

# SaaS MVP Development Tutorial

---

## Table of Contents

---

1. [Project Overview](#)
2. [Planning Phase](#)
3. [Architecture Design](#)
4. [Development Setup](#)
5. [Core Feature Implementation](#)
6. [Testing and Quality Assurance](#)
7. [Deployment and Launch](#)
8. [Post-Launch Optimization](#)

## Project Overview

---

In this comprehensive tutorial, we'll build **TaskMaster Pro** - a complete SaaS task management application from concept to deployment using the S.P.A. ecosystem. This tutorial demonstrates the full power of the S.P.A. System by creating a production-ready MVP in record time.

### What We'll Build

- **Frontend:** Modern React 18 + TypeScript dashboard
- **Backend:** Node.js + Express API with PostgreSQL
- **Authentication:** JWT-based user management
- **Real-time Features:** WebSocket notifications
- **Payment Integration:** Stripe subscription billing
- **Deployment:** Automated CI/CD to Netlify + Railway

### Key Features

- User registration and authentication
- Project and task management
- Team collaboration
- Real-time notifications
- Subscription billing
- Analytics dashboard
- Mobile-responsive design

### Learning Outcomes

By completing this tutorial, you'll master:

- S.P.A. System workflow and agent coordination
- Full-stack TypeScript development
- Modern React patterns and state management
- API design and database modeling
- Authentication and authorization
- Payment processing integration

- Automated testing and deployment
- Performance optimization techniques

## Planning Phase

---

### Step 1: Requirements Gathering with Product Owner Agent

Let's start by using the S.P.A. Product Owner agent to define our requirements:

#### Prompt for Product Owner Agent:

```
Act as Product Owner for TaskMaster Pro - a SaaS task management platform. Define:  
1. User personas and target market  
2. Core user stories with acceptance criteria  
3. MVP feature prioritization  
4. Success metrics and KPIs
```

#### Expected Output:

```

# TaskMaster Pro - Product Requirements

## User Personas
### Primary: Small Team Leaders (5-20 people)
- Pain: Scattered task management across multiple tools
- Goal: Centralized, simple task tracking with team visibility
- Budget: $10-50/month per team

### Secondary: Freelancers and Solo Entrepreneurs
- Pain: Need professional task management without complexity
- Goal: Organized workflow with client project separation
- Budget: $5-15/month individual

## Core User Stories

### Epic 1: User Management
- **US-001**: As a user, I can register with email/password
- **US-002**: As a user, I can log in securely
- **US-003**: As a user, I can reset my password
- **US-004**: As a user, I can manage my profile

### Epic 2: Project Management
- **US-005**: As a user, I can create projects
- **US-006**: As a user, I can invite team members
- **US-007**: As a user, I can set project permissions

### Epic 3: Task Management
- **US-008**: As a user, I can create tasks with details
- **US-009**: As a user, I can assign tasks to team members
- **US-010**: As a user, I can track task progress
- **US-011**: As a user, I can set due dates and priorities

## MVP Features (Priority Order)
1. User authentication and profiles
2. Basic project creation and management
3. Task CRUD operations
4. Simple team collaboration
5. Basic dashboard and reporting
6. Subscription billing integration

```

## Step 2: Market Research with Strategic Business Advisor

Prompt for Strategic Business Advisor:

```

Analyze the task management SaaS market for TaskMaster Pro. Provide:
1. Competitive landscape analysis
2. Pricing strategy recommendations
3. Go-to-market strategy
4. Revenue projections for first year

```

## Step 3: Technical Planning with Technical Architecture Consultant

Prompt for Technical Architecture Consultant:

Design the technical architecture **for** TaskMaster Pro SaaS platform:

1. System architecture diagram
2. Technology stack recommendations
3. Database schema design
4. API structure and endpoints
5. Security considerations
6. Scalability planning

