TripBi Implementation Guide for Cursor

Optimized for AI-Assisted Development with Cursor Pro

🎯 Purpose

This guide helps you build TripBi efficiently using Cursor's AI capabilities. It includes:

- Project structure optimized for Cursor's codebase understanding
- Effective prompts for each implementation phase
- Best practices for AI pair programming
- Common patterns to establish early

📁 Recommended Project Structure

Set up your repository with this structure for optimal Cursor AI understanding:

```
tripbi-app/
├── .cursorrules                # Cursor-specific instructions (create this!)
├── .env.local                  # Environment variables (git-ignored)
├── .env.example                # Template for env vars
├── .gitignore
├── README.md
├── package.json
├── vite.config.ts
├── tsconfig.json
├── tailwind.config.js
├── index.html
│
├── public/                     # Static assets
│   ├── favicon.ico
│   └── images/
│
├── src/
│   ├── main.tsx                # Entry point
│   ├── App.tsx                 # Root component
│   │
│   ├── components/             # Reusable UI components
│   │   ├── ui/                 # Basic UI elements
│   │   │   ├── Button.tsx
│   │   │   ├── Input.tsx
│   │   │   ├── Modal.tsx
│   │   │   └── Card.tsx
│   │   ├── layout/             # Layout components
│   │   │   ├── Header.tsx
│   │   │   ├── Sidebar.tsx
│   │   │   └── Footer.tsx
│   │   └── shared/             # Shared complex components
│   │       ├── LoadingSpinner.tsx
│   │       └── ErrorBoundary.tsx
│   │
│   ├── features/               # Feature-based organization
│   │   ├── auth/
│   │   │   ├── components/
│   │   │   ├── hooks/
│   │   │   ├── types.ts
│   │   │   └── utils.ts
│   │   ├── trips/
│   │   │   ├── components/
│   │   │   ├── hooks/
```

```
│   │   │   ├── types.ts
│   │   │   └── utils.ts
│   │   ├── proposals/
│   │   ├── voting/
│   │   ├── bookings/
│   │   ├── timeline/
│   │   └── splitbi/
│   │
│   ├── lib/                    # Core utilities and configs
│   │   ├── firebase.ts         # Firebase initialization
│   │   ├── api.ts              # API client setup
│   │   └── constants.ts        # App-wide constants
│   │
│   ├── hooks/                  # Shared custom hooks
│   │   ├── useAuth.ts
│   │   ├── useFirestore.ts
│   │   └── useToast.ts
│   │
│   ├── contexts/               # React contexts
│   │   ├── AuthContext.tsx
│   │   └── TripContext.tsx
│   │
│   ├── types/                  # TypeScript type definitions
│   │   ├── trip.ts
│   │   ├── proposal.ts
│   │   ├── booking.ts
│   │   └── user.ts
│   │
│   ├── utils/                  # Helper functions
│   │   ├── dateFormat.ts
│   │   ├── validation.ts
│   │   └── storage.ts
│   │
│   ├── pages/                  # Page components (routes)
│   │   ├── Home.tsx
│   │   ├── Login.tsx
│   │   ├── TripDashboard.tsx
│   │   ├── TripDetails.tsx
│   │   └── Timeline.tsx
│   │
│   └── styles/                 # Global styles
│       ├── globals.css
│       └── tailwind.css
│
```

```
├── functions/                # Firebase Cloud Functions
│   ├── src/
│   │   ├── index.ts
│   │   ├── splitbi/          # Splitbi integration functions
│   │   ├── email/            # Email notification functions
│   │   └── utils/
│   ├── package.json
│   └── tsconfig.json
│
└── docs/                     # Documentation
    ├── API.md
    ├── DATABASE.md
    └── DEPLOYMENT.md
```

🤖 Creating .cursorrules File

Create a `.cursorrules` file in your project root. This guides Cursor's AI behavior:

```markdown
```

TripBi Project Rules for Cursor AI

Project Context
This is TripBi, a group trip planning platform built with React, TypeScript, and Firebase.
Users coordinate travel plans through proposals, voting, and booking tracking.

