ProTakeoff.ai - Technical Documentation (Phase 1)

1. Overview / Introduction

Project Name

ProTakeoff.ai - Digital Marketplace for Landscaping & Irrigation Takeoffs

Purpose

ProTakeoff.ai is a digital marketplace platform that enables small to midsize landscaping and irrigation companies to instantly purchase pre-completed project takeoffs. The platform eliminates the need for manual blueprint analysis, allowing contractors to bid faster and more efficiently on construction projects.

Target Users:

- Small to midsize landscaping companies
- Irrigation contractors
- Construction estimators
- General contractors seeking subcontractor bids

Scope

Phase 1 Features (Current Implementation):

- User registration and authentication system
- ZIP code and project size-based search functionality
- Takeoff catalog browsing and filtering
- Secure payment processing via Stripe
- Instant digital delivery of Excel takeoff files
- User dashboard for purchase history
- Admin panel for takeoff management
- Email notification system

Limitations:

- Manual takeoff creation (no AI automation yet)
- Limited to Texas market initially
- Fixed pricing tiers ($50-$150)

- Excel-only format delivery
- No real-time collaboration features

Technology Stack

Frontend:

- React 18 + TypeScript
- React Router DOM 6+ for routing
- Tailwind CSS for styling
- Axios for HTTP requests
- React Hook Form for form validation
- React Query for state management and caching

Backend:

- Node.js + Express
- TypeScript for type safety
- JSON Web Tokens (JWT) for authentication
- Bcrypt for password hashing
- Multer for file uploads
- Nodemailer for email services
- Express-rate-limit for API rate limiting

Database & Auth:

- MongoDB
- Mongoose ODM for object modeling

Payment Processing:

- Stripe API for payment processing
- Stripe Webhooks for payment confirmation

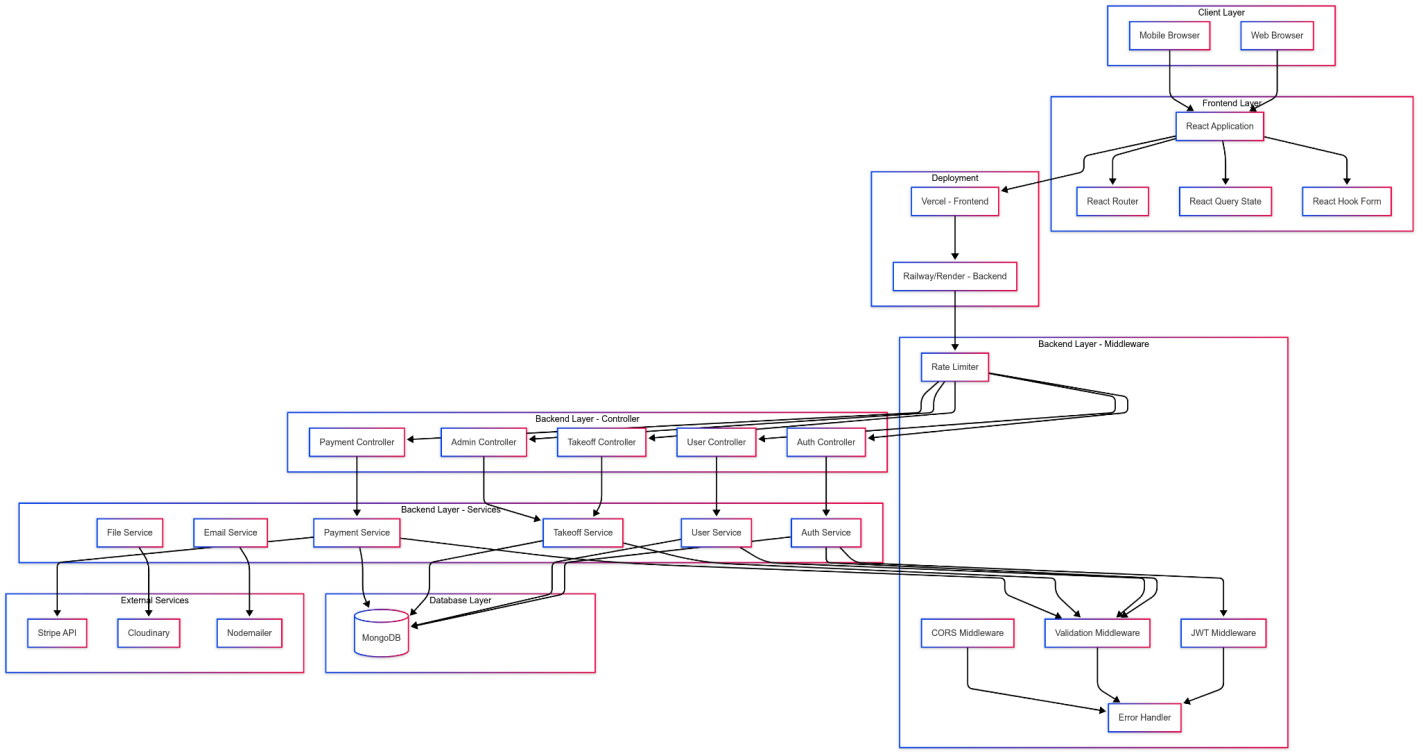
Cloud Storage:

- Cloudinary for file storage and media management
- Multer for handling file uploads

Deployment:

- Frontend: Vercel
- Backend: Railway or Render

Architecture Diagram



2. System Architecture

Frontend Architecture

Component Hierarchy:

```
src/
├── components/
│   ├── common/
│   │   ├── Header.tsx
│   │   ├── Footer.tsx
│   │   ├── Loading.tsx
│   │   ├── ErrorBoundary.tsx
│   │   └── Modal.tsx
│   ├── auth/
│   │   ├── LoginForm.tsx
│   │   ├── RegisterForm.tsx
│   │   └── ProtectedRoute.tsx
│   ├── takeoffs/
│   │   ├── TakeoffCard.tsx
```

```
│   │   ├── TakeoffSearch.tsx
│   │   ├── TakeoffFilters.tsx
│   │   └── TakeoffDetails.tsx
│   ├── payment/
│   │   ├── PaymentForm.tsx
│   │   ├── CheckoutSummary.tsx
│   │   └── PaymentSuccess.tsx
│   └── dashboard/
│       ├── UserDashboard.tsx
│       ├── PurchaseHistory.tsx
│       └── AdminPanel.tsx
├── pages/
│   ├── Home.tsx
│   ├── Browse.tsx
│   ├── Checkout.tsx
│   ├── Dashboard.tsx
│   └── Admin.tsx
├── hooks/
│   ├── useAuth.ts
│   ├── useTakeoffs.ts
│   └── usePayment.ts
├── services/
│   ├── api.ts
│   ├── auth.ts
│   └── payment.ts
├── types/
│   ├── user.ts
│   ├── takeoff.ts
│   └── payment.ts
└── utils/
    ├── constants.ts
    ├── helpers.ts
    └── validation.ts
```

State Management Strategy:

- **React Query** for server state management and caching
- **Context API** for global app state (user authentication)
- **Component state** for local UI state
- **Form state** managed by React Hook Form

Backend Architecture

MVC + Service Layer Pattern:

```
src/
├── controllers/
│   ├── authController.ts
│   ├── userController.ts
│   ├── takeoffController.ts
│   ├── paymentController.ts
│   └── adminController.ts
├── services/
│   ├── authService.ts
│   ├── userService.ts
│   ├── takeoffService.ts
│   ├── paymentService.ts
│   ├── emailService.ts
│   └── fileService.ts
├── models/
│   ├── User.ts
│   ├── Takeoff.ts
│   ├── Purchase.ts
│   └── Admin.ts
├── middleware/
│   ├── auth.ts
│   ├── validation.ts
│   ├── errorHandler.ts
│   └── rateLimit.ts
├── routes/
│   ├── auth.ts
│   ├── users.ts
│   ├── takeoffs.ts
│   ├── payments.ts
│   └── admin.ts
├── utils/
│   ├── database.ts
│   ├── jwt.ts
│   ├── email.ts
│   └── constants.ts
└── config/
    ├── database.ts
    ├── stripe.ts
    └── email.ts
```

RESTful API Structure:

- **Controllers:** Handle HTTP requests/responses
- **Services:** Business logic and data processing

- **Models:** Database schema definitions
- **Middleware:** Authentication, validation, error handling
- **Routes:** API endpoint definitions

Database Schema

MongoDB Collections:

Users Collection:

```
{
  _id: ObjectId,
  email: String (unique, required),
  password: String (hashed, required),
  firstName: String (required),
  lastName: String (required),
  company: String (required),
  phone: String,
  address: {
    street: String,
    city: String,
    state: String,
    zipCode: String (required)
  },
  role: String (enum: ['user', 'admin'], default: 'user'),
  isVerified: Boolean (default: false),
  verificationToken: String,
  resetPasswordToken: String,
  resetPasswordExpires: Date,
  createdAt: Date (default: Date.now),
  updatedAt: Date (default: Date.now),
  lastLogin: Date
}
```

Takeoffs Collection:

```
{
  _id: ObjectId,
  title: String (required),
  description: String (required),
  projectType: String (enum: ['landscaping', 'irrigation', 'both']),
  projectSize: String (enum: ['small', 'medium', 'large']),
```

```
  zipCode: String (required),
  price: Number (required),
  features: [String],
  specifications: {
    area: Number, // square feet
    complexity: String (enum: ['basic', 'intermediate', 'advanced']),
    materials: [String],
    estimatedHours: Number
  },
  files: [{
    filename: String,
    originalName: String,
    size: Number,
    cloudinaryPublicId: String, // Cloudinary public ID
    cloudinaryUrl: String, // Cloudinary secure URL
    resourceType: String, // 'raw' for files, 'image' for images
    uploadDate: Date
  }],
  images: [{
    cloudinaryPublicId: String, // Cloudinary public ID for images
    cloudinaryUrl: String, // Cloudinary secure URL
    thumbnailUrl: String, // Cloudinary thumbnail URL
    width: Number,
    height: Number
  }],
  tags: [String],
  isActive: Boolean (default: true),
  downloadCount: Number (default: 0),
  createdBy: ObjectId (ref: 'User'),
  createdAt: Date (default: Date.now),
  updatedAt: Date (default: Date.now)
}
```

Purchases Collection:

```
{
  _id: ObjectId,
  userId: ObjectId (ref: 'User', required),
  takeoffId: ObjectId (ref: 'Takeoff', required),
  stripePaymentIntentId: String (required),
  amount: Number (required),
  currency: String (default: 'usd'),
  status: String (enum: ['pending', 'completed', 'failed', 'refunded']),
```

```
  paymentMethod: {
    type: String, // card type
    last4: String,
    brand: String
  },
  downloadUrl: String,
  downloadExpires: Date,
  downloadCount: Number (default: 0),
  maxDownloads: Number (default: 5),
  purchaseDate: Date (default: Date.now),
  refundDate: Date,
  refundReason: String
}
```

Key Relationships:

- Users ↔ Purchases (One-to-Many)
- Takeoffs ↔ Purchases (One-to-Many)
- Users ↔ Takeoffs (Many-to-Many through Purchases)

3. Setup and Installation

Prerequisites

System Requirements:

- Node.js 18.0 or higher
- MongoDB 6.0 or higher
- Git
- Docker (optional, for containerized deployment)

Accounts Required:

- Stripe account (for payment processing)
- Cloudinary account (alternative email service)

Local Setup Instructions

1. Clone Repository:

```
git clone https://github.com/your-org/protakeoff-ai.git
cd protakeoff-ai
```

2. Install Dependencies:

```
# Install backend dependencies
cd backend
npm install

# Install frontend dependencies
cd ../frontend
npm install
```

