Routie Roo - Technical Summary & Architecture

Project: Routie Roo
Version: 40d03018
Last Updated: December 1, 2024
Status: Production (Multi-User SaaS)

Executive Summary

Routie Roo is a comprehensive driving route planning and execution platform that integrates with Google Contacts, Google Calendar, and Google Maps to help users create, manage, and execute optimized routes for their contacts. Originally built for home healthcare route planning, it has evolved into a full-featured multi-user SaaS platform with real-time execution tracking, team collaboration, and calendar integration.

Core Value Proposition:

- Sync contacts from Google with addresses, phone numbers, and photos
- Create optimized driving routes with Google Maps integration
- Track route execution in real-time (complete/miss stops, add notes)
- Share routes with drivers via public links
- Integrate with Google Calendar for scheduling
- Manage important dates and reminders for contacts

Technology Stack

Frontend

Core Framework:

- Next.js 14 with App Router
- TypeScript for type safety
- React 19 for UI components
- Wouter for lightweight client-side routing

Styling & UI:

- Tailwind CSS 4 for utility-first styling
- shadcn/ui components (built on Radix UI primitives)
- Custom mobile.css for touch-friendly optimizations
- Responsive design with mobile-first approach

State Management:

- tRPC React Query hooks for data fetching
- React Hook Form for form management
- Zod for runtime validation
- SuperJSON for Date/Map/Set serialization

Maps Integration:

- Google Maps JavaScript API
- Custom React wrapper components
- Real-time route visualization
- Marker clustering and custom icons

Backend

API Layer:

- tRPC 11 for end-to-end type safety
- Express 4 as HTTP server
- Node.js runtime
- SuperJSON for complex type serialization

Database:

- MySQL (TiDB) for production
- Drizzle ORM for type-safe queries
- Schema-first migrations
- Connection pooling

Authentication:

- Google OAuth 2.0
- JWT sessions with httpOnly cookies
- Token refresh handling
- Role-based access control (admin/user)

External APIs:

- Google People API (contacts sync with read/write)
- Google Calendar API (event management)
- Google Maps Routes API (route optimization)
- Google Maps Geocoding API (address validation)
- Google Maps Places API (address autocomplete)

File Storage:

- AWS S3 for document uploads
- Pre-signed URLs for secure access
- Organized by user and contact

Architecture Overview

Request Flow

```
User Browser
    ↓
Next.js Frontend (React)
    ↓
tRPC Client (React Query)
    ↓
tRPC Server (Express)
    ↓
Business Logic (TypeScript)
    ↓
Drizzle ORM
    ↓
MySQL Database (TiDB)
```

Authentication Flow

1. User clicks "Sign in with Google"
2. Redirect to Google OAuth consent screen
3. User grants permissions (contacts, calendar)
4. Google redirects to /api/oauth/callback with code
5. Exchange code for access token + refresh token
6. Store tokens in database
7. Create JWT session cookie
8. Redirect to home page
9. All subsequent requests include session cookie
10. tRPC context extracts user from JWT
11. Procedures access ctx.user for authorization

Data Sync Flow

```
Google Contacts
    ↓ (OAuth + People API)
cached_contacts table
    ↓ (User selection)
route_waypoints table
    ↓ (Route execution)
Status updates, notes, reschedules
    ↓ (Two-way sync)
Google Contacts (address/phone updates)
```

Calendar Integration Flow

```
1. User creates route
2. User clicks "Add to Calendar"
3. Select Google Calendar from list
4. Create individual event for each waypoint
5. Store calendarEventId in route_waypoints
6. When waypoint edited: update Google Calendar event
7. When waypoint deleted: delete Google Calendar event
8. When waypoint added: create new Google Calendar event
```

Database Schema

Core Tables

users

- Primary key: `id` (auto-increment)
- OAuth: `openId`, `email`, `name`, `loginMethod`
- Tokens: `accessToken`, `refreshToken`, `tokenExpiry`
- Role: `role` (enum: admin, user)
- Preferences: `distanceUnit`, `callingService`, `defaultStartingPoint`, `autoArchiveDays`, `defaultStopType`, `allowMultipleVisits`