Tech Stack
- Frontend: React 19, TypeScript 5.8, Vite 6.2, Tailwind CSS
- Backend: Firebase (Firestore, Auth, Functions, Storage)
- Mobile: Capacitor for app store deployment

Code Style & Patterns

TypeScript
- Use strict TypeScript configuration
- Prefer interfaces over types for object shapes
- Export types alongside components
- Use enums for fixed value sets (ProposalStatus, VoteType)

React Patterns
- Use functional components with hooks
- Prefer named exports for components
- Use custom hooks for Firebase interactions
- Co-locate component files: Component.tsx, Component.test.tsx
- Use React.FC sparingly (prefer explicit prop typing)

Firebase Patterns
- Always unsubscribe from Firestore listeners in useEffect cleanup
- Use Firebase Timestamp for dates (not JavaScript Date)
- Batch writes when updating multiple documents
- Use subcollections for one-to-many relationships

State Management
- Use Context API for global state (auth, current trip)
- Use local state for component-specific data
- Consider Zustand if Context becomes unwieldy (Phase 2)

File Organization
- Feature-based structure: features/[feature]/{components,hooks,types,utils}
- Shared components in components/{ui,layout,shared}
- Types in types/ folder with feature-specific types in feature folders

Styling

- Use Tailwind utility classes
- Create reusable component variants with cva (class-variance-authority)
- Avoid inline styles unless absolutely necessary
- Use Tailwind's responsive prefixes (sm:, md:, lg:)

Error Handling
- Always wrap async Firebase calls in try-catch
- Display user-friendly error messages (use Toast component)
- Log detailed errors to console in development
- Use Sentry for production error tracking (Phase 2)

Testing (Future)
- Write tests for critical paths (auth, voting, timeline generation)
- Use React Testing Library
- Mock Firebase calls in tests

Naming Conventions
- Components: PascalCase (TripCard.tsx)
- Hooks: camelCase with "use" prefix (useTrip.ts)
- Utils: camelCase (formatDate.ts)
- Types: PascalCase (TripData, ProposalStatus)
- Constants: SCREAMING_SNAKE_CASE (MAX_TRIP_MEMBERS)

Import Order
1. External libraries (react, firebase, etc.)
2. Internal utilities and configs
3. Components
4. Types
5. Styles

Comments
- Use JSDoc for public functions/components
- Explain "why" not "what" in inline comments
- Document complex Firebase queries
- Add TODO comments with context

Security
- Never commit .env files
- Use environment variables for API keys
- Validate user input before Firestore writes
- Implement proper Firestore security rules
- Sanitize user-generated content before display

Performance

- Use React.memo for expensive renders
- Implement pagination for large lists (proposals, bookings)
- Lazy load routes with React.lazy
- Optimize images (WebP format, lazy loading)

Accessibility
- Use semantic HTML elements
- Add ARIA labels where needed
- Ensure keyboard navigation works
- Test with screen readers (future)

Integration with Splitbi
- Use Cloud Functions for API calls (never client-side API keys)
- Handle Splitbi API failures gracefully (show error, continue without integration)
- Store minimal Splitbi data (just groupId reference)

When Making Suggestions
- Prioritize Firebase best practices (I'm familiar with this stack from Splitbi)
- Reference React 19 patterns (I'm using latest version)
- Suggest Tailwind solutions (I'm using this for styling)
- Consider mobile implications (will deploy to app stores)
- Keep code similar to Splitbi patterns when possible (for consistency)

💬 Effective Cursor Prompts by Phase

Phase 1: Project Setup

Initial Scaffolding:

Create a new React + TypeScript + Vite project for TripBi with:
- Tailwind CSS configured
- Firebase SDK installed (firestore, auth, functions, storage)
- React Router for navigation
- Basic folder structure following features/ pattern
- Environment variable setup (.env.example with Firebase config placeholders)