3. Database Setup:

```
# Start MongoDB (if running locally)
mongod --dbpath /path/to/your/db

# Or using Docker
docker run -d -p 27017:27017 --name mongodb mongo:6
```

4. Environment Configuration:

```
# Backend environment
cd backend
cp .env.example .env
# Edit .env with your configuration

# Frontend environment
cd ../frontend
cp .env.example .env
# Edit .env with your configuration
```

5. Run Development Servers:

```
# Terminal 1 - Backend
cd backend
npm run dev

# Terminal 2 - Frontend
cd frontend
npm run dev
```

6. Database Seeding (Optional):

cd backend
npm run seed

Environment Variables (.env)

Backend Environment Variables:

```
# Server Configuration
NODE_ENV=development
PORT=5000
API_URL=http://localhost:5000

# Database
MONGO_URI=mongodb://localhost:27017/protakeoff
MONGO_TEST_URI=mongodb://localhost:27017/protakeoff_test

# JWT Configuration
JWT_SECRET=your-super-secret-jwt-key-min-32-characters
JWT_EXPIRE=7d
JWT_REFRESH_SECRET=your-refresh-token-secret
JWT_REFRESH_EXPIRE=30d

# 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

# Email Configuration (Nodemailer)
EMAIL_SERVICE=gmail # or your preferred service
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_SECURE=false
EMAIL_USER=your-email@gmail.com
EMAIL_PASS=your-app-password
EMAIL_FROM=noreply@protakeoff.ai
EMAIL_FROM_NAME=ProTakeoff Team

# Cloudinary Configuration
CLOUDINARY_CLOUD_NAME=your-cloud-name
CLOUDINARY_API_KEY=your-api-key
CLOUDINARY_API_SECRET=your-api-secret
CLOUDINARY_FOLDER=protakeoff # Optional: folder name in Cloudinary
```

```
# File Upload Configuration
MAX_FILE_SIZE=10485760 # 10MB in bytes
ALLOWED_FILE_TYPES=.xlsx,.xls,.csv
ALLOWED_IMAGE_TYPES=.jpg,.jpeg,.png,.gif,.webp

# Rate Limiting
RATE_LIMIT_WINDOW_MS=900000 # 15 minutes
RATE_LIMIT_MAX_REQUESTS=100

# Frontend URL (for CORS and emails)
FRONTEND_URL=http://localhost:3000

# Deployment URLs (for production)
VERCEL_URL=https://your-app.vercel.app
RAILWAY_URL=https://your-backend.railway.app
```

Frontend Environment Variables:

```
# API Configuration
REACT_APP_API_URL=http://localhost:5000/api
REACT_APP_API_TIMEOUT=10000

# Stripe Configuration
REACT_APP_STRIPE_PUBLISHABLE_KEY=pk_test_your_stripe_publishable_key

# App Configuration
REACT_APP_NAME=ProTakeoff.ai
REACT_APP_VERSION=1.0.0
REACT_APP_ENVIRONMENT=development

# Feature Flags
REACT_APP_ENABLE_ANALYTICS=false
REACT_APP_ENABLE_CHAT_SUPPORT=false
```

4. API Documentation

Base URL

- **Development:** `http://localhost:5000/api`
- **Production:** `https://api.protakeoff.ai/api`

Authentication Endpoints

POST /api/auth/register

- **Purpose:** Register new user account
- **Body:**

```
{
  "email": "user@example.com",
  "password": "SecurePass123",
  "firstName": "John",
  "lastName": "Doe",
  "company": "ABC Landscaping",
  "phone": "555-123-4567",
  "address": {
    "street": "123 Main St",
    "city": "Austin",
    "state": "TX",
    "zipCode": "78701"
  }
}
```

- **Response (201):**

```
{
  "success": true,
  "message": "Registration successful. Please check your email for verification.",
  "data": {
    "user": {
      "id": "64a1b2c3d4e5f6789abcdef0",
      "email": "user@example.com",
      "firstName": "John",
      "lastName": "Doe",
      "isVerified": false
    }
  }
}
```

POST /api/auth/login

- **Purpose:** User login
- **Body:**

```
{
  "email": "user@example.com",
  "password": "SecurePass123"
}
```

- **Response (200):**

```
{
  "success": true,
  "message": "Login successful",
  "data": {
    "user": {
      "id": "64a1b2c3d4e5f6789abcdef0",
      "email": "user@example.com",
      "firstName": "John",
      "lastName": "Doe",
      "company": "ABC Landscaping"
    },
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
  }
}
```

Takeoff Endpoints

GET /api/takeoffs

- **Purpose:** Browse available takeoffs
- **Query Parameters:**
  - `zipCode` (optional): Filter by ZIP code
  - `projectType` (optional): 'landscaping', 'irrigation', 'both'
  - `projectSize` (optional): 'small', 'medium', 'large'
  - `minPrice` (optional): Minimum price filter
  - `maxPrice` (optional): Maximum price filter
  - `page` (optional): Page number (default: 1)
  - `limit` (optional): Items per page (default: 20)
- **Response (200):**

```
{
  "success": true,
  "data": {
    "takeoffs": [
```

```
    {
      "id": "64a1b2c3d4e5f6789abcdef1",
      "title": "Residential Landscaping - 2,500 sq ft",
      "description": "Complete landscaping takeoff for suburban home",
      "projectType": "landscaping",
      "projectSize": "medium",
      "zipCode": "78701",
      "price": 75,
      "images": ["https://s3.amazonaws.com/protakeoff/preview1.jpg"],
      "features": ["Tree installation", "Sod installation", "Mulch beds"],
      "downloadCount": 45
    }
   ],
   "pagination": {
    "currentPage": 1,
    "totalPages": 5,
    "totalItems": 92,
    "hasNext": true,
    "hasPrev": false
   }
  }
}
```