## Architecture Design

### System Architecture

```
graph TB
    subgraph "Frontend (React + TypeScript)"
        A[Dashboard] --> B[Task Management]
        A --> C[Project Management]
        A --> D[Team Management]
        A --> E[Analytics]
    end

    subgraph "Backend (Node.js + Express)"
        F[API Gateway] --> G[Auth Service]
        F --> H[Task Service]
        F --> I[Project Service]
        F --> J[User Service]
        F --> K[Billing Service]
    end

    subgraph "Database Layer"
        L[(PostgreSQL)]
        M[(Redis Cache)]
    end

    subgraph "External Services"
        N[Stripe API]
        O[SendGrid Email]
        P[WebSocket Server]
    end

    A --> F
    G --> L
    H --> L
    I --> L
    J --> L
    K --> N
    F --> M
    F --> P
    J --> O
```

### Technology Stack

#### Frontend:

- React 18 with TypeScript
- Vite for build tooling
- TailwindCSS for styling

- React Query for state management
- React Hook Form for forms
- React Router for navigation

**Backend:**

- Node.js 18+ with Express
- TypeScript for type safety
- PostgreSQL for primary database
- Redis for caching and sessions
- Socket.io for real-time features
- JWT for authentication

**DevOps & Deployment:**

- GitHub Actions for CI/CD
- Netlify for frontend hosting
- Railway for backend hosting
- Stripe for payment processing
- SendGrid for email services

## Database Schema

```
-- Users table
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  first_name VARCHAR(100) NOT NULL,
  last_name VARCHAR(100) NOT NULL,
  avatar_url VARCHAR(500),
  subscription_status VARCHAR(20) DEFAULT 'trial',
  subscription_id VARCHAR(100),
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Projects table
CREATE TABLE projects (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name VARCHAR(200) NOT NULL,
  description TEXT,
  owner_id UUID REFERENCES users(id) ON DELETE CASCADE,
  status VARCHAR(20) DEFAULT 'active',
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Project members table
CREATE TABLE project_members (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  project_id UUID REFERENCES projects(id) ON DELETE CASCADE,
  user_id UUID REFERENCES users(id) ON DELETE CASCADE,
  role VARCHAR(20) DEFAULT 'member',
  joined_at TIMESTAMP DEFAULT NOW(),
  UNIQUE(project_id, user_id)
);

-- Tasks table
CREATE TABLE tasks (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  title VARCHAR(200) NOT NULL,
  description TEXT,
  project_id UUID REFERENCES projects(id) ON DELETE CASCADE,
  assignee_id UUID REFERENCES users(id) ON DELETE SET NULL,
  creator_id UUID REFERENCES users(id) ON DELETE CASCADE,
  status VARCHAR(20) DEFAULT 'todo',
  priority VARCHAR(10) DEFAULT 'medium',
  due_date TIMESTAMP,
  completed_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT NOW(),
  updated_at TIMESTAMP DEFAULT NOW()
);

-- Create indexes for performance
CREATE INDEX idx_tasks_project_id ON tasks(project_id);
CREATE INDEX idx_tasks_assignee_id ON tasks(assignee_id);
CREATE INDEX idx_tasks_status ON tasks(status);
CREATE INDEX idx_project_members_user_id ON project_members(user_id);
```

# Development Setup

## Step 1: Initialize Project Structure

```
# Create project directory
mkdir taskmaster-pro
cd taskmaster-pro

# Initialize S.P.A. project
~/SPA_System/scripts/create-project.sh "TaskMaster Pro" saas-mvp

# Create frontend and backend directories
mkdir frontend backend
cd frontend

# Initialize React app with Vite
npm create vite@latest . -- --template react-ts
npm install

# Install additional dependencies
npm install @tanstack/react-query react-router-dom react-hook-form
npm install @headlessui/react @heroicons/react
npm install tailwindcss @tailwindcss/forms @tailwindcss/typography
npm install axios date-fns clsx

# Setup backend
cd ../backend
npm init -y
npm install express cors helmet morgan compression
npm install jsonwebtoken bcryptjs joi
npm install pg redis socket.io stripe nodemailer
npm install @types/node @types/express @types/jsonwebtoken
npm install @types/bcryptjs @types/cors typescript ts-node nodemon
```

## Step 2: Configure Development Environment

Frontend Configuration (frontend/vite.config.ts):

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import path from 'path'

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
    },
  },
  server: {
    port: 3000,
    proxy: {
      '/api': {
        target: 'http://localhost:5000',
        changeOrigin: true,
      },
    },
  },
})
```

#### Backend Configuration (backend/src/config/database.ts):