- Timestamps: `createdAt`, `updatedAt`, `lastSignedIn`

cached_contacts

- Primary key: `id` (auto-increment)
- Foreign key: `userId` → users.id
- Google: `googleContactId`, `resourceName`, `etag`
- Basic: `name`, `email`, `photoUrl`
- Location: `address`, `latitude`, `longitude`
- Communication: `phoneNumbers` (JSON array)
- Organization: `labels` (JSON array), `isActive` (boolean)
- Timestamps: `createdAt`, `updatedAt`, `lastSynced`

routes

- Primary key: `id` (auto-increment)
- Foreign key: `userId` → users.id, `folderId` → folders.id
- Metadata: `name`, `notes`, `distance`, `duration`
- Settings: `optimized` (boolean), `startingPoint`
- Sharing: `shareToken` (UUID), `isPublic` (boolean)
- Calendar: `googleCalendarId`, `calendarEventIds` (JSON)
- Status: `completedAt`, `archivedAt`, `hasManualOrder`
- Schedule: `scheduledDate`
- Timestamps: `createdAt`, `updatedAt`

route_waypoints

- Primary key: `id` (auto-increment)
- Foreign keys: `routeId` → routes.id, `contactId` → cached_contacts.id
- Contact: `contactName`, `address`, `phoneNumbers` (JSON), `photoUrl`, `contactLabels` (JSON)
- Location: `latitude`, `longitude`

- Stop: `stopType`, `stopColor`, `stopNumber`, `executionOrder`
- Gap: `isGapStop`, `gapDuration`, `gapDescription`
- Status: `status` (enum: pending, in_progress, complete, missed)
- Execution: `completedAt`, `missedReason`, `executionNotes`, `rescheduledDate`, `needsReschedule`
- Calendar: `calendarEventId`
- Timestamps: `createdAt`, `updatedAt`

folders

- Primary key: `id` (auto-increment)
- Foreign key: `userId` → users.id
- Data: `name`
- Timestamps: `createdAt`, `updatedAt`

stop_types

- Primary key: `id` (auto-increment)
- Foreign key: `userId` → users.id
- Data: `name`, `color` (hex)
- Timestamps: `createdAt`, `updatedAt`

starting_points

- Primary key: `id` (auto-increment)
- Foreign key: `userId` → users.id
- Data: `name`, `address`
- Timestamps: `createdAt`, `updatedAt`

route_notes

- Primary key: `id` (auto-increment)
- Foreign keys: `routeId` → routes.id, `userId` → users.id
- Data: `content`

- Timestamps: `createdAt`, `updatedAt`

contact_documents

- Primary key: `id` (auto-increment)
- Foreign keys: `contactId` → cached_contacts.id, `userId` → users.id
- Data: `fileName`, `fileUrl`, `fileSize`, `mimeType`
- Timestamps: `uploadedAt`

scheduler_notes

- Primary key: `id` (auto-increment)
- Foreign key: `userId` → users.id
- Data: `content`, `completed` (boolean), `completedAt`
- Position: `positionX`, `positionY`, `width`, `height`, `isCollapsed`
- Timestamps: `createdAt`, `updatedAt`

reschedule_history

- Primary key: `id` (auto-increment)
- Foreign keys: `waypointId` → route_waypoints.id, `routeId` → routes.id, `userId` → users.id
- Data: `contactName`, `address`, `originalDate`, `rescheduledDate`, `reason`
- Timestamps: `createdAt`

important_dates

- Primary key: `id` (auto-increment)
- Foreign keys: `contactId` → cached_contacts.id, `userId` → users.id
- Data: `dateType`, `date`, `notes`, `reminderSent`
- Timestamps: `createdAt`, `updatedAt`

label_colors

- Primary key: `id` (auto-increment)
- Foreign key: `userId` → users.id

- Data: `labelName`, `color` (hex)
- Timestamps: `createdAt`, `updatedAt`

Key Features Deep Dive

Contact Management

Sync Process:

1. User clicks "Sync Contacts"
2. OAuth redirect to Google for permissions
3. Fetch all contacts from Google People API
4. Parse contact data (name, email, address, phone, photo, labels)
5. Geocode addresses using Google Maps Geocoding API
6. Store in cached_contacts table with etag for change detection
7. Display in "Your Kangaroo Crew" section

Two-Way Sync:

- When user edits contact address or phone numbers
- Changes save to local database
- If contactId exists, sync back to Google Contacts
- Use etag for conflict detection
- Automatic token refresh if expired
- Graceful error handling (logs warning but doesn't fail)

Contact Features:

- Display contact photos or initials
- Multiple phone numbers with labels (mobile, work, home)
- Contact labels/groups from Google (colored badges)
- Mark contacts as active/inactive (soft delete)
- Edit contact details with Google sync

- Important dates tracking (birthdays, renewals, anniversaries)
- Custom comment options per user
- Document uploads per contact (S3 storage)
- Bulk document upload by label
- Address validation with Google Maps
- Changed addresses report for manual Google sync

Route Planning

Route Creation Flow:

1. User selects contacts from “Your Kangaroo Crew”
2. Enter route name, notes, scheduled date
3. Select starting point (saved or custom)
4. Select folder for organization
5. Click “Create Route”
6. Backend validates all contacts have addresses
7. Backend calls Google Maps Routes API with waypoints
8. API returns optimized order, distance, duration, polyline
9. Save route and waypoints to database
10. Redirect to route detail page

Route Optimization:

- Uses Google Maps Routes Optimize Waypoints API
- Traveling salesman problem (TSP) solver
- Minimizes total driving time
- Option to disable optimization (manual order)
- Re-optimize button to recalculate after adding stops

Custom Stop Types:

- Users define custom stop types (e.g., Eval, OASIS, Visit, RE, DC)
- Assign colors to each type (hex codes)

- Default stop type in settings for auto-assignment
- Individual stop types editable per waypoint
- Stop type colors display on map markers
- Legend shows all stop types used in route

Gap Stops:

- Non-contact time blocks (lunch, meetings, travel)
- Specify name, duration (minutes), optional description
- Insert after specific stop number
- Display with cute emoji in stop list
- Excluded from map (no physical location)
- Duration included in calendar timing calculations

Route Features:

- Folder organization for categorization
- Schedule routes with date picker
- Route notes for context
- Starting point (custom or saved)
- Manual waypoint ordering option
- Multiple visits to same contact (optional setting)
- Automatic route completion detection
- Archive completed routes (manual or auto)

Route Execution

Status Tracking:

- Waypoint status: pending, in_progress, complete, missed
- Completion timestamps
- Missed stop reasons
- Execution notes per stop
- Reschedule missed stops with date picker

- Progress bar showing completion percentage

Execution Controls:

- Complete button (green)
- Miss button (red) with reason dialog
- Add Note button
- Reschedule button for missed stops
- Edit Details button (name, stop type, address, phone)
- Remove waypoint button

Bulk Actions:

- Complete All Remaining button
- Mark All as Missed button
- Batch status updates

Reordering:

- Stop number input fields for reordering
- Save Order button appears when changed
- Updates executionOrder field
- hasManualOrder flag tracks manual reordering

Manager Dashboard:

- View all missed stops across all routes
- Filter by needs reschedule vs rescheduled
- Route links for easy navigation
- Reschedule history tracking

Map Visualization

Map Features:

- Interactive Google Maps display
- Numbered markers (1, 2, 3…) matching waypoint list

- Custom stop type colors on markers
- Dual-color markers (label color center + stop type color border)
- Route polyline with directions
- Sticky map on desktop (scrolls with waypoints)
- Routie Roo mascot for starting point
- Gap stops excluded from map

Marker Logic:

- If contact has exactly one colored label: dual-color marker
- Otherwise: stop type color only
- Numbered labels for easy correlation with list
- Custom shapes for pickup/delivery/visit

Map Updates:

- Real-time updates when waypoints added/removed
- Automatic recalculation when addresses edited
- Map resize trigger for mobile rendering

