Node.js Project

- Create `package.json` with project metadata

```
bash
```

```
npm init -y
```

- Install core dependencies:

```
bash
```

```
npm install express mongoose dotenv cors bcryptjs jsonwebtoken
```

- Install development dependencies:

```
bash
```

```
npm install --save-dev nodemon
```

- Create project structure:

```
journal-backend/
├── config/
│   ├── db.js
│   └── constants.js
├── models/
│   ├── User.js
│   ├── Journal.js
│   ├── Category.js
│   └── JournalPrompt.js
├── routes/
│   ├── auth.js
│   ├── journals.js
│   ├── categories.js
│   └── prompts.js
├── controllers/
│   ├── authController.js
│   ├── journalController.js
│   ├── categoryController.js
│   └── promptController.js
├── middleware/
│   ├── auth.js
│   ├── validation.js
│   └── errorHandler.js
└── utils/
    ├── tokenGenerator.js
    ├── validators.js
    └── seedPrompts.js
├── .env
├── .env.example
├── .gitignore
├── server.js
├── package.json
└── README.md
```

Environment Variables (.env)

Set up `.env` file with:

```
env
```

```
PORT=5000
MONGODB_URI=mongodb://localhost:27017/journal-app
JWT_SECRET=your-super-secret-jwt-key-change-this
JWT_EXPIRE=7d
NODE_ENV=development
CORS_ORIGIN=http://localhost:3000
```

Express Server Setup

- Create `server.js` with Express app
 - Configure CORS for frontend communication
 - Set up body parser middleware (JSON)
 - Connect to MongoDB
 - Set up error handling middleware
 - Configure routes
 - Add health check endpoint
-

2. Database Schema Design

User Model (`models/User.js`)

- Create User schema with fields:
 - `username` (String, required, unique, 3-30 chars)
 - `email` (String, required, unique, lowercase, valid format)
 - `password` (String, required, hashed, 6-128 chars)
 - `profile` (Object):
 - `firstName` (String, max 50 chars)
 - `lastName` (String, max 50 chars)
 - `avatar` (String, URL)
 - `bio` (String, max 500 chars)
 - `preferences` (Object):
 - `theme` (String, enum: light/dark/auto, default: auto)
 - `timezone` (String, default: UTC)
 - `reminderTime` (String, HH:MM format)
 - `role` (String, enum: user/admin, default: user)
 - `isActive` (Boolean, default: true)
 - `createdAt` (Date, auto)
 - `updatedAt` (Date, auto)

- Add indexes on `email` and `username`
- Add pre-save hook to hash password (bcrypt, 10 rounds)
- Add instance method `comparePassword(candidatePassword)`
- Add instance method `toJSON()` to remove password from responses
- Add static method `findByEmail(email)` with password selection
- Add virtual field `fullName` (`firstName + lastName`)

Journal Model (models/Journal.js)

- Create Journal schema with fields:
 - `userId` (ObjectId, ref: User, required, indexed)
 - `title` (String, required, 1-200 chars)
 - `content` (String, required, 1-50000 chars)
 - `mood` (String, enum: happy/sad/excited/anxious/calm/angry/neutral/grateful/tired/motivated, default: neutral)
 - `tags` (Array of Strings, lowercase, max 10 tags, 1-30 chars each)
 - `isFavorite` (Boolean, default: false)
 - `isPrivate` (Boolean, default: true)
 - `attachments` (Array of Objects):
 - `fileName` (String, required)
 - `fileUrl` (String, required, valid URL)
 - `fileType` (String, enum: allowed types)
 - `fileSize` (Number, max 5MB)
 - `location` (Object):
 - `type` (String, default: Point)
 - `coordinates` (Array of Numbers, [longitude, latitude])
 - `placeName` (String, max 100 chars)
 - `weather` (Object):
 - `condition` (String)
 - `temperature` (Number, -100 to 100)
 - `icon` (String)
 - `createdAt` (Date, auto)
 - `updatedAt` (Date, auto)
- Add compound indexes:
 - `{ userId: 1, createdAt: -1 }`
 - `{ userId: 1, tags: 1 }`
 - `{ userId: 1, mood: 1 }`