```
import { Pool } from 'pg';

const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl: process.env.NODE_ENV === 'production' ? { rejectUnauthorized: false } : false,
});

export default pool;
```

#### Environment Setup (.env files):

```
# backend/.env
NODE_ENV=development
PORT=5000
DATABASE_URL=postgresql://username:password@localhost:5432/taskmaster_dev
REDIS_URL=redis://localhost:6379
JWT_SECRET=your-super-secret-jwt-key
STRIPE_SECRET_KEY=sk_test_your_stripe_key
SENDGRID_API_KEY=your-sendgrid-key

# frontend/.env
VITE_API_URL=http://localhost:5000/api
VITE_STRIPE_PUBLISHABLE_KEY=pk_test_your_stripe_key
```



## Step 3: Setup Database

```
# Install PostgreSQL (Ubuntu/Debian)
sudo apt update
sudo apt install postgresql postgresql-contrib

# Create database and user
sudo -u postgres psql
CREATE DATABASE taskmaster_dev;
CREATE USER taskmaster_user WITH PASSWORD 'your_password';
GRANT ALL PRIVILEGES ON DATABASE taskmaster_dev TO taskmaster_user;
\q

# Run database migrations
cd backend
npm run migrate
```

## Core Feature Implementation

---

### Step 1: Authentication System

Backend Auth Service (backend/src/services/auth.service.ts):

```

import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import pool from '../config/database';
import { User, CreateUserData, LoginData } from '../types/user';

export class AuthService {
  private readonly JWT_SECRET = process.env.JWT_SECRET!;
  private readonly SALT_ROUNDS = 12;

  async register(userData: CreateUserData): Promise<{ user: User; token: string }> {
    const { email, password, firstName, lastName } = userData;

    // Check if user exists
    const existingUser = await pool.query(
      'SELECT id FROM users WHERE email = $1',
      [email]
    );

    if (existingUser.rows.length > 0) {
      throw new Error('User already exists');
    }

    // Hash password
    const passwordHash = await bcrypt.hash(password, this.SALT_ROUNDS);

    // Create user
    const result = await pool.query(
      `INSERT INTO users (email, password_hash, first_name, last_name)
      VALUES ($1, $2, $3, $4)
      RETURNING id, email, first_name, last_name, created_at`,
      [email, passwordHash, firstName, lastName]
    );

    const user = result.rows[0];
    const token = this.generateToken(user.id);

    return { user, token };
  }

  async login(loginData: LoginData): Promise<{ user: User; token: string }> {
    const { email, password } = loginData;

    // Find user
    const result = await pool.query(
      'SELECT * FROM users WHERE email = $1',
      [email]
    );

    if (result.rows.length === 0) {
      throw new Error('Invalid credentials');
    }

    const user = result.rows[0];

    // Verify password
    const isValidPassword = await bcrypt.compare(password, user.password_hash);
    if (!isValidPassword) {
      throw new Error('Invalid credentials');
    }
  }
}

```

```
    }

    const token = this.generateToken(user.id);

    // Remove password from response
    delete user.password_hash;

    return { user, token };
  }

  private generateToken(userId: string): string {
    return jwt.sign({ userId }, this.JWT_SECRET, { expiresIn: '7d' });
  }

  verifyToken(token: string): { userId: string } {
    return jwt.verify(token, this.JWT_SECRET) as { userId: string };
  }
}
```

Frontend Auth Hook (frontend/src/hooks/useAuth.ts):

```

import { createContext, useContext, useState, useEffect, ReactNode } from 'react';
import { User } from '../types/user';
import { authApi } from '../services/api';

interface AuthContextType {
  user: User | null;
  login: (email: string, password: string) => Promise<void>;
  register: (userData: RegisterData) => Promise<void>;
  logout: () => void;
  isLoading: boolean;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export function AuthProvider({ children }: { children: ReactNode }) {
  const [user, setUser] = useState<User | null>(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      authApi.verifyToken(token)
        .then(setUser)
        .catch(() => localStorage.removeItem('token'))
        .finally(() => setIsLoading(false));
    } else {
      setIsLoading(false);
    }
  }, []);

  const login = async (email: string, password: string) => {
    const { user, token } = await authApi.login({ email, password });
    localStorage.setItem('token', token);
    setUser(user);
  };

  const register = async (userData: RegisterData) => {
    const { user, token } = await authApi.register(userData);
    localStorage.setItem('token', token);
    setUser(user);
  };

  const logout = () => {
    localStorage.removeItem('token');
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, register, logout, isLoading }}>
      {children}
    </AuthContext.Provider>
  );
}

export function useAuth() {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
}

```

