Dreams Uncharted Funding System

Complete Hybrid Implementation Guide

Version: 2.0
Date: January 29, 2025
Platform: Node.js + React Hybrid Architecture

Table of Contents

Overview

The Dreams Uncharted Funding System is a comprehensive crowdfunding solution that allows creators to raise money for new comic pages, chapters, and series. The system uses a hybrid architecture that combines your existing HTML structure with modern React components for funding features.

Key Features

- **Funding Goals**: Creators can set funding targets for new content
- **Donation Processing**: Secure payments via Stripe and PayPal
- **Reward Tiers**: Different perks for different donation amounts
- **Progress Tracking**: Real-time funding goal progress
- **Admin Dashboard**: Creator management interface
- **User Authentication**: Google OAuth integration
- **Responsive Design**: Mobile and desktop compatible

Technology Stack

Backend: - Node.js with Express.js - MongoDB with Mongoose - Google OAuth (Passport.js) - Stripe & PayPal payment processing - JWT authentication

Frontend: - React 18 components - Stripe React integration - Webpack build system - CSS with cosmic theme - HTML integration helpers

---

Project Structure

Your final project structure should look like this:

```
GeorgeTheDragonSlayer-Monarch.github.io/
├── admin/                          # Admin HTML pages
│   ├── assets/
│   │   └── js/
│   │       └── funding-embed.js    # Integration helper
│   ├── content.html                # Enhanced with funding
│   ├── dashboard.html              # Enhanced with funding
│   └── ...
├── public/                         # Public HTML pages
│   ├── assets/
│   │   └── js/
│   │       └── funding-embed.js    # Integration helper
│   ├── comics.html                 # Enhanced with funding
│   ├── series.html                 # Enhanced with funding
│   └── ...
├── server/                         # Backend API
│   ├── models/
│   │   ├── Content.js
│   │   ├── Donation.js             # NEW
│   │   ├── FundingGoal.js          # NEW
│   │   ├── Series.js
│   │   └── User.js
│   ├── routes/
│   │   ├── auth.js
│   │   ├── content.js
│   │   ├── funding.js              # NEW
│   │   └── payments.js             # NEW
│   ├── app.js                      # Updated
│   ├── package.json                # Updated
│   └── .env                        # Updated
└── funding-components/             # NEW - React components
    ├── src/
    │   ├── components/
    │   ├── embeds/
    │   ├── hooks/
    │   ├── services/
    │   ├── utils/
    │   ├── styles/
    │   └── index.js
    ├── dist/                       # Built components
    ├── package.json
    └── webpack.config.js
```

Backend Implementation

Database Models

FundingGoal Model

```
 const FundingGoalSchema = new Schema({
  title: { type: String, required: true },
  description: { type: String, required: true },
  targetAmount: { type: Number, required: true },
  currentAmount: { type: Number, default: 0 },
  creator: { type: Schema.Types.ObjectId, ref: 'User', required: true },
  seriesId: { type: Schema.Types.ObjectId, ref: 'Series' },
  fundingType: {
    type: String,
    enum: ['chapter', 'episode', 'page', 'artwork', 'bonus-content', 'series-
continuation'],
    required: true
  },
  status: {
    type: String,
    enum: ['active', 'completed', 'paused', 'cancelled'],
    default: 'active'
  },
  deadline: { type: Date },
  rewardTiers: [{
    amount: { type: Number, required: true },
    title: { type: String, required: true },
    description: { type: String, required: true },
    maxBackers: { type: Number },
    currentBackers: { type: Number, default: 0 }
  }],
  settings: {
    allowAnonymousDonations: { type: Boolean, default: true },
    showDonorList: { type: Boolean, default: true },
    sendUpdates: { type: Boolean, default: true }
  }
}, { timestamps: true });
```

