Corn Job App - Full Stack Project Report

Project Overview

A full-stack application built to manage cron jobs and webhooks, featuring a modern web interface and robust backend API.

Technical Stack

Frontend (Next.js)

- **Framework**: Next.js 15.1.6
- **Key Dependencies**:
  - @tanstack/react-query: For data fetching and state management
  - axios: HTTP client for API requests
  - react-hook-form: Form handling and validation
  - TailwindCSS: Styling and UI components

Backend (NestJS)

- **Framework**: NestJS 11.0
- **Key Dependencies**:
  - @nestjs/mongoose: MongoDB integration
  - @nestjs/schedule: Cron job scheduling
  - @nestjs/throttler: Rate limiting
  - class-validator: Input validation

Architecture

Frontend Architecture

1. **API Layer**:

```
// frontend/app/api/api.js
const api = axios.create({
  baseURL: process.env.NEXT_PUBLIC_API_URL,
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json'
  }
});
```

2. **Components Structure**:
   - Navigation: Global navigation component
   - QueryProvider: React Query configuration
   - Pages:
     - Home: Cron jobs listing
     - Create: New job creation
     - Edit: Job modification
     - Webhooks: Webhook history
3. **Features**:
   - CRUD operations for cron jobs
   - Real-time form validation
   - Responsive design
   - Dark mode support
   - Loading states
   - Error handling

Backend Architecture

1. **Module Structure**:
   - CronJob Module: Manages cron job operations

- Webhook Module: Handles webhook events
- Database Module: MongoDB connection and models
- Seed Module: Initial data population

2. **Key Features**:

- RESTful API endpoints
- MongoDB integration
- Cron job scheduling
- Webhook processing
- Input validation
- Error handling
- Rate limiting

API Endpoints

Cron Jobs

GET /cron-jobs - List all jobs
GET /cron-jobs/:id - Get single job
POST /cron-jobs - Create job
PUT /cron-jobs/:id - Update job
DELETE /cron-jobs/:id - Delete job

Webhooks

GET /webhooks - List webhook history
POST /webhooks - Receive webhook data

Database Schema

CronJob Model

```
interface CronJob {
  name: string;
  schedule: string;
  link: string;
  apiKey: string;
  startDate: Date;
  isActive: boolean;
  history: Array<{
    triggeredAt: Date;
    response: any;
    status: string;
  }>;
}
```

Webhook Model

```
interface Webhook {
  cronJobId: string;
  data: any;
  createdAt: Date;
}
```

Deployment

Frontend Deployment (Vercel)

```
# Environment Variables
NEXT_PUBLIC_API_URL=https://your-backend-url.com

# Deploy Command
```

vercel

Backend Deployment (Mau/AWS)

```
# Environment Variables
MONGODB_URI=mongodb://your-mongodb-uri
FRONTEND_URL=https://your-frontend-url.com
PORT=3000

# Deploy Command
mau deploy
```

Security Measures

1. **API Security**:
   - Rate limiting
   - Input validation
   - CORS configuration
   - API key protection
2. **Data Security**:
   - Encrypted API keys
   - Secure MongoDB connection
   - Request validation

Performance Optimizations

1. **Frontend**:
   - React Query caching
   - Optimized build size
   - Lazy loading
   - Image optimization
2. **Backend**:
   - Database indexing
   - Request caching
   - Rate limiting
   - Efficient queries

Error Handling

1. **Frontend**:

```
api.interceptors.response.use(
  response => response,
  error => {
    console.error('API Error:', error);
    return Promise.reject(error);
  }
);
```

2. **Backend**:

```
try {
  // Operation logic
} catch (error) {
  throw new HttpException({
    status: HttpStatus.INTERNAL_SERVER_ERROR,
    error: 'Something went wrong'
  }, HttpStatus.INTERNAL_SERVER_ERROR);
}
```

Future Improvements

1. **Features**:
- Authentication/Authorization
- Job execution history
- Advanced scheduling options
- Email notifications
- Webhook retry logic
2. **Technical**:
- Unit test coverage
- E2E testing
- Performance monitoring
- Automated deployment
- Documentation

Testing Strategy

1. **Frontend Tests**:
- Component testing
- Integration testing
- E2E testing with Cypress
2. **Backend Tests**:
- Unit tests
- Integration tests
- E2E API tests

Project Statistics

- Frontend Components: ~10
- Backend Endpoints: ~7
- Database Models: 2
- Total Dependencies: ~25
- Development Time: Estimated 2-3 weeks

This fullstack application demonstrates modern web development practices, scalable architecture, and maintainable code structure using Next.js and NestJS frameworks.