```
}  
  return context;  
}
```

## Step 2: Task Management System

Task Service (`backend/src/services/task.service.ts`):

```

import pool from '../config/database';
import { Task, CreateTaskData, UpdateTaskData } from '../types/task';

export class TaskService {
  async createTask(taskData: CreateTaskData, creatorId: string): Promise<Task> {
    const { title, description, projectId, assigneeId, priority, dueDate } = taskData;

    const result = await pool.query(
      `INSERT INTO tasks (title, description, project_id, assignee_id, creator_id,
priority, due_date)
      VALUES ($1, $2, $3, $4, $5, $6, $7)
      RETURNING *`,
      [title, description, projectId, assigneeId, creatorId, priority, dueDate]
    );

    return result.rows[0];
  }

  async getProjectTasks(projectId: string): Promise<Task[]> {
    const result = await pool.query(
      `SELECT t.*,
      u1.first_name as assignee_first_name, u1.last_name as assignee_last_name,
      u2.first_name as creator_first_name, u2.last_name as creator_last_name
      FROM tasks t
      LEFT JOIN users u1 ON t.assignee_id = u1.id
      LEFT JOIN users u2 ON t.creator_id = u2.id
      WHERE t.project_id = $1
      ORDER BY t.created_at DESC`,
      [projectId]
    );

    return result.rows;
  }

  async updateTask(taskId: string, updates: UpdateTaskData): Promise<Task> {
    const fields = Object.keys(updates);
    const values = Object.values(updates);

    const setClause = fields.map((field, index) =>
      `${field} = ${values[index + 1]}`
    ).join(', ');

    const result = await pool.query(
      `UPDATE tasks SET ${setClause}, updated_at = NOW()
      WHERE id = ${values[fields.length + 1]}
      RETURNING *`,
      [...values, taskId]
    );

    return result.rows[0];
  }

  async deleteTask(taskId: string): Promise<void> {
    await pool.query('DELETE FROM tasks WHERE id = $1', [taskId]);
  }
}

```

Frontend Task Components (frontend/src/components/TaskBoard.tsx):

```

import React, { useState } from 'react';
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query';
import { DragDropContext, Droppable, Draggable } from 'react-beautiful-dnd';
import { Task } from '../types/task';
import { taskApi } from '../services/api';
import TaskCard from '../TaskCard';
import CreateTaskModal from '../CreateTaskModal';

interface TaskBoardProps {
  projectId: string;
}

const COLUMNS = [
  { id: 'todo', title: 'To Do', color: 'bg-gray-100' },
  { id: 'in_progress', title: 'In Progress', color: 'bg-blue-100' },
  { id: 'review', title: 'Review', color: 'bg-yellow-100' },
  { id: 'done', title: 'Done', color: 'bg-green-100' },
];

export default function TaskBoard({ projectId }: TaskBoardProps) {
  const [isCreateModalOpen, setIsCreateModalOpen] = useState(false);
  const queryClient = useQueryClient();

  const { data: tasks = [], isLoading } = useQuery({
    queryKey: ['tasks', projectId],
    queryFn: () => taskApi.getProjectTasks(projectId),
  });

  const updateTaskMutation = useMutation({
    mutationFn: ({ taskId, updates }: { taskId: string; updates: any }) =>
      taskApi.updateTask(taskId, updates),
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ['tasks', projectId] });
    },
  });

  const handleDragEnd = (result: any) => {
    if (!result.destination) return;

    const { draggableId, destination } = result;
    const newStatus = destination.droppableId;

    updateTaskMutation.mutate({
      taskId: draggableId,
      updates: { status: newStatus },
    });
  };

  const getTasksByStatus = (status: string) =>
    tasks.filter((task: Task) => task.status === status);

  if (isLoading) {
    return <div className="flex justify-center p-8">Loading tasks...</div>;
  }

  return (
    <div className="h-full">
      <div className="flex justify-between items-center mb-6">

```