- `{ userId: 1, isFavorite: 1 }`
- Add text index on `title` and `content` for search
- Add pre-save hook to remove duplicate tags
- Add instance method `belongsToUser(userId)`
- Add static method `findByUser(userId, options)`
- Add static method `getMoodStats(userId)`

Category Model (models/Category.js)

- Create Category schema with fields:
- `userId` (ObjectId, ref: User, required, indexed)
 - `name` (String, required, 1-50 chars)
 - `color` (String, hex color, default: `#3b82f6`)
 - `icon` (String, max 50 chars)
 - `description` (String, max 200 chars)
 - `createdAt` (Date, auto)
 - `updatedAt` (Date, auto)

- Add compound unique index `{ userId: 1, name: 1 }`
- Add pre-save hook to check duplicate category names per user
- Add instance method `belongsToUser(userId)`
- Add static method `findByUser(userId)`

Journal Prompt Model (models/JournalPrompt.js)

- Create JournalPrompt schema with fields:
- `prompt` (String, required, unique)
 - `category` (String, enum: reflection/gratitude/goals/creativity/mindfulness/relationships/personal_growth, required)
 - `isActive` (Boolean, default: true)
 - `createdAt` (Date, auto)
 - `updatedAt` (Date, auto)
- Add index on `category`
- Add index on `isActive`
- Add static method `getRandomPrompt()`
- Add static method `getByCategory(category)`
-

3. Authentication APIs

Sign Up API (POST /api/auth/register)

- Validate input (username, email, password)
 - Username: 3-30 chars, alphanumeric with underscore/hyphen
 - Email: valid email format
 - Password: minimum 6 characters
- Check if email already exists
- Check if username already exists
- Hash password using bcrypt (10 rounds)
- Create new user in database
- Generate JWT token (7 days expiration)
- Return user data (without password) and token
- Handle errors:
 - 400: Validation errors
 - 409: User already exists
 - 500: Server error

Login API (POST /api/auth/login)

- Validate email and password input
- Find user by email (include password in query)
- Compare password with hashed password
- Check if user is active
- Generate JWT token
- Return user data (without password) and token
- Handle errors:
 - 400: Missing email or password
 - 401: Invalid credentials
 - 500: Server error

Get Current User (GET /api/auth/me)

- Protect route with authentication middleware
- Get user ID from JWT token
- Find user in database (exclude password)
- Return user data with preferences and profile
- Handle errors:
 - 401: Unauthorized (invalid/missing token)
 - 404: User not found

- 500: Server error

Update Profile (PUT /api/auth/profile)

- Protect route with authentication middleware
- Validate input (firstName, lastName, bio, avatar)
- Get user ID from JWT token
- Update user profile fields
- Update updatedAt timestamp
- Return updated user data
- Handle errors:
 - 400: Validation errors
 - 401: Unauthorized
 - 500: Server error

Update Preferences (PUT /api/auth/preferences)

- Protect route with authentication middleware
- Validate input (theme, timezone, reminderTime)
- Get user ID from JWT token
- Update user preferences
- Return updated preferences
- Handle errors:
 - 400: Validation errors
 - 401: Unauthorized
 - 500: Server error

Change Password (PUT /api/auth/password)

- Protect route with authentication middleware
- Validate input (currentPassword, newPassword)
- Get user ID from JWT token
- Find user with password field
- Verify current password
- Hash new password
- Update user password
- Return success message
- Handle errors:
 - 400: Validation errors
 - 401: Current password incorrect
 - 500: Server error

Logout API (POST /api/auth/logout)

- Client-side token removal (frontend handles this)
 - Optional: Implement token blacklist
 - Return success message
-

4. Journal APIs

Get All Journals (GET /api/journals)