GET /api/takeoffs/:id

- **Purpose:** Get detailed takeoff information
- **Authentication:** Required
- **Response (200):**


```
{
  "success": true,
  "data": {
    "takeoff": {
      "id": "64a1b2c3d4e5f6789abcdef1",
      "title": "Residential Landscaping - 2,500 sq ft",
      "description": "Complete landscaping takeoff including material quantities, labor estimates,
and pricing guidelines.",
      "projectType": "landscaping",
      "projectSize": "medium",
      "zipCode": "78701",
      "price": 75,
      "specifications": {
        "area": 2500,
        "complexity": "intermediate",
```

```
      "materials": ["Sod", "Trees", "Shrubs", "Mulch", "Irrigation"],
      "estimatedHours": 32
    },
    "features": [
      "15 trees (various species)",
      "1,200 sq ft sod installation",
      "8 shrub groupings",
      "Mulch bed preparation"
    ],
    "images": [
      "https://res.cloudinary.com/da6bsligl/image/upload/protakeoff/preview1.jpg",
      "https://res.cloudinary.com/da6bsligl/image/upload/protakeoff/preview2.jpg"
    ],
    "tags": ["residential", "suburban", "medium-complexity"],
    "downloadCount": 45,
    "createdAt": "2024-01-15T10:30:00Z"
   }
  }
}
```


Payment Endpoints

POST /api/payments/create-intent

- **Purpose:** Create Stripe payment intent
- **Authentication:** Required
- **Body:**


```
{
  "takeoffId": "64a1b2c3d4e5f6789abcdef1"
}
```


- **Response (200):**


```
{
  "success": true,
  "data": {
    "clientSecret": "pi_1234567890_secret_abcdef",
    "amount": 7500,
    "currency": "usd",
    "takeoffId": "64a1b2c3d4e5f6789abcdef1"
  }
}
```

POST /api/payments/confirm

- **Purpose:** Confirm successful payment and create purchase record
- **Authentication:** Required
- **Body:**

```
{
  "paymentIntentId": "pi_1234567890",
  "takeoffId": "64a1b2c3d4e5f6789abcdef1"
}
```

- **Response (200):**

```
{
  "success": true,
  "message": "Payment confirmed successfully",
  "data": {
    "purchase": {
      "id": "64a1b2c3d4e5f6789abcdef2",
      "takeoffId": "64a1b2c3d4e5f6789abcdef1",
      "amount": 75,
      "status": "completed",
      "downloadUrl": "https://api.protakeoff.ai/api/downloads/secure-token-123",
      "downloadExpires": "2024-02-15T10:30:00Z",
      "purchaseDate": "2024-01-15T10:30:00Z"
    }
  }
}
```

User Dashboard Endpoints

GET /api/users/purchases

- **Purpose:** Get user's purchase history
- **Authentication:** Required
- **Query Parameters:**
  - `page` (optional): Page number
  - `limit` (optional): Items per page
- **Response (200):**

```
{
```

```
  "success": true,
  "data": {
    "purchases": [
      {
        "id": "64a1b2c3d4e5f6789abcdef2",
        "takeoff": {
          "id": "64a1b2c3d4e5f6789abcdef1",
          "title": "Residential Landscaping - 2,500 sq ft",
          "projectType": "landscaping"
        },
        "amount": 75,
        "status": "completed",
        "purchaseDate": "2024-01-15T10:30:00Z",
        "downloadCount": 3,
        "maxDownloads": 5,
        "downloadUrl": "https://api.protakeoff.ai/api/downloads/secure-token-123"
      }
    ],
    "pagination": {
      "currentPage": 1,
      "totalPages": 2,
      "totalItems": 12
    }
  }
}
```


Admin Endpoints

GET /api/admin/takeoffs

- **Purpose:** Get all takeoffs for admin management
- **Authentication:** Required (Admin role)
- **Response (200):**


```
{
  "success": true,
  "data": {
    "takeoffs": [
      {
        "id": "64a1b2c3d4e5f6789abcdef1",
        "title": "Residential Landscaping - 2,500 sq ft",
        "price": 75,
        "isActive": true,
        "downloadCount": 45,
```

```
      "createdAt": "2024-01-15T10:30:00Z"
    }
   ]
  }
}
```

POST /api/admin/takeoffs

- **Purpose:** Create new takeoff
- **Authentication:** Required (Admin role)
- **Content-Type:** multipart/form-data
- **Body:** FormData with takeoff details and files

Error Response Format

```
{
  "success": false,
  "error": {
    "message": "Error description",
    "code": "ERROR_CODE",
    "details": {}
  }
}
```

Common HTTP Status Codes:

- `200` - Success
- `201` - Created
- `400` - Bad Request
- `401` - Unauthorized
- `403` - Forbidden
- `404` - Not Found
- `422` - Validation Error
- `429` - Too Many Requests
- `500` - Internal Server Error

Authentication Strategy

JWT Token Implementation:

- **Access Token:** Short-lived (7 days), used for API requests
- **Refresh Token:** Long-lived (30 days), used to generate new access tokens

- **Token Storage:** HTTP-only cookies (production) or localStorage (development)

Headers:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

5. Frontend Documentation

Component Structure

Layout Components:

- `Header.tsx` - Navigation, user menu, cart icon
- `Footer.tsx` - Links, contact info, legal
- `Layout.tsx` - Common page wrapper
- `Sidebar.tsx` - Filters and navigation (for browse page)

Authentication Components:

- `LoginForm.tsx` - User login with validation
- `RegisterForm.tsx` - New user registration
- `ProtectedRoute.tsx` - Route protection wrapper
- `AuthProvider.tsx` - Authentication context provider

Takeoff Components:

- `TakeoffCard.tsx` - Takeoff preview card
- `TakeoffGrid.tsx` - Grid layout for takeoff cards
- `TakeoffDetails.tsx` - Detailed takeoff view
- `TakeoffSearch.tsx` - Search input and filters
- `TakeoffFilters.tsx` - Advanced filtering options

Routing (React Router)