```

<h2 className="text-2xl font-bold text-gray-900">Task Board</h2>
<button
  onClick={() => setIsCreateModalOpen(true)}
  className="bg-blue-600 text-white px-4 py-2 rounded-lg hover:bg-blue-700"
>
  Add Task
</button>
</div>

<DragDropContext onDragEnd={handleDragEnd}>
  <div className="grid grid-cols-4 gap-6 h-full">
    {COLUMNS.map((column) => (
      <div key={column.id} className="flex flex-col">
        <div className={` ${column.color} p-3 rounded-t-lg`} >
          <h3 className="font-semibold text-gray-800">{column.title}</h3>
          <span className="text-sm text-gray-600">
            {getTasksByStatus(column.id).length} tasks
          </span>
        </div>

        <Draggable draggableId={column.id}>
          {(provided, snapshot) => (
            <div
              ref={provided.innerRef}
              {...provided.draggableProps}
              className={`flex-1 p-2 bg-gray-50 rounded-b-lg min-h-96 ${
                snapshot.isDraggingOver ? 'bg-gray-100' : ''
              }`}
            >
              {getTasksByStatus(column.id).map((task: Task, index: number) => (
                <Draggable key={task.id} draggableId={task.id} index={index}>
                  {(provided, snapshot) => (
                    <div
                      ref={provided.innerRef}
                      {...provided.draggableProps}
                      {...provided.dragHandleProps}
                      className={`mb-3 ${
                        snapshot.isDragging ? 'rotate-3 shadow-lg' : ''
                      }`}
                    >
                      <TaskCard task={task} />
                    </div>
                  </Draggable>
                </div>
              </div>
            </div>
          </Draggable>
        </div>
      </div>
    </div>
  </div>
</DragDropContext>

<CreateTaskModal
  isOpen={isCreateModalOpen}
  onClose={() => setIsCreateModalOpen(false)}
  projectId={projectId}
/>

```



```
    </div>  
  );  
}
```

### Step 3: Real-time Notifications

WebSocket Server (backend/src/services/socket.service.ts):

```

import { Server } from 'socket.io';
import { Server as HttpServer } from 'http';
import jwt from 'jsonwebtoken';

export class SocketService {
  private io: Server;

  constructor(server: HttpServer) {
    this.io = new Server(server, {
      cors: {
        origin: process.env.FRONTEND_URL,
        methods: ['GET', 'POST'],
      },
    });

    this.setupMiddleware();
    this.setupEventHandlers();
  }

  private setupMiddleware() {
    this.io.use((socket, next) => {
      const token = socket.handshake.auth.token;

      try {
        const decoded = jwt.verify(token, process.env.JWT_SECRET!) as any;
        socket.userId = decoded.userId;
        next();
      } catch (err) {
        next(new Error('Authentication error'));
      }
    });
  }

  private setupEventHandlers() {
    this.io.on('connection', (socket) => {
      console.log(`User ${socket.userId} connected`);

      socket.on('join-project', (projectId: string) => {
        socket.join(`project:${projectId}`);
      });

      socket.on('leave-project', (projectId: string) => {
        socket.leave(`project:${projectId}`);
      });

      socket.on('disconnect', () => {
        console.log(`User ${socket.userId} disconnected`);
      });
    });
  }

  notifyTaskUpdate(projectId: string, task: any) {
    this.io.to(`project:${projectId}`).emit('task-updated', task);
  }

  notifyTaskCreated(projectId: string, task: any) {
    this.io.to(`project:${projectId}`).emit('task-created', task);
  }
}

```

```
}  
}
```

## Testing and Quality Assurance

---

### Step 1: Unit Testing Setup

**Jest Configuration (backend/jest.config.js):**

```
module.exports = {  
  preset: 'ts-jest',  
  testEnvironment: 'node',  
  roots: ['<rootDir>/src'],  
  testMatch: ['**/__tests__/**/*.test.ts'],  
  collectCoverageFrom: [  
    'src/**/*.ts',  
    '!src/**/*.d.ts',  
    '!src/**/*.test.ts',  
  ],  
  coverageThreshold: {  
    global: {  
      branches: 80,  
      functions: 80,  
      lines: 80,  
      statements: 80,  
    },  
  },  
};
```

**Example Unit Tests (backend/src/services/tests/auth.service.test.ts):**