Donation Model

```
 const DonationSchema = new Schema({
  fundingGoal: { type: Schema.Types.ObjectId, ref: 'FundingGoal', required:
true },
  donor: { type: Schema.Types.ObjectId, ref: 'User' },
  amount: { type: Number, required: true },
  rewardTier: {
    amount: Number,
    title: String,
    description: String
  },
  paymentMethod: {
    type: String,
    enum: ['stripe', 'paypal'],
    required: true
  },
  paymentId: { type: String, required: true },
  status: {
    type: String,
    enum: ['pending', 'completed', 'failed', 'refunded'],
    default: 'pending'
  },
  isAnonymous: { type: Boolean, default: false },
  donorEmail: { type: String },
  donorName: { type: String },
  message: { type: String },
  fees: {
    processingFee: { type: Number, default: 0 },
    platformFee: { type: Number, default: 0 }
  }
}, { timestamps: true });
```

API Endpoints

Funding Routes (`/api/funding`)

- `GET /goals` - Get all funding goals
- `GET /goals/:id` - Get specific funding goal
- `POST /goals` - Create new funding goal
- `PUT /goals/:id` - Update funding goal
- `DELETE /goals/:id` - Delete funding goal
- `GET /creator/:creatorId/goals` - Get creator's funding goals
- `GET /series/:seriesId/goals` - Get series funding goals
- `GET /stats/:creatorId` - Get creator funding statistics

Payment Routes (`/api/payments`)

- `POST /stripe/create-payment-intent` - Create Stripe payment
- `POST /stripe/webhook` - Stripe webhook handler
- `POST /paypal/create-payment` - Create PayPal payment
- `POST /paypal/execute-payment` - Execute PayPal payment
- `POST /paypal/ipn` - PayPal IPN handler

Frontend Components

Core Components

FundingGoalCard

Displays individual funding goals with progress bars, reward tiers, and donation buttons.

DonationModal

Modal interface for making donations with payment form integration.

PaymentForm

Stripe payment form with card input and validation.

ProgressBar

Animated progress bar showing funding goal completion.

RewardTiers

Interactive reward tier selection component.

Embed Components

FundingGoalsEmbed

Shows funding goals on public pages (series, comics).

DonationButtonEmbed

Simple donation button for any page.

AdminFundingEmbed

Admin dashboard for managing funding goals.

CreateFundingGoalEmbed

Form for creating new funding goals.

Hooks & Services

useFunding Hook

```
// Get funding goals
const { data, isLoading } = useFundingGoals({ limit: 5 });

// Get creator's goals
const { data } = useCreatorFundingGoals(creatorId);

// Create funding goal
const createMutation = useCreateFundingGoal();

// Process donation
const donateMutation = useProcessDonation();
```

fundingAPI Service

```
// API calls
fundingAPI.getFundingGoals()
fundingAPI.createFundingGoal(goalData)
fundingAPI.processDonation(donationData)
fundingAPI.getFundingStats(creatorId)
```

File Placement Guide

Step 1: Backend Files

Place these files in your `server/` directory:

```
server/
├── models/
│   ├── FundingGoal.js      ← Copy from provided files
│   └── Donation.js         ← Copy from provided files
├── routes/
│   ├── funding.js          ← Copy from provided files
│   └── payments.js         ← Copy from provided files
├── app.js                  ← Replace with updated version
└── package.json            ← Replace with updated version
```

Step 2: Frontend Components

Create and populate the `funding-components/` directory:

```
funding-components/
├── src/
│   ├── components/
│   │   ├── FundingGoalCard.jsx
│   │   ├── DonationModal.jsx
│   │   ├── PaymentForm.jsx
│   │   ├── ProgressBar.jsx
│   │   └── RewardTiers.jsx
│   ├── embeds/
│   │   ├── FundingGoalsEmbed.jsx
│   │   ├── DonationButtonEmbed.jsx
│   │   ├── AdminFundingEmbed.jsx
│   │   └── CreateFundingGoalEmbed.jsx
│   ├── hooks/
│   │   └── useFunding.js
│   ├── services/
│   │   └── fundingAPI.js
│   ├── utils/
│   │   └── formatCurrency.js
│   ├── styles/
│   │   └── funding-components.css
│   └── index.js
├── package.json
└── webpack.config.js
```

Step 3: Integration Files

Copy `funding-embed.js` to: - `public/assets/js/funding-embed.js` - `admin/assets/js/funding-embed.js`

