Technology Stack & Libraries Documentation

Table of Contents

Why Next.js + MongoDB Over Django + SQL

Strategic Decision Factors

1. Team Expertise & Familiarity PRIMARY REASON

- **Current Knowledge Base**: Our development team has extensive experience with JavaScript/TypeScript and React ecosystems
- **Reduced Learning Curve**: Using Next.js means we can leverage existing skills rather than learning Python and Django conventions
- **Faster Development**: Familiarity with the stack allows us to build features quickly without constant documentation lookups
- **Maintenance Efficiency**: Team can troubleshoot and optimize code more effectively in a familiar environment

2. Full-Stack JavaScript/TypeScript

- **Single Language**: Use TypeScript across frontend, backend, and database queries
- **Code Sharing**: Share types, utilities, and validation schemas between client and server
- **Mental Model**: No context switching between Python (backend) and JavaScript (frontend)
- **Type Safety**: End-to-end type safety from database to UI

3. Modern Developer Experience

- **Hot Module Replacement**: Instant feedback during development
- **API Routes**: Serverless function approach is simpler than Django views
- **File-Based Routing**: Intuitive routing system vs Django URLs configuration

- **Built-in Optimization**: Automatic code splitting, image optimization, and performance features

4. MongoDB vs SQL for This Use Case

- **Flexible Schema**: Transportation/logistics data has varying structures (different shipment types, custom fields)
- **Document Model**: Natural fit for nested data (addresses, tracking history, packages)
- **Rapid Iteration**: Schema changes don't require migrations
- **JSON-Native**: Easy to work with JSON data from frontend forms
- **Horizontal Scaling**: Better suited for future growth

Note: We acknowledge SQL databases excel at complex joins and transactions. For this project, the benefits of MongoDB's flexibility and our team's NoSQL experience outweigh those advantages.

5. Deployment & Scalability

- **Vercel Integration**: One-click deployment with Next.js on Vercel
- **Serverless Architecture**: Auto-scaling without infrastructure management
- **Edge Computing**: Deploy API routes to edge locations for lower latency
- **Static Generation**: Pre-render pages for better performance

6. Ecosystem & Community

- **Rich Package Ecosystem**: NPM has extensive libraries for any feature
- **Active Community**: Large Next.js and MongoDB communities for support
- **Modern Tooling**: Better integration with modern dev tools (ESLint, Prettier, TypeScript)

Why NOT Django + SQL

While Django is an excellent framework, we chose against it because:

1. **Python Learning Curve**: Team would need to learn Python idioms, Django ORM, and template language
2. **Frontend Integration**: Separate frontend (React) + backend (Django) requires more configuration
3. **Type Safety Gap**: Python's dynamic typing doesn't integrate well with TypeScript frontend
4. **Deployment Complexity**: Django requires traditional server setup vs Next.js serverless simplicity
5. **Rapid Prototyping**: Next.js allows faster iteration for startup environment

---

Core Framework

Next.js 15 (App Router)

Version: `^15.1.6`
Purpose: Full-stack React framework with server-side rendering and API routes

Why Next.js: - **File-based Routing**: Intuitive page structure in `app/` directory - **Server Components**: Reduce client-side JavaScript, improve performance - **API Routes**: Built-in backend without separate server - **SEO Optimization**: Server-side rendering for better search engine visibility - **Image Optimization**: Automatic image resizing and lazy loading - **Code Splitting**: Automatic bundle optimization per route

Key Features Used: - App Router (latest routing paradigm) - Server Actions for form submissions - Middleware for authentication - Edge Runtime for API routes - Static Site Generation (SSG) for marketing pages

Alternatives Considered: - **Create React App**: No SSR, no API routes, deprecated - **Vite + React**: Great for SPAs but requires separate backend - **Remix**: Similar to Next.js but smaller ecosystem

---

React 19

Version: `^19.0.0`
Purpose: UI component library

Why React: - Industry standard with massive ecosystem - Component reusability across the application - Virtual DOM for efficient updates - Hooks for state management and side effects - React Server Components support in Next.js