Firebase Configuration:

Set up Firebase initialization in lib/firebase.ts with:
- Firestore, Auth, Storage, and Functions imports
- Environment variables from .env.local
- Export configured instances
- Include TypeScript types

Phase 2: Authentication

Auth Context:

Create an AuthContext in contexts/AuthContext.tsx that:
- Wraps Firebase Auth
- Provides current user state
- Includes login, logout, signup functions
- Handles loading states
- Uses TypeScript for user type

Login Component:

Build a Login component in features/auth/components/Login.tsx:
- Email/password form with validation
- Google Sign-in button
- Error message display
- Loading states during auth
- Redirect to home after successful login
- Uses Tailwind for styling

Protected Routes:

Create a ProtectedRoute component that:
- Checks if user is authenticated
- Redirects to login if not
- Shows loading spinner while checking auth state
- Works with React Router

Phase 3: Trip Management

Trip Type Definition:

Define a Trip type in types/trip.ts with:
- Trip data structure (id, name, destination, dates, members, etc.)
- Member type (userId, role, joinedAt)
- Status enum (planning, active, completed)
- TypeScript interfaces
- Include JSDoc comments

Create Trip Flow:

Build a CreateTrip component with:
- Form fields: name, destination, start date, end date
- Date picker (use react-datepicker library)
- Validation (required fields, end after start)
- Firebase create function (add to trips collection)
- Add creator as first member with admin role
- Success/error handling
- Tailwind styling

Trip Dashboard:

Create TripDashboard page showing:
- List of user's trips (query Firestore where member)
- Trip cards with name, destination, date range, member count
- "Create New Trip" button
- Empty state if no trips
- Loading skeleton while fetching
- Real-time updates (Firestore listener)

Phase 4: Proposals & Voting

Proposal Type:

Define Proposal interface in types/proposal.ts:
- ProposalStatus enum (proposed, discussing, decided)
- ProposalCategory enum (flights, hotels, activities, tasks)
- Vote type (userId, vote, timestamp)
- Comment type (userId, text, timestamp)
- Full proposal structure with all fields

Create Proposal Component:

Build CreateProposal component with:
- Category dropdown (flights, hotels, activities, tasks)
- Dynamic form fields based on category
- Rich text description field
- Option to set voting deadline
- Submit to Firestore proposals collection
- Validate required fields
- Success notification

Voting Component:

Create VotingCard component that:
- Displays proposal details
- Shows current vote count (yes/no/abstain)
- Vote buttons for current user
- Real-time vote updates
- Disable voting if deadline passed
- Show who voted what (transparency)
- Update Firestore votes array

Proposal Status Management:

Create a function to transition proposal status:
- Check vote threshold (majority yes = move to decided)
- Update proposal status in Firestore
- Trigger timeline regeneration
- Send notifications to members
- Handle edge cases (ties, low participation)

Phase 5: Booking Tracker

Booking Type:

Define Booking interface in types/booking.ts:
- Links to trip and proposal
- Booking status (confirmed, pending)
- Optional confirmation number
- Optional proof URL (Firebase Storage)
- Notes field
- Booked for count (if booking for multiple people)

My Bookings Dashboard:

Create MyBookings component showing:
- List of user's confirmed bookings
- Pending bookings with "Book Now" links
- Group by trip
- Show confirmation numbers and proof
- Filter/sort options
- Empty state if no bookings

Group Booking Status:

Build GroupBookingStatus component that:
- Shows all decided proposals for a trip
- Display booking status per member (confirmed/pending)
- Progress indicator (X/Y members booked)
- "Send Reminder" button for pending members
- Color-coded status (green=confirmed, yellow=pending)
- Real-time updates

Mark as Booked Flow:

Create ConfirmBooking modal with:
- "I've booked this" checkbox
- Optional confirmation number input
- Optional file upload (screenshot/PDF to Firebase Storage)
- Notes textarea
- Submit to create booking document in Firestore
- Update group booking status aggregation

Phase 6: Timeline

Timeline Generation:

Create a function in features/timeline/utils.ts that:
- Queries all decided proposals for a trip
- Sorts by date and time
- Groups by day
- Formats into timeline structure
- Handles all-day events vs timed events
- Returns TimelineData type

Timeline View Component:

Build Timeline component displaying:
- Day-by-day breakdown
- Time-sorted events per day
- Color coding by category (flights=blue, hotels=purple, etc.)
- Icons per category
- Responsive design (mobile-friendly)
- Print-friendly CSS

Export Timeline:

Add timeline export functionality:
- Generate PDF using jsPDF or similar
- Include trip name, dates, all timeline items
- Format for readability
- Download to device
- Option to share via link (read-only Firebase doc)

Phase 7: Splitbi Integration

Cloud Function for Create Group:

```
Create a Cloud Function in functions/src/splitbi/createGroup.ts:
- Accepts trip data (name, members, creator)
- Calls Splitbi API: POST /api/groups/create
- Uses API key from environment
- Returns Splitbi groupId
- Error handling (log and throw)
- Update trip document with splitbiGroupId
```

Client-Side Integration:

```
Create useSplitbi hook in features/splitbi/hooks/useSplitbi.ts:
- Function to enable Splitbi (calls Cloud Function)
- Function to fetch expense summary (GET /api/groups/{id}/summary)
- Loading and error states
- Returns data and functions
```

Splitbi Card Component:

```
Build SplitbiCard for trip dashboard:
- Shows expense summary (total, member balances)
- "Enable Splitbi" button if not connected
- "View in Splitbi" link if connected
- Refresh button to fetch latest data
- Error message if API fails
- Graceful fallback if Splitbi unavailable
```

🔧 Cursor Pro Tips

Use @codebase Reference

When asking Cursor for help, reference your codebase:

```
@codebase How do I handle Firestore real-time listeners in this project?
```

Reuse Patterns from Splitbi

If you have Splitbi open:

> @Splitbi/src/hooks/useAuth.ts Create a similar auth hook for TripBi

Generate Multiple Files at Once

> Create the following files for trip management:
> 1. types/trip.ts - Trip and Member interfaces
> 2. features/trips/components/TripCard.tsx - Card component
> 3. features/trips/hooks/useTrips.ts - Custom hook for Firestore
> 4. features/trips/utils/tripHelpers.ts - Helper functions

Ask for Improvements

> Review this component and suggest improvements for:
> - TypeScript safety
> - Performance (unnecessary re-renders)
> - Accessibility (ARIA labels)
> - Error handling

Batch Related Changes

> Update all components that use the old User type to use the new UserProfile type

---

🐛 Common Issues & Solutions

Issue: Firestore Listener Memory Leaks

Cursor Prompt:

> Review all Firestore listeners in the codebase and ensure they have proper cleanup in useEffect

Issue: Type Errors with Firebase Timestamp

Cursor Prompt:

> Create a utility function to safely convert Firebase Timestamp to JavaScript Date throughout the app

Issue: Slow Re-renders

Cursor Prompt:

> Identify components that re-render unnecessarily and wrap them with React.memo where appropriate

Issue: Inconsistent Error Handling

Cursor Prompt:

> Create a centralized error handling utility and update all try-catch blocks to use it

---

📊 Cursor + Claude Workflow

Best Practice: Use both tools strategically

When to Use Cursor (Implementation)

- Writing components and hooks
- Refactoring code
- Generating boilerplate
- Finding and fixing bugs
- Writing tests

When to Use Claude (Architecture)

- Designing data models
- Choosing between approaches
- Planning complex features
- Debugging architectural issues
- Reviewing overall code structure

Example Workflow:

1. **Claude:** "How should I structure the voting system data model?"
2. **Implement:** Use Cursor to write components based on Claude's design
3. **Claude:** "Review this voting implementation for issues"
4. **Refine:** Use Cursor to fix issues Claude identified

🚀 Implementation Sequence

Follow this order for fastest progress:

Week 1: Foundation

1. Project setup (Vite + React + TypeScript + Tailwind)
2. Firebase configuration
3. Authentication system (login, signup, protected routes)
4. Basic routing structure

Cursor prompts to use:

- "Set up project with [stack] and generate folder structure"
- "Create auth context and login/signup components"

Week 2: Trip Management

1. Trip data model (types)
2. Create trip flow
3. Trip dashboard (list view)
4. Trip details page
5. Invite members functionality

Cursor prompts to use:

- "Generate Trip and Member TypeScript interfaces"
- "Build CreateTrip form with validation"
- "Create real-time trip list with Firestore listener"

Week 3: Proposals

1. Proposal data model
2. Create proposal form (dynamic by category)
3. Proposal list view
4. Proposal detail view
5. Status management

Cursor prompts to use:

- "Create Proposal types with category variants"
- "Build dynamic form that changes based on proposal category"

Week 4: Voting

1. Voting UI components
2. Vote submission logic
3. Real-time vote updates
4. Comments system
5. Status transitions (proposed → discussing → decided)

Cursor prompts to use:

- "Create VotingCard component with real-time updates"
- "Implement proposal status transition logic"

Week 5: Booking Tracker

1. Booking data model
2. "Mark as Booked" flow
3. File upload (confirmations)
4. My Bookings dashboard
5. Group Booking Status view

Cursor prompts to use:

- "Build ConfirmBooking modal with file upload to Firebase Storage"
- "Create dashboard showing booking status per member"

Week 6: Timeline

1. Timeline data structure
2. Timeline generation logic
3. Timeline UI component
4. Export to PDF
5. Shareable timeline

Cursor prompts to use:

- "Create function to generate timeline from decided proposals"
- "Build Timeline component with day-by-day layout"

Week 7: Splitbi Integration

1. Cloud Functions setup
2. Splitbi API wrapper
3. Create group function
4. Get summary function

5. UI integration

Cursor prompts to use:

- "Create Cloud Function to call external API with error handling"
- "Build React hook for Splitbi integration"

Week 8: Polish & Testing

1. Loading states everywhere
2. Error boundaries
3. Responsive design fixes
4. Performance optimization
5. Manual testing checklist

Cursor prompts to use:

- "Add loading skeletons to all data-fetching components"
- "Identify and fix mobile layout issues"

📱 Capacitor Setup (Mobile)

When ready to package for mobile:

Cursor Prompt:

```
Set up Capacitor for this React app:
1. Install Capacitor dependencies
2. Initialize Capacitor config
3. Add iOS and Android platforms
4. Configure app icons and splash screens
5. Update build scripts in package.json
```

🔒 Security Checklist

Before deploying, ask Cursor to review:

```
Review the codebase for security issues:
- Environment variables properly used (no hardcoded keys)
- User input sanitized before Firestore writes
- Firestore security rules defined (provide examples)
- Authentication properly checked on protected routes
- API keys not exposed in client code
```

📝 Documentation Generation

Use Cursor to help document as you build:

```
Generate JSDoc comments for all exported functions in [file]
```

```
Create a README for the features/trips folder explaining the architecture
```

🎓 Learning Resources

If Cursor suggests unfamiliar patterns, ask for explanation:

```
Explain how [pattern/concept] works and why it's used here
```

```
Show me best practices for [Firebase feature] in React
```

✅ Ready to Start

With this guide, you're set up to:

- Structure your project optimally for Cursor
- Use effective prompts for each feature

- Follow a logical build sequence
- Leverage both Cursor (implementation) and Claude (architecture)

Next: Complete prerequisites, then start implementation with Cursor!