Sharing & Collaboration

Share Flow:

1. User clicks “Share” button on route
2. Backend generates unique UUID shareToken
3. Copy shareable link: https://routieroo.manus.space/shared/{shareToken}
4. Send link to driver (SMS, email, etc.)
5. Driver opens link (no login required)
6. Driver sees route map, waypoints, phone numbers
7. Driver can mark stops complete/missed, add notes, reschedule
8. Updates sync back to creator’s account

Driver Capabilities:

- View route map with numbered markers
- See all waypoint details (name, address, phone, stop type)
- Mark stops as complete or missed
- Add execution notes
- Reschedule missed stops
- Call or text contacts (tel:/sms: links)
- Reorder stops via drag-and-drop or number inputs
- All changes persist to database

Security:

- Unique UUID tokens per route (non-enumerable)
- Token revocation capability
- Read-only for drivers (can’ t delete route)
- No authentication required for drivers

Calendar Integration

Add to Calendar Flow:

1. User clicks “Add to Calendar” on route
2. Select Google Calendar from dropdown
3. Backend creates individual event for each waypoint
4. Event details:
    - Title: “{stopType}: {contactName}”
    - Location: waypoint address
    - Start time: calculated from route start + previous durations
    - Duration: user’ s default stop duration setting
    - Description: route notes
5. Store calendarEventId in route_waypoints
6. Display “On Calendar” badge on route

Automatic Calendar Sync:

- When waypoint added: create new calendar event
- When waypoint edited: update calendar event (title, location, time)
- When waypoint removed: delete calendar event
- When route removed from calendar: delete all events
- Token refresh handling for expired tokens
- Graceful error handling

Calendar Features:

- View all calendars in unified view
- Calendar sidebar with visibility toggles
- Day/Week/Month views
- Edit Google Calendar events (title, time, location, description)
- Remove routes from calendar
- Rescheduled stops appear on calendar
- Gap stop duration included in timing calculations
- Starting points excluded from calendar events

Settings & Preferences

User Preferences:

- Calling service (Phone, Google Voice, WhatsApp, Skype, FaceTime)
- Distance unit (kilometers vs miles)
- Default starting point
- Auto-archive settings (7/30/60/90 days)
- Default stop type for route creation
- Allow multiple visits to same contact
- Default stop duration for calendar events
- Calendar event duration mode (stop only vs include drive time)

Custom Configurations:

- Custom stop types with colors

- Custom comment options for contacts
- Important date types
- Email reminders for important dates
- Saved starting points management
- Label colors for map markers

Admin Features

User Management:

- View all users
- Merge duplicate accounts
- Transfer data between users
- Delete users
- Promote users to admin

Reports & Dashboards:

- Missed stops dashboard (all users)
- Reschedule history tracking
- Archive management
- Changed addresses report
- Reminder history log

Mobile Optimization

Responsive Design:

- Mobile-first approach with Tailwind breakpoints
- Bottom tab navigation (Home, Routes, Calendar, Settings, More)
- Slide-out menu for additional navigation
- Sticky compact header
- Safe area insets for notched devices

Touch Optimization:

- 44px minimum touch targets (touch-target class)
- 16px font size on inputs (prevents iOS zoom)
- Full-width buttons on mobile
- Swipe gestures on contact cards (Call, Text, Navigate)
- Pull-to-refresh on home page
- Floating action button for Add Contact

Mobile-Specific Features:

- Expandable contact details
- Mobile actions dropdown menu
- Sticky note toggle button
- Responsive calendar views
- Mobile-optimized forms
- Vertical button stacking

API Endpoints

tRPC Procedures

Authentication:

- `auth.me` - Get current user
- `auth.logout` - Clear session cookie

Contacts:

- `contacts.list` - Get all contacts for user
- `contacts.sync` - Sync contacts from Google
- `contacts.update` - Update contact details (with Google sync)
- `contacts.toggleActive` - Mark contact active/inactive
- `contacts.delete` - Delete contact