Key Features Used: - Functional components with hooks - `useState`, `useEffect`, `useContext` - Custom hooks for business logic - Client/Server component separation

---

TypeScript 5

Version: `^5`
Purpose: Type-safe JavaScript superset

Why TypeScript: - **Catch Errors Early**: Type checking prevents runtime bugs - **Better IDE Support**: Autocomplete and IntelliSense - **Self-Documenting Code**: Types serve as inline documentation - **Refactoring Confidence**: Safe code reorganization - **Team Collaboration**: Clear contracts between functions

Configuration: - Strict mode enabled - Path aliases (`@/` for imports) - React JSX support

---

Database & ORM

MongoDB

Cloud Service: MongoDB Atlas
Purpose: NoSQL document database

Why MongoDB: - **Flexible Schema**: Easy to add fields without migrations - **Document Model**: Natural fit for complex nested data (addresses, tracking history) - **JSON-Native**: Works naturally with JavaScript objects - **Scalability**: Horizontal scaling for future growth - **Team Experience**: Team already familiar with MongoDB

Collections: - `users` - System accounts - `clients` - Customer records - `drivers` - Driver profiles - `vehicles` - Fleet management - `shipments` - Delivery packages - `deliverytours` - Route assignments - `invoices` - Billing records - `payments` - Payment tracking - `incidents` - Issue reports - `complaints` - Customer complaints

---

Mongoose 8

Version: `^8.9.4`
Purpose: MongoDB ODM (Object Document Mapper)

Why Mongoose: - **Schema Validation**: Define structure for MongoDB documents - **Type Safety**: Integration with TypeScript - **Middleware**: Pre/post hooks for business logic - **Virtuals**: Computed properties - **Population**: Automatic reference resolution (like SQL joins) - **Query Builder**: Intuitive API for complex queries

Key Features Used: - Schema definitions with validation - Static methods for common queries - Instance methods for model operations - Pre-save hooks for auto-generation (IDs, numbers) - Indexes for query optimization - Virtuals for computed fields

Example Schema:

```
const clientSchema = new Schema({
  code: { type: String, unique: true },
  firstName: { type: String, required: true },
  email: { type: String, required: true },
  // ... more fields
});

clientSchema.statics.findByEmail = function(email) {
```

```
  return this.findOne({ email });
};
```

Alternatives: - **Prisma**: Great for SQL, less mature for MongoDB - **Native MongoDB Driver**: Too low-level, no schema validation - **TypeORM**: Better for SQL databases

---

Authentication

NextAuth.js v5 (@auth/core)

Version: `^5.0.0-beta.25`
Purpose: Authentication for Next.js applications

Why NextAuth: - **Next.js Integration**: Built specifically for Next.js - **Multiple Providers**: Email, Google, GitHub, etc. - **Session Management**: JWT or database sessions - **CSRF Protection**: Built-in security - **TypeScript Support**: Full type safety

Features Used: - Credentials provider (email/password) - JWT session strategy - Custom callbacks for role-based access - Middleware for route protection

Configuration:

```
providers: [
  Credentials({
    credentials: {
      email: {},
      password: {}
    },
    authorize: async (credentials) => {
      // Custom authentication logic
    }
  })
]
```

Roles Implemented: - `admin` - Full system access - `agent` - Manage operations - `driver` - Execute deliveries

---

UI Components & Styling

Tailwind CSS 3

Version: `^3.4.17`
Purpose: Utility-first CSS framework

Why Tailwind: - **Rapid Development**: Build UIs without writing custom CSS - **Consistency**: Design system through configuration - **Performance**: Purges unused styles in production - **Responsive Design**: Mobile-first breakpoint system - **Dark Mode**: Built-in dark mode support

Custom Configuration: - Custom color palette (zinc, green, blue, red) - Extended spacing and sizing - Custom animations - Dark mode class strategy

Example Usage:

```
<div className="bg-zinc-900 p-4 rounded-2xl border border-zinc-800">
  <h2 className="text-xl font-bold text-zinc-100">Title</h2>
</div>
```

Radix UI

Version: Multiple packages (`@radix-ui/*`)
Purpose: Unstyled, accessible UI component primitives