```

import { AuthService } from '../auth.service';
import pool from '../../config/database';
import bcrypt from 'bcryptjs';

jest.mock('../../config/database');
jest.mock('bcryptjs');

describe('AuthService', () => {
  let authService: AuthService;
  const mockPool = pool as jest.Mocked<typeof pool>;

  beforeEach(() => {
    authService = new AuthService();
    jest.clearAllMocks();
  });

  describe('register', () => {
    it('should create new user successfully', async () => {
      const userData = {
        email: 'test@example.com',
        password: 'password123',
        firstName: 'John',
        lastName: 'Doe',
      };

      mockPool.query
        .mockResolvedValueOnce({ rows: [] }) // Check existing user
        .mockResolvedValueOnce({ // Create user
          rows: [{
            id: '123',
            email: userData.email,
            first_name: userData.firstName,
            last_name: userData.lastName,
            created_at: new Date(),
          }],
        });

      (bcrypt.hash as jest.Mock).mockResolvedValue('hashedPassword');

      const result = await authService.register(userData);

      expect(result.user.email).toBe(userData.email);
      expect(result.token).toBeDefined();
      expect(bcrypt.hash).toHaveBeenCalledWith(userData.password, 12);
    });

    it('should throw error if user already exists', async () => {
      const userData = {
        email: 'existing@example.com',
        password: 'password123',
        firstName: 'John',
        lastName: 'Doe',
      };

      mockPool.query.mockResolvedValueOnce({ rows: [{ id: '123' }] });

      await expect(authService.register(userData)).rejects.toThrow('User already exists');
    });
  });
});

```

```
});
});
});
```

## Step 2: Integration Testing

API Integration Tests (backend/src/tests/auth.integration.test.ts):

```
import request from 'supertest';
import { app } from '../app';
import pool from '../config/database';

describe('Auth API', () => {
  beforeAll(async () => {
    // Setup test database
    await pool.query('BEGIN');
  });

  afterAll(async () => {
    // Cleanup test database
    await pool.query('ROLLBACK');
    await pool.end();
  });

  describe('POST /api/auth/register', () => {
    it('should register new user', async () => {
      const userData = {
        email: 'test@example.com',
        password: 'password123',
        firstName: 'John',
        lastName: 'Doe',
      };

      const response = await request(app)
        .post('/api/auth/register')
        .send(userData)
        .expect(201);

      expect(response.body.user.email).toBe(userData.email);
      expect(response.body.token).toBeDefined();
      expect(response.body.user.password).toBeUndefined();
    });

    it('should validate required fields', async () => {
      const response = await request(app)
        .post('/api/auth/register')
        .send({ email: 'invalid' })
        .expect(400);

      expect(response.body.errors).toBeDefined();
    });
  });
});
```

## Step 3: End-to-End Testing

Playwright E2E Tests (e2e/auth.spec.ts):

```
import { test, expect } from '@playwright/test';

test.describe('Authentication Flow', () => {
  test('should complete registration and login flow', async ({ page }) => {
    // Registration
    await page.goto('/register');

    await page.fill('[data-testid=first-name]', 'John');
    await page.fill('[data-testid=last-name]', 'Doe');
    await page.fill('[data-testid=email]', 'john@example.com');
    await page.fill('[data-testid=password]', 'password123');
    await page.fill('[data-testid=confirm-password]', 'password123');

    await page.click('[data-testid=register-button]');

    // Should redirect to dashboard
    await expect(page).toHaveURL('/dashboard');
    await expect(page.locator('[data-testid=welcome-message]'))
      .toContainText('Welcome, John');

    // Logout
    await page.click('[data-testid=user-menu]');
    await page.click('[data-testid=logout-button]');

    // Should redirect to login
    await expect(page).toHaveURL('/login');

    // Login
    await page.fill('[data-testid=email]', 'john@example.com');
    await page.fill('[data-testid=password]', 'password123');
    await page.click('[data-testid=login-button]');