- `contacts.getDocuments` - Get documents for contact
- `contacts.uploadDocument` - Upload document to S3
- `contacts.deleteDocument` - Delete document from S3

Routes:

- `routes.list` - Get all routes for user
- `routes.create` - Create new route with optimization
- `routes.getById` - Get route by ID
- `routes.update` - Update route metadata
- `routes.delete` - Delete route
- `routes.copyRoute` - Duplicate route
- `routes.archive` - Archive route
- `routes.unarchive` - Restore archived route
- `routes.addWaypoint` - Add contact to route
- `routes.removeWaypoint` - Remove waypoint from route
- `routes.updateWaypoint` - Update waypoint details (with Google sync)
- `routes.updateWaypointStatus` - Mark waypoint complete/missed
- `routes.reorderWaypoints` - Save new waypoint order
- `routes.reoptimize` - Recalculate route optimization
- `routes.addGapStop` - Add gap stop to route
- `routes.exportCSV` - Export route to CSV

Folders:

- `folders.list` - Get all folders for user
- `folders.create` - Create new folder
- `folders.update` - Rename folder
- `folders.delete` - Delete folder

Stop Types:

- `stopTypes.list` - Get custom stop types for user
- `stopTypes.create` - Create custom stop type
- `stopTypes.update` - Update stop type
- `stopTypes.delete` - Delete stop type

Starting Points:

- `startingPoints.list` - Get saved starting points
- `startingPoints.create` - Create starting point
- `startingPoints.update` - Update starting point
- `startingPoints.delete` - Delete starting point

Calendar:

- `calendar.getEvents` - Get all calendar events
- `calendar.createEvent` - Create calendar event
- `calendar.updateEvent` - Update calendar event
- `calendar.deleteEvent` - Delete calendar event
- `calendar.addRouteToCalendar` - Add route to calendar
- `calendar.removeRouteFromCalendar` - Remove route from calendar

Settings:

- `settings.getPreferences` - Get user preferences
- `settings.updatePreferences` - Update user preferences

Admin:

- `admin.getUsers` - Get all users (admin only)
- `admin.mergeUsers` - Merge duplicate accounts (admin only)
- `admin.deleteUser` - Delete user (admin only)
- `admin.getMissedStops` - Get all missed stops (admin only)
- `admin.getRescheduleHistory` - Get reschedule history (admin only)

Notes:

- `notes.list` - Get route notes
- `notes.create` - Create route note
- `notes.update` - Update route note
- `notes.delete` - Delete route note

Scheduler Notes:

- `schedulerNotes.list` - Get scheduler sticky notes
- `schedulerNotes.create` - Create sticky note
- `schedulerNotes.update` - Update sticky note
- `schedulerNotes.delete` - Delete sticky note
- `schedulerNotes.updatePosition` - Update sticky note position/size

Google API Integration

OAuth Scopes

Required Scopes:

- `https://www.googleapis.com/auth/contacts` - Read/write Google Contacts
- `https://www.googleapis.com/auth/calendar.events` - Read/write Calendar events
- `https://www.googleapis.com/auth/userinfo.email` - User email
- `https://www.googleapis.com/auth/userinfo.profile` - User profile

Token Management

Token Storage:

- Access token stored in users.accessToken
- Refresh token stored in users.refreshToken
- Expiry timestamp in users.tokenExpiry

Token Refresh:

```
async function refreshGoogleToken(userId: number) {
  const user = await getUserById(userId);
  if (!user.refreshToken) throw new Error("No refresh token");

  const response = await fetch("https://oauth2.googleapis.com/token", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      client_id: ENV.googleClientId,
      client_secret: ENV.googleClientSecret,
      refresh_token: user.refreshToken,
      grant_type: "refresh_token",
    }),
  });

  const data = await response.json();
  const expiresIn = data.expires_in || 3600;
  const tokenExpiry = new Date(Date.now() + expiresIn * 1000);

  await updateUserTokens(userId, {
    accessToken: data.access_token,
    tokenExpiry,
  });

  return data.access_token;
}
```