- Protect route with authentication middleware
- Get user ID from JWT token
- Support query parameters:
 - `(page)` (default: 1)
 - `(limit)` (default: 10, max: 100)
 - `(mood)` (filter by mood)
 - `(tags)` (comma-separated, filter by tags)
 - `(search)` (search in title and content)
 - `(isFavorite)` (true/false)
 - `(startDate)` (filter from date)
 - `(endDate)` (filter to date)
 - `(sort)` (createdAt/-createdAt/updatedAt/-updatedAt)
- Query journals filtered by user ID and parameters
- Implement pagination
- Return journals array with metadata:
 - `(journals)` (array)
 - `(totalPages)` (number)
 - `(currentPage)` (number)
 - `(total)` (total count)
- Handle errors:
 - 400: Invalid query parameters
 - 401: Unauthorized
 - 500: Server error

Get Single Journal (GET /api/journals/:id)

- Protect route with authentication middleware
- Validate journal ID format
- Get user ID from JWT token

Find journal by ID and user ID

Return journal data

Handle errors:

- 400: Invalid ID format
- 401: Unauthorized
- 404: Journal not found or not owned by user
- 500: Server error

Create Journal (POST /api/journals)

Protect route with authentication middleware

Validate input:

- `title` (required, 1-200 chars)
- `content` (required, 1-50000 chars)
- `mood` (optional, valid enum value)
- `tags` (optional, array, max 10 tags)
- `isFavorite` (optional, boolean)
- `isPrivate` (optional, boolean, default: true)
- `location` (optional, object)
- `weather` (optional, object)
- `attachments` (optional, array)

Get user ID from JWT token

Create new journal with user reference

Save to database

Return created journal with 201 status

Handle errors:

- 400: Validation errors
- 401: Unauthorized
- 500: Server error

Update Journal (PUT /api/journals/:id)

Protect route with authentication middleware

Validate journal ID format

Validate input fields (same as create)

Get user ID from JWT token

Find journal by ID and user ID

Verify user owns the journal

Update journal fields

- Update `(updatedAt)` timestamp
- Save to database
- Return updated journal
- Handle errors:
 - 400: Validation errors
 - 401: Unauthorized
 - 404: Journal not found or not owned by user
 - 500: Server error

Delete Journal (DELETE /api/journals/:id)

- Protect route with authentication middleware
- Validate journal ID format
- Get user ID from JWT token
- Find journal by ID and user ID
- Verify user owns the journal
- Delete journal from database
- Return success message with 200 status
- Handle errors:
 - 400: Invalid ID format
 - 401: Unauthorized
 - 404: Journal not found or not owned by user
 - 500: Server error

Toggle Favorite (PATCH /api/journals/:id/favorite)

- Protect route with authentication middleware
- Validate journal ID format
- Get user ID from JWT token
- Find journal by ID and user ID
- Toggle `(isFavorite)` field
- Save to database
- Return updated journal
- Handle errors:
 - 400: Invalid ID format
 - 401: Unauthorized
 - 404: Journal not found
 - 500: Server error

Get Mood Statistics (GET /api/journals/stats/mood)

- Protect route with authentication middleware
- Get user ID from JWT token
- Use aggregation pipeline to group by mood
- Calculate count for each mood
- Return statistics object with mood counts
- Handle errors:
 - 401: Unauthorized
 - 500: Server error

Get Journal Analytics (GET /api/journals/stats/analytics)

- Protect route with authentication middleware
- Get user ID from JWT token
- Calculate statistics:
 - Total journals count
 - Journals this week
 - Journals this month
 - Most used tags (top 10)
 - Mood distribution
 - Favorite count
 - Average journals per week
- Return analytics object
- Handle errors:
 - 401: Unauthorized
 - 500: Server error

Search Journals (GET /api/journals/search)

- Protect route with authentication middleware
- Get search query from query parameters
- Use text index for full-text search
- Search in title and content
- Filter by user ID
- Support pagination
- Return matching journals
- Handle errors:
 - 400: Missing search query
 - 401: Unauthorized

- 500: Server error
-

5. Category APIs

Get All Categories (GET /api/categories)

- Protect route with authentication middleware
- Get user ID from JWT token
- Find all categories for user
- Sort by name (alphabetically)
- Return categories array
- Handle errors:
 - 401: Unauthorized
 - 500: Server error