Installation Steps

Step 1: Install Backend Dependencies

```
cd server/
npm install stripe nodemailer node-cron express-rate-limit helmet express-validator
```

Step 2: Create Funding Components

```
# Create the funding-components directory
mkdir funding-components
cd funding-components

# Initialize with package.json
npm init -y

# Install dependencies
npm install react react-dom @stripe/stripe-js @stripe/react-stripe-js
npm install --save-dev @babel/core @babel/preset-react @babel/preset-env
webpack webpack-cli babel-loader css-loader style-loader
```

Step 3: Copy Component Files

Copy all the provided React component files to their respective directories in `funding-components/src/`.

Step 4: Build Components

```
cd funding-components/
npm run build
```

This creates `dist/funding-components.js` which your HTML pages will load.

Configuration

Environment Variables

Add these to your `server/.env` file:

```
# Database
MONGODB_URI=mongodb://localhost:27017/dreams-uncharted

# Google OAuth
GOOGLE_CLIENT_ID=your_google_client_id
GOOGLE_CLIENT_SECRET=your_google_client_secret

# Stripe Configuration
STRIPE_PUBLISHABLE_KEY=pk_test_your_stripe_publishable_key
STRIPE_SECRET_KEY=sk_test_your_stripe_secret_key
STRIPE_WEBHOOK_SECRET=whsec_your_webhook_secret

# PayPal Configuration
PAYPAL_EMAIL=your_paypal_email@example.com
PAYPAL_CLIENT_ID=your_paypal_client_id
PAYPAL_CLIENT_SECRET=your_paypal_client_secret

# URLs
CLIENT_URL=http://localhost:3000
SERVER_URL=http://localhost:3001

# Session
SESSION_SECRET=your-super-secret-session-key

# Email (for notifications)
EMAIL_SERVICE=gmail
EMAIL_USER=your_email@gmail.com
EMAIL_PASS=your_app_password
```

Stripe Setup

1. **Create Stripe Account**: Sign up at stripe.com
2. **Get API Keys**: Dashboard → Developers → API Keys
3. **Set up Webhooks**:
4. Endpoint: `http://localhost:3001/api/payments/stripe/webhook`
5. Events: `payment_intent.succeeded`, `payment_intent.payment_failed`
6. **Copy Webhook Secret**: Use in `STRIPE_WEBHOOK_SECRET`

PayPal Setup

1. **Create PayPal Developer Account**: developer.paypal.com
2. **Create App**: Get Client ID and Secret
3. **Configure IPN**: Set IPN URL to `http://localhost:3001/api/payments/paypal/ipn`

HTML Integration

Method 1: Data Attributes (Easiest)

Add funding components to any HTML page using data attributes:

```
<!DOCTYPE html>
<html>
<head>
    <title>Series Page</title>
    <!-- React Dependencies -->
    <script crossorigin
src="https://unpkg.com/react@18/umd/react.production.min.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.production.min.js"></script>
    <script src="https://js.stripe.com/v3/"></script>

    <!-- Funding Components -->
    <script src="../funding-components/dist/funding-components.js"></script>
    <script src="assets/js/funding-embed.js"></script>

    <script>
        window.STRIPE_PUBLISHABLE_KEY = 'pk_test_your_key';
        window.currentUser = {
            id: 'user-id',
            name: 'User Name',
            role: 'user'
        };
    </script>
</head>
<body>
    <!-- Your existing content -->

    <!-- Funding Goals -->
    <div data-funding-goals
         data-series-id="your-series-id"
         data-limit="3">
    </div>

    <!-- Donation Button -->
    <div data-donation-button
         data-goal-id="funding-goal-id"
         data-button-text="Support This Comic">
    </div>

    <!-- Admin Funding (for admin pages) -->
    <div data-admin-funding
         data-creator-id="creator-user-id">
    </div>
</body>
</html>
```