Google People API

Fetch Contacts:

```
const response = await fetch(
  "https://people.googleapis.com/v1/people/me/connections?" +

"personFields=names,emailAddresses,addresses,phoneNumbers,photos,memberships",
  {
    headers: { Authorization: `Bearer ${accessToken}` },
  }
);
```

Update Contact:

```
const response = await fetch(
  `https://people.googleapis.com/v1/${resourceName}:updateContact?` +
  `updatePersonFields=addresses,phoneNumbers`,
  {
    method: "PATCH",
    headers: {
      Authorization: `Bearer ${accessToken}`,
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      resourceName,
      etag,
      addresses: [{ streetAddress: newAddress }],
      phoneNumbers: updatedPhones,
    }),
  }
);
```

Google Calendar API

Create Event:

```
const response = await fetch(
  `https://www.googleapis.com/calendar/v3/calendars/${calendarId}/events`,
  {
    method: "POST",
    headers: {
      Authorization: `Bearer ${accessToken}`,
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      summary: `${stopType}: ${contactName}`,
      location: address,
      start: { dateTime: startTime },
      end: { dateTime: endTime },
      description: routeNotes,
    }),
  }
);
```

Update Event:

```
const response = await fetch(

`https://www.googleapis.com/calendar/v3/calendars/${calendarId}/events/${event
  {
    method: "PUT",
    headers: {
      Authorization: `Bearer ${accessToken}`,
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      summary: updatedTitle,
      location: updatedAddress,
      start: { dateTime: updatedStartTime },
      end: { dateTime: updatedEndTime },
    }),
  }
);
```

Google Maps APIs

Routes API (Optimization):

```
const response = await fetch(
  "https://routes.googleapis.com/directions/v2:computeRoutes",
  {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "X-Goog-Api-Key": ENV.googleMapsApiKey,
      "X-Goog-FieldMask":
"routes.duration,routes.distanceMeters,routes.polyline,routes.optimizedInterme
    },
    body: JSON.stringify({
      origin: { address: startingPoint },
      destination: { address: startingPoint },
      intermediates: waypoints.map(w => ({ address: w.address })),
      travelMode: "DRIVE",
      optimizeWaypointOrder: true,
    }),
  }
);
```

Geocoding API:

```
const response = await fetch(
  `https://maps.googleapis.com/maps/api/geocode/json?` +
  `address=${encodeURIComponent(address)}&key=${ENV.googleMapsApiKey}`
);
const { lat, lng } = response.results[0].geometry.location;
```

Deployment

Current Production Deployment

Platform: Manus Cloud (Azure-based)
URL: https://routieroo.manus.space
Environment: Production
Database: TiDB (MySQL-compatible)
File Storage: S3-compatible storage

Environment Variables:

- `DATABASE_URL` - MySQL connection string
- `JWT_SECRET` - Session cookie signing secret
- `GOOGLE_CLIENT_ID` - Google OAuth client ID
- `GOOGLE_CLIENT_SECRET` - Google OAuth client secret
- `GOOGLE_MAPS_API_KEY` - Google Maps API key
- `VITE_APP_TITLE` - App title (Routie Roo)
- `VITE_APP_LOGO` - Logo URL
- `PUBLIC_URL` - Public domain for OAuth redirects

Alternative Railway Deployment

Setup:

1. Initialize Git repository
2. Push to GitHub
3. Connect Railway to GitHub repo
4. Configure environment variables
5. Deploy

Documentation:

- `SETUP_INSTRUCTIONS.md` - Step-by-step deployment guide
- `README.md` - Project overview and quick start

Testing

Test Coverage

Total Tests: 187 passing

Test Categories:

- Authentication (logout, session validation)
- Contact management (sync, update, toggle active)
- Route creation (validation, optimization)
- Route execution (status updates, reordering)
- Calendar integration (event creation, sync)
- Mobile optimization (responsive design, touch targets)
- Admin features (user management, reports)

Test Framework:

- Vitest for unit and integration tests
- React Testing Library for component tests
- tRPC test utilities for procedure tests