Route Structure:

```
// App.tsx
<BrowserRouter>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/browse" element={<Browse />} />
    <Route path="/takeoff/:id" element={<TakeoffDetails />} />
```

```
    <Route path="/login" element={<Login />} />
    <Route path="/register" element={<Register />} />

    {/* Protected Routes */}
    <Route element={<ProtectedRoute />}>
      <Route path="/checkout/:id" element={<Checkout />} />
      <Route path="/dashboard" element={<Dashboard />} />
      <Route path="/purchases" element={<PurchaseHistory />} />
    </Route>

    {/* Admin Routes */}
    <Route element={<ProtectedRoute requiredRole="admin" />}>
      <Route path="/admin" element={<AdminDashboard />} />
      <Route path="/admin/takeoffs" element={<TakeoffManagement />} />
    </Route>

    <Route path="*" element={<NotFound />} />
  </Routes>
</BrowserRouter>
```

Route Purposes:

- `/` - Landing page with hero section and featured takeoffs
- `/browse` - Main catalog with search and filters
- `/takeoff/:id` - Detailed takeoff view and purchase button
- `/checkout/:id` - Payment form and order summary
- `/dashboard` - User dashboard with purchase history
- `/admin/*` - Admin panel for takeoff management

State Management

React Query Implementation:

```
// hooks/useTakeoffs.ts
export const useTakeoffs = (filters: TakeoffFilters) => {
  return useQuery({
    queryKey: ['takeoffs', filters],
    queryFn: () => api.getTakeoffs(filters),
    staleTime: 5 * 60 * 1000, // 5 minutes
    cacheTime: 10 * 60 * 1000, // 10 minutes
  });
};
```

```
export const useTakeoffDetails = (id: string) => {
  return useQuery({
    queryKey: ['takeoff', id],
    queryFn: () => api.getTakeoffById(id),
    enabled: !!id,
  });
};

export const usePurchaseMutation = () => {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: api.createPurchase,
    onSuccess: () => {
      queryClient.invalidateQueries(['purchases']);
    },
  });
};
```

Context API for Authentication:

```
// contexts/AuthContext.tsx
interface AuthContextType {
  user: User | null;
  login: (email: string, password: string) => Promise<void>;
  logout: () => void;
  isLoading: boolean;
  isAuthenticated: boolean;
}

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

  // Implementation details...
};
```

Form Handling & Validation

React Hook Form Integration:

```
// components/forms/LoginForm.tsx
interface LoginFormData {
```

```
  email: string;
  password: string;
}

const LoginForm: React.FC = () => {
  const {
    register,
    handleSubmit,
    formState: { errors, isSubmitting }
  } = useForm<LoginFormData>({
    resolver: yupResolver(loginSchema)
  });

  const onSubmit = async (data: LoginFormData) => {
    try {
      await login(data.email, data.password);
    } catch (error) {
      // Handle error
    }
  };

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input
        {...register('email')}
        type="email"
        placeholder="Email"
        className="form-input"
      />
      {errors.email && (
        <span className="error">{errors.email.message}</span>
      )}

      <input
        {...register('password')}
        type="password"
        placeholder="Password"
        className="form-input"
      />
      {errors.password && (
        <span className="error">{errors.password.message}</span>
      )}

      <button type="submit" disabled={isSubmitting}>
```

```
      {isSubmitting ? 'Signing In...' : 'Sign In'}
    </button>
  </form>
 );
};
```

Validation Schemas (Yup):

```
// utils/validation.ts
export const loginSchema = yup.object({
 email: yup
  .string()
  .email('Invalid email format')
  .required('Email is required'),
 password: yup
  .string()
  .min(8, 'Password must be at least 8 characters')
  .required('Password is required'),
});

export const registerSchema = yup.object({
 email: yup
  .string()
  .email('Invalid email format')
  .required('Email is required'),
 password: yup
  .string()
  .min(8, 'Password must be at least 8 characters')
  .matches(
   /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/,
   'Password must contain uppercase, lowercase, and number'
  )
  .required('Password is required'),
 firstName: yup
  .string()
  .min(2, 'First name must be at least 2 characters')
  .required('First name is required'),
 lastName: yup
  .string()
  .min(2, 'Last name must be at least 2 characters')
  .required('Last name is required'),
 company: yup
  .string()
  .min(2, 'Company name must be at least 2 characters')
```

```
    .required('Company name is required'),
  address: yup.object({
    zipCode: yup
      .string()
      .matches(/^\d{5}(-\d{4})?$/, 'Invalid ZIP code format')
      .required('ZIP code is required'),
  }),
});
```

6. Authentication & Authorization

Login/Signup Flow

Frontend Registration Process:

1. User fills registration form with validation
2. Form data sent to `/api/auth/register`
3. Server creates user account (password hashed with bcrypt)
4. Verification email sent to user
5. User clicks verification link
6. Account activated, user can login

Frontend Login Process:

1. User enters email/password
2. Credentials sent to `/api/auth/login`
3. Server validates credentials
4. JWT tokens generated and returned
5. Tokens stored in httpOnly cookies (production) or localStorage (development)
6. User redirected to dashboard or intended page

Backend Authentication Flow:

```
// services/authService.ts
export class AuthService {
  async register(userData: RegisterData): Promise<{ user: User }> {
    // Check if user already exists
    const existingUser = await User.findOne({ email: userData.email });
    if (existingUser) {
      throw new ApiError('User already exists', 400);
    }

    // Hash password
```

```
  const hashedPassword = await bcrypt.hash(userData.password, 12);

  // Create user
  const user = await User.create({
    ...userData,
    password: hashedPassword,
    verificationToken: crypto.randomBytes(32).toString('hex'),
  });

  // Send verification email
  await EmailService.sendVerificationEmail(user);

  return { user: user.toObject({ transform: removePassword }) };
}

async login(email: string, password: string): Promise<LoginResponse> {
  // Find user and include password for comparison
  const user = await User.findOne({ email }).select('+password');
  if (!user) {
    throw new ApiError('Invalid credentials', 401);
  }

  // Check if account is verified
  if (!user.isVerified) {
    throw new ApiError('Please verify your email before logging in', 401);
  }

  // Verify password
  const isPasswordValid = await bcrypt.compare(password, user.password);
  if (!isPasswordValid) {
    throw new ApiError('Invalid credentials', 401);
  }

  // Generate tokens
  const accessToken = JWTService.generateAccessToken(user._id);
  const refreshToken = JWTService.generateRefreshToken(user._id);

  // Update last login
  user.lastLogin = new Date();
  await user.save();

  return {
    user: user.toObject({ transform: removePassword }),
    accessToken,
```

```
      refreshToken,
    };
  }
}
```