    // Should be back in dashboard
    await expect(page).toHaveURL('/dashboard');
  });
});
```

## Deployment and Launch

### Step 1: CI/CD Pipeline Setup

GitHub Actions Workflow (.github/workflows/deploy.yml):

```

name: Deploy TaskMaster Pro

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

env:
  NODE_VERSION: '18'

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:14
        env:
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: taskmaster_test
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

    steps:
      - uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${ env.NODE_VERSION }
          cache: 'npm'
          cache-dependency-path: |
            frontend/package-lock.json
            backend/package-lock.json

      - name: Install dependencies
        run: |
          cd frontend && npm ci
          cd ../backend && npm ci

      - name: Run backend tests
        run: |
          cd backend
          npm run test
        env:
          DATABASE_URL: postgresql://postgres:postgres@localhost:5432/taskmaster_test

      - name: Run frontend tests
        run: |
          cd frontend
          npm run test

      - name: Build frontend
        run: |

```

```

    cd frontend
    npm run build

deploy-frontend:
  needs: test
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/main'

  steps:
    - uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${ env.NODE_VERSION }
        cache: 'npm'
        cache-dependency-path: frontend/package-lock.json

    - name: Install and build
      run: |
        cd frontend
        npm ci
        npm run build

    - name: Deploy to Netlify
      uses: netlify/actions/cli@master
      with:
        args: deploy --prod --dir=frontend/dist
      env:
        NETLIFY_SITE_ID: ${ secrets.NETLIFY_SITE_ID }
        NETLIFY_AUTH_TOKEN: ${ secrets.NETLIFY_AUTH_TOKEN }

deploy-backend:
  needs: test
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/main'

  steps:
    - uses: actions/checkout@v4

    - name: Deploy to Railway
      uses: railway-app/railway-action@v1
      with:
        api_token: ${ secrets.RAILWAY_TOKEN }
        project_id: ${ secrets.RAILWAY_PROJECT_ID }
        service: backend

```

## Step 2: Environment Configuration

Production Environment Variables:



```
# Backend (Railway)
NODE_ENV=production
DATABASE_URL=postgresql://user:pass@host:port/db
REDIS_URL=redis://user:pass@host:port
JWT_SECRET=production-jwt-secret
STRIPE_SECRET_KEY=sk_live_your_stripe_key
SENDGRID_API_KEY=your-sendgrid-key
FRONTEND_URL=https://taskmaster-pro.netlify.app

# Frontend (Netlify)
VITE_API_URL=https://taskmaster-api.railway.app/api
VITE_STRIPE_PUBLISHABLE_KEY=pk_live_your_stripe_key
```

### Step 3: Database Migration

Production Migration Script (backend/scripts/migrate-production.js):

```
const { Pool } = require('pg');

const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl: { rejectUnauthorized: false }
});

async function runMigrations() {
  try {
    console.log('Running production migrations...');

    // Create tables
    await pool.query(`
      CREATE TABLE IF NOT EXISTS users (
        id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
        email VARCHAR(255) UNIQUE NOT NULL,
        password_hash VARCHAR(255) NOT NULL,
        first_name VARCHAR(100) NOT NULL,
        last_name VARCHAR(100) NOT NULL,
        avatar_url VARCHAR(500),
        subscription_status VARCHAR(20) DEFAULT 'trial',
        subscription_id VARCHAR(100),
        created_at TIMESTAMP DEFAULT NOW(),
        updated_at TIMESTAMP DEFAULT NOW()
      );
    `);

    // Add indexes
    await pool.query('CREATE INDEX IF NOT EXISTS idx_users_email ON users(email);');

    console.log('Migrations completed successfully');
  } catch (error) {
    console.error('Migration failed:', error);
    process.exit(1);
  } finally {
    await pool.end();
  }
}

runMigrations();
```

## Post-Launch Optimization

### Step 1: Performance Monitoring

Analytics Integration (frontend/src/utls/analytics.ts):