Get Single Category (GET /api/categories/:id)

- Protect route with authentication middleware
- Validate category ID format
- Get user ID from JWT token
- Find category by ID and user ID
- Return category data
- Handle errors:
 - 400: Invalid ID format
 - 401: Unauthorized
 - 404: Category not found or not owned by user
 - 500: Server error

Create Category (POST /api/categories)

- Protect route with authentication middleware
- Validate input:
 - `name` (required, 1-50 chars, unique per user)
 - `color` (optional, valid hex color, default: `#3b82f6`)
 - `icon` (optional, max 50 chars)
 - `description` (optional, max 200 chars)
- Get user ID from JWT token
- Check if category name already exists for user
- Create new category with user reference
- Save to database

Return created category with 201 status

Handle errors:

- 400: Validation errors
- 401: Unauthorized
- 409: Category name already exists
- 500: Server error

Update Category (PUT /api/categories/:id)

Protect route with authentication middleware

Validate category ID format

Validate input fields (same as create)

Get user ID from JWT token

Find category by ID and user ID

Verify user owns the category

Check if new name conflicts with existing category

Update category fields

Save to database

Return updated category

Handle errors:

- 400: Validation errors
- 401: Unauthorized
- 404: Category not found or not owned by user
- 409: Category name already exists
- 500: Server error

Delete Category (DELETE /api/categories/:id)

Protect route with authentication middleware

Validate category ID format

Get user ID from JWT token

Find category by ID and user ID

Verify user owns the category

Delete category from database

Return success message

Handle errors:

- 400: Invalid ID format
- 401: Unauthorized
- 404: Category not found or not owned by user
- 500: Server error

6. Prompt APIs

Get All Prompts (GET /api/prompts)

- Public route (no authentication required)
- Query only active prompts (`(isActive: true)`)
- Support query parameter:
 - `category` (optional, filter by category)
- Return prompts array
- Handle errors:
 - 500: Server error

Get Random Prompt (GET /api/prompts/random)

- Public route (no authentication required)
- Support query parameter:
 - `category` (optional, filter by category)
- Get random active prompt
- Use MongoDB aggregation `($sample)`
- Return single prompt object
- Handle errors:
 - 404: No prompts found
 - 500: Server error

Get Prompts by Category (GET /api/prompts/category/:category)

- Public route (no authentication required)
- Validate category parameter (valid enum value)
- Find active prompts by category
- Return prompts array
- Handle errors:
 - 400: Invalid category
 - 404: No prompts found for category
 - 500: Server error

Create Prompt (POST /api/prompts) - Admin Only

- Protect route with authentication middleware
- Check if user is admin
- Validate input:

- `(prompt)` (required, unique)
- `(category)` (required, valid enum value)
- `(isActive)` (optional, boolean, default: true)

Create new prompt

Save to database

Return created prompt with 201 status

Handle errors:

- 400: Validation errors
- 401: Unauthorized
- 403: Forbidden (not admin)
- 409: Prompt already exists
- 500: Server error

Update Prompt (PUT /api/prompts/:id) - Admin Only

Protect route with authentication middleware

Check if user is admin

Validate prompt ID format

Validate input fields

Find and update prompt

Return updated prompt

Handle errors:

- 400: Validation errors
- 401: Unauthorized
- 403: Forbidden (not admin)
- 404: Prompt not found
- 500: Server error

Delete Prompt (DELETE /api/prompts/:id) - Admin Only

Protect route with authentication middleware

Check if user is admin

Validate prompt ID format

Find and delete prompt

Return success message

Handle errors:

- 400: Invalid ID format
- 401: Unauthorized
- 403: Forbidden (not admin)

- 404: Prompt not found
 - 500: Server error
-

7. Security Implementation

Authentication Middleware (middleware/auth.js)