Role-based Access Control

User Roles:

- `user` - Regular customers (can browse and purchase takeoffs)
- `admin` - System administrators (can manage takeoffs and users)

Permission Matrix:

```
// utils/permissions.ts
export const PERMISSIONS = {
  // User permissions
  BROWSE_TAKEOFFS: ['user', 'admin'],
  PURCHASE_TAKEOFFS: ['user', 'admin'],
  VIEW_PURCHASE_HISTORY: ['user', 'admin'],
  DOWNLOAD_PURCHASED_FILES: ['user', 'admin'],

  // Admin permissions
  MANAGE_TAKEOFFS: ['admin'],
  MANAGE_USERS: ['admin'],
  VIEW_ANALYTICS: ['admin'],
  PROCESS_REFUNDS: ['admin'],
} as const;

export const hasPermission = (userRole: string, permission: keyof typeof PERMISSIONS):
boolean => {
  return PERMISSIONS[permission].includes(userRole as any);
};
```

Backend Authorization Middleware:

```
// middleware/auth.ts
export const authenticate = async (req: Request, res: Response, next: NextFunction) => {
  try {
    const token = extractTokenFromRequest(req);
    if (!token) {
      throw new ApiError('Access token required', 401);
    }
```

```
    const decoded = JWTService.verifyAccessToken(token);
    const user = await User.findById(decoded.userId);

    if (!user) {
      throw new ApiError('User not found', 401);
    }

    req.user = user;
    next();
  } catch (error) {
    next(new ApiError('Invalid token', 401));
  }
};

export const authorize = (roles: string[]) => {
  return (req: Request, res: Response, next: NextFunction) => {
    if (!req.user) {
      return next(new ApiError('Authentication required', 401));
    }

    if (!roles.includes(req.user.role)) {
      return next(new ApiError('Insufficient permissions', 403));
    }

    next();
  };
};
```

Frontend Route Protection:

```
// components/auth/ProtectedRoute.tsx
interface ProtectedRouteProps {
  requiredRole?: 'user' | 'admin';
  children?: React.ReactNode;
}

export const ProtectedRoute: React.FC<ProtectedRouteProps> = ({
  requiredRole = 'user',
  children
}) => {
  const { user, isLoading } = useAuth();
  const location = useLocation();
```

```
  if (isLoading) {
    return <LoadingSpinner />;
  }

  if (!user) {
    return <Navigate to="/login" state={{ from: location }} replace />;
  }

  if (requiredRole === 'admin' && user.role !== 'admin') {
    return <Navigate to="/unauthorized" replace />;
  }

  return <Outlet />;
};
```

Security Measures

Password Security:

- Minimum 8 characters with complexity requirements
- Bcrypt hashing with salt rounds of 12
- Password reset tokens with 1-hour expiration

JWT Security:

- Access tokens: 7-day expiration
- Refresh tokens: 30-day expiration
- HTTP-only cookies in production
- Token blacklisting on logout

Input Sanitization:

```
// middleware/validation.ts
import mongoSanitize from 'express-mongo-sanitize';
import xss from 'xss';

export const sanitizeInput = (req: Request, res: Response, next: NextFunction) => {
  // Prevent NoSQL injection
  mongoSanitize(req.body);
  mongoSanitize(req.query);
  mongoSanitize(req.params);

  // Prevent XSS attacks
  if (req.body) {
```

```
    Object.keys(req.body).forEach(key => {
      if (typeof req.body[key] === 'string') {
        req.body[key] = xss(req.body[key]);
      }
    });
  }

  next();
};
```

Rate Limiting:

```
// middleware/rateLimit.ts
import rateLimit from 'express-rate-limit';

export const authLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 5, // 5 attempts per window
  message: 'Too many authentication attempts, please try again later',
  standardHeaders: true,
  legacyHeaders: false,
});

export const apiLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // 100 requests per window
  message: 'Too many requests, please try again later',
});
```