```
import { Analytics } from '@segment/analytics-node';

class AnalyticsService {
  private analytics: Analytics;

  constructor() {
    this.analytics = new Analytics({
      writeKey: process.env.VITE_SEGMENT_WRITE_KEY!,
    });
  }

  track(event: string, properties?: Record<string, any>) {
    this.analytics.track({
      userId: this.getCurrentUserId(),
      event,
      properties,
    });
  }

  identify(userId: string, traits?: Record<string, any>) {
    this.analytics.identify({
      userId,
      traits,
    });
  }

  private getCurrentUserId(): string {
    // Get from auth context
    return 'user-id';
  }
}

export const analytics = new AnalyticsService();
```

### Step 2: Performance Optimization

Database Query Optimization:

```

-- Add composite indexes for common queries
CREATE INDEX idx_tasks_project_status ON tasks(project_id, status);
CREATE INDEX idx_tasks_assignee_status ON tasks(assignee_id, status);

-- Optimize task queries with materialized views
CREATE MATERIALIZED VIEW project_task_summary AS
SELECT
  p.id as project_id,
  p.name as project_name,
  COUNT(t.id) as total_tasks,
  COUNT(CASE WHEN t.status = 'done' THEN 1 END) as completed_tasks,
  COUNT(CASE WHEN t.due_date < NOW() AND t.status != 'done' THEN 1 END) as over-
due_tasks
FROM projects p
LEFT JOIN tasks t ON p.id = t.project_id
GROUP BY p.id, p.name;

-- Refresh materialized view periodically
CREATE OR REPLACE FUNCTION refresh_project_summary()
RETURNS void AS $$
BEGIN
  REFRESH MATERIALIZED VIEW project_task_summary;
END;
$$ LANGUAGE plpgsql;

```

Frontend Performance Optimization:

```

// Implement virtual scrolling for large task lists
import { FixedSizeList as List } from 'react-window';

function VirtualTaskList({ tasks }: { tasks: Task[] }) {
  const Row = ({ index, style }: { index: number; style: any }) => (
    <div style={style}>
      <TaskCard task={tasks[index]} />
    </div>
  );

  return (
    <List
      height={600}
      itemCount={tasks.length}
      itemSize={120}
      width="100%"
    >
      {Row}
    </List>
  );
}

// Implement optimistic updates
const updateTaskMutation = useMutation({
  mutationFn: taskApi.updateTask,
  onMutate: async (variables) => {
    await queryClient.cancelQueries({ queryKey: ['tasks', projectId] });

    const previousTasks = queryClient.getQueryData(['tasks', projectId]);

    queryClient.setQueryData(['tasks', projectId], (old: Task[]) =>
      old.map(task =>
        task.id === variables.taskId
          ? { ...task, ...variables.updates }
          : task
      )
    );

    return { previousTasks };
  },
  onError: (err, variables, context) => {
    queryClient.setQueryData(['tasks', projectId], context?.previousTasks);
  },
});

```

### Step 3: Scaling Considerations

#### Horizontal Scaling Setup:

```
// Load balancer configuration for multiple backend instances
// nginx.conf
upstream backend {
    server backend-1:5000;
    server backend-2:5000;
    server backend-3:5000;
}

server {
    listen 80;

    location /api {
        proxy_pass http://backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}

// Database connection pooling
import { Pool } from 'pg';

const pool = new Pool({
    connectionString: process.env.DATABASE_URL,
    max: 20, // Maximum number of connections
    idleTimeoutMillis: 30000,
    connectionTimeoutMillis: 2000,
});
```

### Caching Strategy:

```
// Redis caching for frequently accessed data
import Redis from 'ioredis';

class CacheService {
    private redis = new Redis(process.env.REDIS_URL);

    async getProjectTasks(projectId: string): Promise<Task[] | null> {
        const cached = await this.redis.get(`project:${projectId}:tasks`);
        return cached ? JSON.parse(cached) : null;
    }

    async setProjectTasks(projectId: string, tasks: Task[]): Promise<void> {
        await this.redis.setex(
            `project:${projectId}:tasks`,
            300, // 5 minutes TTL
            JSON.stringify(tasks)
        );
    }

    async invalidateProjectTasks(projectId: string): Promise<void> {
        await this.redis.del(`project:${projectId}:tasks`);
    }
}
```

Congratulations! You've successfully built and deployed TaskMaster Pro using the S.P.A. ecosystem. This tutorial demonstrates the power of combining AI-assisted development with modern best practices to create production-ready SaaS applications rapidly and efficiently.