Example Test:

```
describe("auth.logout", () => {
  it("clears the session cookie and reports success", async () => {
    const { ctx, clearedCookies } = createAuthContext();
    const caller = appRouter.createCaller(ctx);

    const result = await caller.auth.logout();

    expect(result).toEqual({ success: true });
    expect(clearedCookies).toHaveLength(1);
    expect(clearedCookies[0]?.name).toBe(COOKIE_NAME);
  });
});
```

Performance Considerations

Database Optimization

Indexes:

- Primary keys on all tables (auto-increment)

- Foreign key indexes for joins
- Index on users.openId for auth lookups
- Index on routes.userId for user queries
- Index on cached_contacts.userId for contact lists
- Index on route_waypoints.routeId for waypoint queries

Query Optimization:

- Use Drizzle ORM for type-safe queries
- Batch queries with Promise.all where possible
- Limit result sets with .limit()
- Use .where() filters to reduce data transfer
- Avoid N+1 queries with proper joins

Frontend Optimization

Code Splitting:

- Next.js automatic code splitting
- Dynamic imports for large components
- Lazy loading for routes

Caching:

- tRPC React Query caching
- Stale-while-revalidate strategy
- Optimistic updates for instant feedback
- Cache invalidation on mutations

Asset Optimization:

- Image optimization with Next.js Image component
- Lazy loading images below fold
- WebP format for photos
- CDN for static assets

API Rate Limits

Google APIs:

- People API: 600 requests per minute per user
- Calendar API: 500 requests per minute per user
- Maps Routes API: 500 requests per day (free tier)
- Geocoding API: 50 requests per second

Mitigation Strategies:

- Cache contact data locally
- Batch calendar operations
- Debounce address autocomplete
- Rate limit sync operations

Security Considerations

Authentication

OAuth Security:

- HTTPS-only OAuth redirects
- State parameter for CSRF protection
- httpOnly session cookies
- Secure cookie flag in production
- SameSite=None for cross-origin requests

Token Security:

- Refresh tokens stored encrypted in database
- Access tokens expire after 1 hour
- Automatic token refresh before expiry
- Token revocation on logout

Authorization

Role-Based Access Control:

- Admin role for user management
- User role for normal operations
- Procedure-level authorization checks
- ctx.user validation in protected procedures

Data Isolation:

- All queries filter by userId
- Routes only accessible by owner or via shareToken
- Shared routes read-only for drivers
- Admin procedures check role

Input Validation

Zod Schemas:

- Runtime validation on all inputs
- Type-safe validation with TypeScript
- Custom error messages
- Sanitization of user input

Example:

```
const createRouteInput = z.object({
  name: z.string().min(1).max(255),
  notes: z.string().optional(),
  contactIds: z.array(z.number()).min(1),
  startingPoint: z.string().min(1),
  folderId: z.number().optional(),
  scheduledDate: z.string().optional(),
});
```

XSS Prevention

React Escaping:

- React automatically escapes JSX
- Use dangerouslySetInnerHTML only for trusted content
- Sanitize markdown with library

Content Security Policy:

- Restrict script sources
- Disable inline scripts
- Allow only trusted domains

Known Limitations

Google OAuth

Unverified App:

- Shows warning screen during login
- Limited to 100 test users
- Requires verification for production

Verification Requirements:

- Privacy Policy URL
- Terms of Service URL
- App homepage URL
- Video demo of app
- Security assessment
- 4-6 weeks review time

API Limitations

Google Maps Routes API:

- 500 requests per day (free tier)
- Upgrade to paid tier for production

Email Reminders:

- Requires manual cron job setup
- Not automated in current deployment

Missing Features

Export Formats:

- GPX export not implemented
- KML export not implemented

Address Management:

- No bulk address validation tool
- No address history tracking
- No address confidence scoring

Future Enhancements

Short-Term (1-3 months)

Google OAuth Verification:

- Create Privacy Policy and Terms of Service
- Record video demo
- Submit for verification
- Remove 100 user limit

Email Reminders:

- Implement automated cron job
- Send reminders for important dates
- Configurable reminder intervals