- Extract JWT token from Authorization header (`(Bearer <token>)`)
- Check if token exists
- Verify JWT token using secret
- Extract user ID from token payload
- Find user in database
- Attach user object to request (`(req.user)`)
- Attach user ID to request (`(req.userId)`)
- Call `next()` to continue
- Handle errors:
 - No token provided
 - Invalid token
 - Token expired
 - User not found
 - User inactive

Admin Middleware (middleware/admin.js)

- Check if user is authenticated (run auth middleware first)
- Check if user role is 'admin'
- Call `next()` if admin
- Return 403 Forbidden if not admin

Validation Middleware (middleware/validation.js)

- Create validation schemas using express-validator
- Validate registration input
- Validate login input
- Validate journal creation/update
- Validate category creation/update
- Validate prompt creation/update
- Return validation errors with 400 status

Error Handler Middleware (middleware/errorHandler.js)

- Catch all errors from routes
- Handle Mongoose validation errors
- Handle MongoDB duplicate key errors (11000)
- Handle JWT errors
- Handle custom errors
- Return consistent error format:

```
json
```

```
{  
  "success": false,  
  "message": "Error message",  
  "errors": [] // optional array of errors  
}
```

- Log errors in development
- Hide sensitive error details in production
- Return appropriate HTTP status codes

Password Security

- Use bcryptjs for password hashing
- Set salt rounds to 10-12
- Hash password in pre-save hook
- Never return password in API responses
- Use `select: false` in schema
- Validate password strength:
 - Minimum 6 characters
 - Optional: Require uppercase, lowercase, number, special char

JWT Token Security

- Store JWT secret in environment variable
- Set appropriate expiration time (7 days for regular, 15 min for refresh)
- Include user ID and email in token payload
- Sign tokens with HS256 algorithm
- Verify tokens on protected routes
- Handle token expiration gracefully
- Optional: Implement refresh token mechanism
- Optional: Implement token blacklist for logout

Input Validation

- Validate all user inputs
- Sanitize inputs to prevent XSS
- Validate email format with regex
- Validate URL format for avatar and file URLs
- Validate hex color codes
- Validate enum values (mood, theme, category)
- Set maximum length for text fields
- Validate ObjectId format for MongoDB IDs
- Trim whitespace from string inputs
- Convert appropriate fields to lowercase (email, username)

Rate Limiting

- Install express-rate-limit package
- Implement rate limiting for:
 - Login: 5 attempts per 15 minutes per IP
 - Registration: 3 attempts per hour per IP
 - All API requests: 100 per 15 minutes per user
 - Password reset: 3 attempts per hour per email
- Return 429 Too Many Requests when limit exceeded
- Add headers to show rate limit status

CORS Configuration

- Configure CORS to allow frontend origin
- Set allowed methods (GET, POST, PUT, DELETE, PATCH)
- Set allowed headers (Content-Type, Authorization)
- Enable credentials if using cookies
- Restrict origins in production

Additional Security Headers

- Install helmet package
- Set security headers:
 - X-Content-Type-Options: nosniff
 - X-Frame-Options: DENY
 - X-XSS-Protection: 1; mode=block
 - Strict-Transport-Security (HTTPS)
- Configure Content Security Policy (CSP)

8. Frontend Integration

API Configuration

- Create API base URL constant (`(http://localhost:5000/api)`)
- Set up axios instance with base URL
- Configure request interceptor to add JWT token:

```
javascript
```

```
config.headers.Authorization = 'Bearer ${token}';
```

- Configure response interceptor for error handling
- Handle 401 errors (redirect to login)
- Handle network errors

Authentication Functions

- `register(username, email, password)` → POST /api/auth/register
- `login(email, password)` → POST /api/auth/login
- `getCurrentUser()` → GET /api/auth/me
- `updateProfile(profileData)` → PUT /api/auth/profile
- `updatePreferences(preferences)` → PUT /api/auth/preferences
- `changePassword(currentPassword, newPassword)` → PUT /api/auth/password
- `logout()` → Clear token from localStorage

Journal Functions

- `getJournals(filters)` → GET /api/journals?page=1&limit=10&...
- `getJournal(id)` → GET /api/journals/:id
- `createJournal(journalData)` → POST /api/journals
- `updateJournal(id, journalData)` → PUT /api/journals/:id
- `deleteJournal(id)` → DELETE /api/journals/:id
- `toggleFavorite(id)` → PATCH /api/journals/:id/favorite
- `getMoodStats()` → GET /api/journals/stats/mood
- `getAnalytics()` → GET /api/journals/stats/analytics
- `searchJournals(query)` → GET /api/journals/search?q=...