7. Payment Integration

Stripe Implementation

Frontend Payment Flow:

```
// components/payment/PaymentForm.tsx
import { loadStripe } from '@stripe/stripe-js';
import {
  Elements,
  CardNumberElement,
  CardExpiryElement,
  CardCvcElement,
```

```
  useStripe,
  useElements,
} from '@stripe/react-stripe-js';

const stripePromise = loadStripe(process.env.REACT_APP_STRIPE_PUBLISHABLE_KEY!);

interface PaymentFormProps {
  takeoffId: string;
  amount: number;
  onSuccess: (purchaseId: string) => void;
}

const PaymentForm: React.FC<PaymentFormProps> = ({ takeoffId, amount, onSuccess }) => {
  const stripe = useStripe();
  const elements = useElements();
  const [isProcessing, setIsProcessing] = useState(false);
  const [paymentError, setPaymentError] = useState<string | null>(null);

  const handleSubmit = async (event: React.FormEvent) => {
    event.preventDefault();

    if (!stripe || !elements) return;

    setIsProcessing(true);
    setPaymentError(null);

    try {
      // Create payment intent
      const { data } = await api.createPaymentIntent({ takeoffId });

      // Confirm payment
      const { error, paymentIntent } = await stripe.confirmCardPayment(
        data.clientSecret,
        {
          payment_method: {
            card: elements.getElement(CardNumberElement)!,
            billing_details: {
              name: 'Customer Name',
              email: 'customer@example.com',
            },
          },
        }
      );
```

```
      if (error) {
        setPaymentError(error.message || 'Payment failed');
      } else if (paymentIntent.status === 'succeeded') {
        // Confirm purchase on backend
        const { data: purchase } = await api.confirmPayment({
          paymentIntentId: paymentIntent.id,
          takeoffId,
        });

        onSuccess(purchase.id);
      }
    } catch (error) {
      setPaymentError('Payment processing failed');
    } finally {
      setIsProcessing(false);
    }
  };

  return (
    <form onSubmit={handleSubmit} className="payment-form">
      <div className="card-element">
        <CardNumberElement options={cardElementOptions} />
      </div>
      <div className="card-row">
        <CardExpiryElement options={cardElementOptions} />
        <CardCvcElement options={cardElementOptions} />
      </div>

      {paymentError && (
        <div className="error-message">{paymentError}</div>
      )}

      <button
        type="submit"
        disabled={!stripe || isProcessing}
        className="payment-button"
      >
        {isProcessing ? 'Processing...' : `Pay ${amount}`}
      </button>
    </form>
  );
};

// Wrapper component with Stripe Elements provider
```

```
export const CheckoutForm: React.FC<PaymentFormProps> = (props) => (
  <Elements stripe={stripePromise}>
    <PaymentForm {...props} />
  </Elements>
);
```

Backend Payment Processing:

```
// services/paymentService.ts
import Stripe from 'stripe';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, {
  apiVersion: '2023-10-16',
});

export class PaymentService {
  async createPaymentIntent(userId: string, takeoffId: string):
Promise<CreatePaymentIntentResponse> {
    // Get takeoff details
    const takeoff = await Takeoff.findById(takeoffId);
    if (!takeoff) {
      throw new ApiError('Takeoff not found', 404);
    }

    // Check if user already purchased this takeoff
    const existingPurchase = await Purchase.findOne({
      userId,
      takeoffId,
      status: 'completed',
    });

    if (existingPurchase) {
      throw new ApiError('You have already purchased this takeoff', 400);
    }

    // Create Stripe payment intent
    const paymentIntent = await stripe.paymentIntents.create({
      amount: takeoff.price * 100, // Convert to cents
      currency: 'usd',
      metadata: {
        userId: userId.toString(),
        takeoffId: takeoffId.toString(),
      },
    });
```

```
  return {
    clientSecret: paymentIntent.client_secret!,
    amount: takeoff.price * 100,
    currency: 'usd',
    takeoffId,
  };
}

async confirmPayment(
  paymentIntentId: string,
  takeoffId: string,
  userId: string
): Promise<Purchase> {
  // Retrieve payment intent from Stripe
  const paymentIntent = await stripe.paymentIntents.retrieve(paymentIntentId);

  if (paymentIntent.status !== 'succeeded') {
    throw new ApiError('Payment not completed', 400);
  }

  // Get takeoff details
  const takeoff = await Takeoff.findById(takeoffId);
  if (!takeoff) {
    throw new ApiError('Takeoff not found', 404);
  }

  // Create purchase record
  const purchase = await Purchase.create({
    userId,
    takeoffId,
    stripePaymentIntentId: paymentIntentId,
    amount: takeoff.price,
    currency: 'usd',
    status: 'completed',
    downloadUrl: await this.generateSecureDownloadUrl(takeoffId),
    downloadExpires: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000), // 30 days
    maxDownloads: 5,
  });

  // Update takeoff download count
  takeoff.downloadCount += 1;
  await takeoff.save();
```

```
    // Send purchase confirmation email
    await EmailService.sendPurchaseConfirmation(userId, takeoff, purchase);

    return purchase;
  }

  private async generateSecureDownloadUrl(takeoffId: string): Promise<string> {
    // Generate secure token
    const token = crypto.randomBytes(32).toString('hex');

    // Store token with expiration (could use Redis for better performance)
    await DownloadToken.create({
      token,
      takeoffId,
      expiresAt: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000), // 30 days
      maxUses: 5,
    });

    return `${process.env.API_URL}/api/downloads/${token}`;
  }
}
```

Webhook Handling

Stripe Webhook Configuration:

```
// routes/webhooks.ts
import express from 'express';
import Stripe from 'stripe';

const router = express.Router();
const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!);

router.post('/stripe', express.raw({ type: 'application/json' }), async (req, res) => {
  const sig = req.headers['stripe-signature'] as string;
  let event: Stripe.Event;

  try {
    event = stripe.webhooks.constructEvent(
      req.body,
      sig,
      process.env.STRIPE_WEBHOOK_SECRET!
    );
```

```
  } catch (err) {
    console.error('Webhook signature verification failed:', err);
    return res.status(400).send('Webhook signature verification failed');
  }

  try {
    switch (event.type) {
      case 'payment_intent.succeeded':
        await handlePaymentSuccess(event.data.object as Stripe.PaymentIntent);
        break;

      case 'payment_intent.payment_failed':
        await handlePaymentFailure(event.data.object as Stripe.PaymentIntent);
        break;

      case 'charge.dispute.created':
        await handleChargeDispute(event.data.object as Stripe.Dispute);
        break;

      default:
        console.log(`Unhandled event type: ${event.type}`);
    }

    res.json({ received: true });
  } catch (error) {
    console.error('Error processing webhook:', error);
    res.status(500).json({ error: 'Webhook processing failed' });
  }
});

async function handlePaymentSuccess(paymentIntent: Stripe.PaymentIntent) {
  // Update purchase status if needed
  const purchase = await Purchase.findOne({
    stripePaymentIntentId: paymentIntent.id,
  });

  if (purchase && purchase.status === 'pending') {
    purchase.status = 'completed';
    await purchase.save();

    // Send confirmation email
    const user = await User.findById(purchase.userId);
    const takeoff = await Takeoff.findById(purchase.takeoffId);
```

```
    if (user && takeoff) {
      await EmailService.sendPurchaseConfirmation(user, takeoff, purchase);
    }
  }
}

async function handlePaymentFailure(paymentIntent: Stripe.PaymentIntent) {
  // Update purchase status
  const purchase = await Purchase.findOne({
    stripePaymentIntentId: paymentIntent.id,
  });

  if (purchase) {
    purchase.status = 'failed';
    await purchase.save();
  }
}

async function handleChargeDispute(dispute: Stripe.Dispute) {
  // Handle dispute - notify admin, update records
  console.log('Charge dispute created:', dispute.id);

  // Could implement admin notification system here
  await EmailService.sendAdminNotification(
    'Payment Dispute',
    `A payment dispute has been created: ${dispute.id}`
  );
}

export default router;
```