Export Formats:

- GPX export for GPS devices
- KML export for Google Earth

Medium-Term (3-6 months)

Address Management:

- Bulk address validation tool
- Address history tracking
- Address confidence scoring
- Automatic address correction suggestions

Analytics:

- Route completion metrics
- Average route duration
- Most visited contacts
- Stop type distribution

Team Features:

- Team accounts with multiple users
- Assign routes to team members
- Team-wide contact library
- Shared folders

Long-Term (6-12 months)

Mobile Apps:

- Native iOS app

- Native Android app
- Offline mode
- Push notifications

Advanced Routing:

- Multi-day route planning
- Time windows for stops
- Vehicle capacity constraints
- Driver preferences

Integrations:

- Zapier integration
- Slack notifications
- SMS reminders (Twilio)
- Voice navigation (Waze, Apple Maps)

Troubleshooting

Common Issues

OAuth Redirect URI Error:

- Ensure PUBLIC_URL environment variable is set
- Check Google Cloud Console authorized redirect URIs
- Must match exactly (including https://)

Map Markers Not Displaying:

- Check waypoints have latitude/longitude
- Verify Google Maps API key is valid
- Check browser console for API errors
- Ensure map resize trigger is called

Contact Sync Failing:

- Check OAuth token is valid
- Verify Google People API is enabled
- Check user granted contacts permission
- Look for rate limit errors

Calendar Events Not Creating:

- Check OAuth token has calendar scope
- Verify Google Calendar API is enabled
- Check user granted calendar permission
- Ensure calendar ID is valid

Route Creation Error:

- Verify all contacts have addresses
- Check Google Maps Routes API quota
- Ensure API key has Routes API enabled
- Validate waypoint count (max 25)

Debug Tools

Server Logs:

```
# View server logs
docker logs -f routieroo-server

# Filter for errors
docker logs routieroo-server | grep ERROR
```

Database Queries:

```sql
-- Check user tokens
SELECT id, email, tokenExpiry FROM users WHERE id = ?;

-- Check route waypoints
SELECT * FROM route_waypoints WHERE routeId = ?;

-- Check calendar events
SELECT * FROM routes WHERE googleCalendarId IS NOT NULL;
```

Browser DevTools:

- Network tab: Check API requests/responses
- Console tab: Check for JavaScript errors
- Application tab: Check cookies and localStorage

Maintenance

Regular Tasks

Daily:

- Monitor error logs
- Check API rate limits
- Verify OAuth tokens refreshing

Weekly:

- Review user feedback
- Check database performance
- Update dependencies

Monthly:

- Review Google API usage
- Optimize slow queries

- Clean up old data

Database Maintenance

Backup:

```
# Backup database
mysqldump -u user -p database > backup.sql

# Restore database
mysql -u user -p database < backup.sql
```

Cleanup:

```
-- Archive old routes
UPDATE routes SET archivedAt = NOW()
WHERE completedAt < DATE_SUB(NOW(), INTERVAL 90 DAY);

-- Delete old reschedule history
DELETE FROM reschedule_history
WHERE createdAt < DATE_SUB(NOW(), INTERVAL 1 YEAR);
```

Monitoring

Key Metrics:

- API response times
- Database query times
- Error rates
- User sign-ups
- Route creations
- Calendar syncs

Alerts:

- Server downtime

- Database connection failures
- API quota exceeded
- OAuth token refresh failures

Support & Documentation

User Documentation

- `USER_GUIDE.md` - End-user guide with screenshots
- In-app tooltips and help text
- Video tutorials (planned)

Developer Documentation

- `README.md` - Project overview and quick start
- `SETUP_INSTRUCTIONS.md` - Deployment guide
- `TECHNICAL_SUMMARY.md` - This document
- `PROJECT_HISTORY.md` - Complete checkpoint history
- Inline code comments

Support Channels

- Email: support@routieroo.com (planned)
- In-app feedback form (planned)
- GitHub issues (planned)

End of Technical Summary

Last Updated: December 1, 2024
Version: 40d03018
Author: Manus AI Agent