Why Radix: - **Accessibility**: WCAG compliant out of the box - **Unstyled**: Full styling control with Tailwind - **Composable**: Build complex components from primitives - **Keyboard Navigation**: Proper focus management - **Type Safe**: Full TypeScript support

Components Used: - Dialog/Modal - For forms and confirmations - Dropdown Menu - Navigation and actions - Select - Form inputs - Checkbox - Multi-select options - Tabs - Content organization - Tooltip - Contextual help - Popover - Additional info display

---

shadcn/ui

Purpose: Re-usable component collection built on Radix UI

Why shadcn: - **Copy-Paste**: Components live in your codebase, not node_modules - **Customizable**: Full control over implementation - **Best Practices**: Well-structured component patterns - **Tailwind Integration**: Designed for Tailwind CSS

Components Implemented: - Button, Input, Textarea - Dialog, Sheet (side panel) - Select, Checkbox - Badge, Card - Table, DataTable - Toast notifications (Sonner)

Lucide React

Version: `^0.468.0`
Purpose: Icon library

Why Lucide: - **Consistent Design**: Cohesive icon set - **Tree-Shakeable**: Only import icons you use - **React Components**: Icons as React components - **Customizable**: Size, color, stroke width - **Large Library**: 1000+ icons

Usage:

```
import { Package, Truck, AlertTriangle } from 'lucide-react';
<Package className="h-5 w-5 text-green-500" />
```

Sonner

Version: `^1.7.3`
Purpose: Toast notifications

Why Sonner: - **Beautiful UI**: Modern toast design - **Stacking**: Multiple toasts stack nicely - **Promise Support**: Auto-update based on async operations - **Customizable**: Full control over appearance - **Accessibility**: Screen reader support

Usage:

```
toast.success("Tour started!");
toast.error("Failed to load data");
```

Form Handling & Validation

React Hook Form

Version: `^7.54.2`
Purpose: Performant form library with easy validation

Why React Hook Form: - **Performance**: Uncontrolled components reduce re-renders - **Easy Validation**: Integrates with Zod - **Developer Experience**: Simple API - **TypeScript**: Full type inference - **Small Bundle**: ~10KB minified

Features Used: - `useForm` hook for form state - Validation with Zod resolver - Error handling and display - Default values and form reset

Example:

```
const form = useForm<FormValues>({
  resolver: zodResolver(schema),
  defaultValues: { name: "" }
});

const onSubmit = (data: FormValues) => {
  // Submit logic
};
```

Zod

Version: `^3.24.1`
Purpose: TypeScript-first schema validation

Why Zod: - **Type Inference**: Automatically infers TypeScript types from schemas - **Composable**: Build complex schemas from simple ones - **Error Messages**: Detailed validation errors - **Runtime Safety**: Validate API responses and form inputs - **Zero Dependencies**: Lightweight and fast

Usage:

```
const schema = z.object({
  email: z.string().email(),
  password: z.string().min(8),
  role: z.enum(['admin', 'agent', 'driver'])
});

type FormData = z.infer<typeof schema>;
```

Schemas Defined: - User creation/update - Client management - Shipment creation - Tour management - Invoice generation - All API request/response validation

---

File Upload & Storage

Cloudinary SDK

Version: `^2.5.3`
Purpose: Cloud-based media management

Why Cloudinary: - **Free Tier**: Generous free plan for startups - **CDN**: Fast image delivery worldwide - **Transformations**: On-the-fly image resizing/optimization - **Upload API**: Simple integration - **Security**: Signed URLs for protected content

Features Used: - Image upload (incidents, POD, complaints) - Auto-optimization - Folder organization - Secure URLs

Upload Flow: 1. Client converts file to base64 2. POST to `/api/upload` 3. Server uploads to Cloudinary 4. Return URL to client 5. Store URL in MongoDB

Utilities & Tools

date-fns

Version: `^4.1.0`
Purpose: Modern date utility library

Why date-fns: - **Tree-Shakeable**: Only import functions you need - **Immutable**: Pure functions, no mutations - **TypeScript**: Full type support - **i18n**: Internationalization support - **Modern**: Uses native Date objects