Method 2: JavaScript Functions

For more control, use JavaScript functions:

```
<div id="funding-container"></div>

<script>
document.addEventListener('DOMContentLoaded', function() {
    // Render funding goals
    embedFundingGoals('funding-container', {
        seriesId: 'space-adventures-123',
        limit: 2
    });

    // Or use the full API
    window.DreamsUnchartedFunding.embed.fundingGoals('funding-container', {
        seriesId: 'space-adventures-123',
        limit: 2,
        showAll: false
    });
});
</script>
```

Integration Examples

Series Page Enhancement

```
<!-- Add to your existing series.html -->
<section class="series-funding">
    <h2>Support This Series</h2>
    <div data-funding-goals data-series-id="{{series.id}}" data-limit="2">
</div>
</section>
```

Comic Reader Enhancement

```
<!-- Add to your comic reader -->
<div class="support-creator">
    <h3>Enjoying this chapter?</h3>
    <div data-donation-button
         data-goal-id="{{activeGoal.id}}"
         data-button-text="Support Next Chapter">
    </div>
</div>
```

Admin Dashboard Enhancement

```
<!-- Add to admin content management -->
<section class="funding-management">
    <h2>Funding Goals</h2>
    <div data-admin-funding data-creator-id="{{user.id}}"></div>
</section>

<section class="create-funding">
    <div data-create-funding-goal data-creator-id="{{user.id}}"></div>
</section>
```

Testing & Verification

Backend Testing

1. **Start the Server** `bash cd server/ npm start`
2. **Check Health Endpoint** `bash curl http://localhost:3001/health`
3. **Test Funding API** `bash curl http://localhost:3001/api/funding/goals`

Frontend Testing

1. **Build Components** `bash cd funding-components/ npm run build`
2. **Check Build Output** Verify `dist/funding-components.js` exists and is not empty.
3. **Test HTML Integration** Open any HTML page with funding components and check browser console for errors.

Payment Testing

Use Stripe test cards: - **Success**: `4242424242424242` - **Decline**: `4000000000000002` - **3D Secure**: `4000002500003155`

Database Verification

Check MongoDB collections:

```
// In MongoDB shell
db.fundinggoals.find()
db.donations.find()
```

Troubleshooting

Common Issues

1. "Module not found" errors

Solution: Ensure all files are in correct locations and dependencies are installed.

```
cd server/ && npm install
cd funding-components/ && npm install
```

2. Empty dist folder

Solution: Build the React components.

```
cd funding-components/
npm run build
```

3. CORS errors

Solution: Check `CLIENT_URL` in server `.env` matches your frontend URL.

4. Stripe errors

Solution: Verify API keys are correct and for test mode. - Publishable key starts with `pk_test_` - Secret key starts with `sk_test_`

5. Components not rendering

Solution: Check browser console for JavaScript errors and ensure React is loaded before funding components.

6. Database connection errors

Solution: Verify MongoDB is running and `MONGODB_URI` is correct.

Debug Steps

1. **Check Server Logs**: Look for error messages in terminal
2. **Browser Console**: Check for JavaScript errors
3. **Network Tab**: Verify API calls are being made
4. **Database**: Check if data is being saved correctly

Support Resources

- **Stripe Documentation**: stripe.com/docs
- **React Documentation**: reactjs.org
- **MongoDB Documentation**: docs.mongodb.com

Conclusion

The Dreams Uncharted Funding System provides a complete crowdfunding solution that integrates seamlessly with your existing HTML structure. The hybrid approach allows you to add modern React functionality without rebuilding your entire frontend.

Key Benefits

- **No Code Rewrite**: Keep your existing HTML pages
- **Modern Features**: React components with real-time updates
- **Secure Payments**: Stripe and PayPal integration
- **Mobile Responsive**: Works on all devices
- **Easy Integration**: Simple data attributes or JavaScript functions
- **Scalable**: Add funding to any page as needed

Next Steps

1. Complete the installation following this guide
2. Test the system with Stripe test cards
3. Customize the styling to match your brand
4. Add funding components to your key pages
5. Set up production environment with live payment keys

The system is designed to grow with your platform and can be extended with additional features as needed.

Dreams Uncharted Funding System v2.0
Complete Implementation Guide