Secure File Download Endpoint:

```
// routes/downloads.ts
import { v2 as cloudinary } from 'cloudinary';
import axios from 'axios';

// Configure Cloudinary
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});
```

```
/**
 * Secure File Download Endpoint with Cloudinary
 */
router.get('/:token', async (req, res) => {
  try {
    const { token } = req.params;

    // Find and validate download token
    const downloadToken = await DownloadToken.findOne({
      token,
      expiresAt: { $gt: new Date() },
      usesRemaining: { $gt: 0 },
    });

    if (!downloadToken) {
      return res.status(404).json({ error: 'Invalid or expired download link' });
    }

    // Get takeoff and file details
    const takeoff = await Takeoff.findById(downloadToken.takeoffId);
    if (!takeoff || !takeoff.files.length) {
      return res.status(404).json({ error: 'File not found' });
    }

    const file = takeoff.files[0]; // Assuming single file per takeoff

    // Generate a temporary signed URL from Cloudinary (expires in 1 hour)
    const signedUrl = cloudinary.url(file.cloudinaryPublicId, {
      resource_type: file.resourceType || 'raw',
      type: 'upload',
      sign_url: true,
      expires_at: Math.floor(Date.now() / 1000) + 3600, // 1 hour from now
    });

    // Fetch file from Cloudinary
    const response = await axios({
      method: 'GET',
      url: signedUrl,
      responseType: 'stream',
    });

    // Update download token usage
    downloadToken.usesRemaining -= 1;
    await downloadToken.save();
```

```
    // Update takeoff download count
    takeoff.downloadCount += 1;
    await takeoff.save();

    // Set appropriate headers
    res.set({
      'Content-Type': getContentType(file.originalName),
      'Content-Disposition': `attachment; filename="${file.originalName}"`,
      'Content-Length': response.headers['content-length'],
    });

    // Stream file to response
    response.data.pipe(res);

  } catch (error) {
    console.error('Download error:', error);
    res.status(500).json({ error: 'Download failed' });
  }
});

/**
 * Generate Secure Download Link
 */
router.post('/generate-link/:takeoffId', authenticateToken, async (req, res) => {
  try {
    const { takeoffId } = req.params;
    const { expiresIn = 24 } = req.body; // hours

    // Verify takeoff exists and user has access
    const takeoff = await Takeoff.findById(takeoffId);
    if (!takeoff) {
      return res.status(404).json({ error: 'Takeoff not found' });
    }

    // Create download token
    const downloadToken = new DownloadToken({
      takeoffId,
      userId: req.user.id,
      token: generateSecureToken(),
      expiresAt: new Date(Date.now() + expiresIn * 60 * 60 * 1000),
      usesRemaining: 3, // Allow 3 downloads
    });
```

```
    await downloadToken.save();

    const downloadUrl = `${process.env.API_URL}/api/downloads/${downloadToken.token}`;

    res.json({
      downloadUrl,
      expiresAt: downloadToken.expiresAt,
      usesRemaining: downloadToken.usesRemaining,
    });

  } catch (error) {
    console.error('Generate link error:', error);
    res.status(500).json({ error: 'Failed to generate download link' });
  }
});

/**
 * File Upload Handler with Cloudinary
 */
const uploadToCloudinary = async (file, folder = 'takeoffs') => {
  return new Promise((resolve, reject) => {
    const uploadStream = cloudinary.uploader.upload_stream(
      {
        folder: `${process.env.CLOUDINARY_FOLDER}/${folder}`,
        resource_type: 'auto', // Automatically detect file type
        use_filename: true,
        unique_filename: true,
      },
      (error, result) => {
        if (error) {
          reject(error);
        } else {
          resolve({
            cloudinaryPublicId: result.public_id,
            cloudinaryUrl: result.secure_url,
            resourceType: result.resource_type,
            width: result.width,
            height: result.height,
          });
        }
      }
    );

    uploadStream.end(file.buffer);
```

```
  });
};

/**
 * Delete File from Cloudinary
 */
const deleteFromCloudinary = async (publicId, resourceType = 'raw') => {
  try {
    const result = await cloudinary.uploader.destroy(publicId, {
      resource_type: resourceType,
    });
    return result;
  } catch (error) {
    console.error('Cloudinary deletion error:', error);
    throw error;
  }
};

// Helper function to determine content type
const getContentType = (filename) => {
  const ext = filename.split('.').pop().toLowerCase();
  const contentTypes = {
    'xlsx': 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
    'xls': 'application/vnd.ms-excel',
    'csv': 'text/csv',
    'pdf': 'application/pdf',
    'jpg': 'image/jpeg',
    'jpeg': 'image/jpeg',
    'png': 'image/png',
    'gif': 'image/gif',
    'webp': 'image/webp',
  };
  return contentTypes[ext] || 'application/octet-stream';
};

// Helper function to generate secure token
const generateSecureToken = () => {
  return require('crypto').randomBytes(32).toString('hex');
};
```

This completes the comprehensive technical documentation for ProTakeoff.ai Phase 1. The documentation covers all major aspects including architecture, setup, API endpoints, authentication, authorization, and payment processing with detailed code examples and implementation strategies.