Usage:

```
import { format, addDays, isAfter } from 'date-fns';
format(new Date(), 'yyyy-MM-dd');
```

---

clsx

Version: `^2.1.1`
Purpose: Conditional className utility

Why clsx: - **Tiny**: 228 bytes - **Fast**: Optimized for performance - **Simple API**: Easy conditional classes

Usage:

```
import clsx from 'clsx';
<div className={clsx(
  'base-class',
  isActive && 'active',
  { 'error': hasError }
)} />
```

---

tailwind-merge

Version: `^2.6.0`
Purpose: Merge Tailwind classes without conflicts

Why tailwind-merge: - **Conflict Resolution**: Later classes override earlier ones - **Clean Output**: No duplicate utility classes - **Performance**: Optimized merging algorithm

Usage:

```
import { cn } from '@/lib/utils';
const className = cn(
  'p-4 bg-red-500',
  'p-8 bg-blue-500' // p-8 and bg-blue-500 win
);
```

---

bcryptjs

Version: `^2.4.3`
Purpose: Password hashing

Why bcryptjs: - **Security**: Industry-standard password hashing - **Salt**: Automatic salt generation - **Configurable**: Adjustable work factor - **Pure JavaScript**: No native dependencies

Usage:

```
const hash = await bcrypt.hash(password, 10);
const isValid = await bcrypt.compare(password, hash);
```

Drag & Drop

dnd-kit

Version: Multiple packages (`@dnd-kit/*`)
Purpose: Modern drag-and-drop toolkit

Why dnd-kit: - **Accessibility**: Keyboard navigation support - **Performance**: Optimized for large lists - **Flexible**: Works with any framework - **TypeScript**: Full type safety - **Modular**: Use only what you need

Packages: - `@dnd-kit/core` - Core functionality - `@dnd-kit/sortable` - Sortable lists - `@dnd-kit/utilities` - Helper functions

Usage: - Shipment reordering in tours - Visual feedback during drag - Touch support for mobile

---

Development Tools

ESLint

Version: `^9`
Purpose: JavaScript/TypeScript linter

Configuration: - Next.js recommended rules - TypeScript strict rules - React hooks rules - Custom rules for code quality

---

Turbopack

Built into Next.js 15
Purpose: Fast bundler replacing Webpack

Benefits: - 700x faster than Webpack for large apps - Incremental compilation - Memory efficient - Better error messages

---

Package Management

npm

Purpose: Node package manager

Scripts Defined: - `npm run dev` - Start development server - `npm run build` - Production build - `npm run start` - Start production server - `npm run lint` - Run ESLint

---

Key Architectural Decisions

Monorepo Structure

Single codebase for frontend and backend in Next.js rather than separate repositories.

Benefits: - Shared types and utilities - Single deployment - Easier local development - Type safety across stack

---

API Client Pattern

Centralized API client (`lib/api-client.ts`) instead of scattered fetch calls.

Benefits: - Consistent error handling - Type-safe API calls - Easy to mock for testing - Single source of truth for endpoints

---

Component Organization

shadcn/ui components in `components/ui/` directory for reusability.

Benefits: - Consistent UI across app - Easy to update globally - Clear component hierarchy - Better developer experience

---

Comparison Summary: Next.js + MongoDB vs Django + SQL

Aspect	Next.js + MongoDB	Django + SQL
Learning Curve	Already familiar	Need to learn
Language	One (TypeScript)	Two (Python + JS)
Development Speed	Faster (familiar)	Slower (learning)
Type Safety	End-to-end	Python dynamic
Deployment	Serverless (easy)	Traditional server
Schema Flexibility	High	Low (migrations)
Complex Queries	Manual	SQL powerful
Transactions	Limited	ACID support
Team Expertise	High	Low
MVP Speed	Very fast	Moderate

Final Verdict: For this project, with this team, Next.js + MongoDB is the superior choice primarily due to **existing expertise** and the need for **rapid development**. The trade-offs in relational querying are acceptable for our logistics use case.