Category Functions

- `getCategories()` → GET /api/categories
- `getCategory(id)` → GET /api/categories/:id
- `createCategory(categoryData)` → POST /api/categories
- `updateCategory(id, categoryData)` → PUT /api/categories/:id

`(deleteCategory(id))` → DELETE /api/categories/:id

Prompt Functions

- `(getPrompts(category?))` → GET /api/prompts?category=...
- `(getRandomPrompt(category?))` → GET /api/prompts/random?category=...
- `(getPromptsByCategory(category))` → GET /api/prompts/category/:category

Token Management

- Store JWT token in localStorage:

```
javascript
```

```
localStorage.setItem('token', token);
```

- Retrieve token for API requests:

```
javascript
```

```
const token = localStorage.getItem('token');
```

- Include token in Authorization header:

```
javascript
```

```
headers: { Authorization: `Bearer ${token}` }
```

- Clear token on logout:

```
javascript
```

```
localStorage.removeItem('token');
```

- Check token expiration
- Redirect to login if token invalid/expired
- Optional: Implement token refresh mechanism

Error Handling

- Display user-friendly error messages
- Handle network errors (offline, timeout)
- Handle 400 errors (validation errors):
 - Display field-specific errors
- Handle 401 errors (unauthorized):
 - Clear token

- Redirect to login page

Handle 403 errors (forbidden):

- Show access denied message

Handle 404 errors (not found):

- Show not found message

Handle 409 errors (conflict):

- Show duplicate entry message

Handle 429 errors (rate limit):

- Show "too many requests" message

Handle 500 errors (server error):

- Show generic error message

- Log error for debugging

Loading States

Show loading spinner during API calls

Disable submit buttons during form submission

Show skeleton screens for content loading

Display progress indicators for file uploads

Show success notifications on successful operations

Show error notifications on failed operations

Implement optimistic UI updates where appropriate

State Management

Store user data in global state (Context, Redux, Zustand)

Store authentication status

Store journals list

Store categories list

Implement pagination state

Implement filter state

Cache API responses where appropriate

Invalidate cache on data mutations

9. Database Seeding

Seed Journal Prompts (utils/seedPrompts.js)

Create seed script

Add prompts for each category:

- **Reflection:** 10+ prompts
- **Gratitude:** 10+ prompts
- **Goals:** 10+ prompts
- **Creativity:** 10+ prompts
- **Mindfulness:** 10+ prompts
- **Relationships:** 10+ prompts
- **Personal Growth:** 10+ prompts

- Check if prompts already exist before inserting
 Set all prompts as active
 Create npm script: `npm run seed`
 Log success/error messages

Example Prompts

javascript

```
const prompts = [
  // Reflection
  { prompt: "What was the most challenging moment of your day?", category: "reflection" },
  { prompt: "What did you learn about yourself this week?", category: "reflection" },

  // Gratitude
  { prompt: "List three things you're grateful for today", category: "gratitude" },
  { prompt: "Who made your day better, and how?", category: "gratitude" },

  // Goals
  { prompt: "What is one thing you want to accomplish this month?", category: "goals" },
  { prompt: "What steps can you take tomorrow toward your goals?", category: "goals" },

  // Creativity
  { prompt: "If you could travel anywhere tomorrow, where would you go?", category: "creativity" },
  { prompt: "Describe your ideal day from start to finish", category: "creativity" },

  // Mindfulness
  { prompt: "How are you feeling right now, both physically and emotionally?", category: "mindfulness" },
  { prompt: "What can you let go of today?", category: "mindfulness" },

  // And more...
];


```

10. Testing

Unit Tests