T1-T4 Implementation Guide

Overview

This document describes the implementation of foundational features T1-T4 for the UPRISE_NEXT platform. These features establish core architectural patterns, geospatial capabilities, real-time communication, and testing infrastructure.

Implementation Summary

✅ T1: Web-Tier Contract Guard

Goal: Prevent direct database access from web application tier.

Components Implemented:

1. **Runtime Guard** (`apps/web/src/lib/web-tier-guard.ts`)
   - `isWebTier()` - Detects if code is running in web/browser context
   - `assertNotWebTier()` - Throws error if called from web tier
   - `guardPrismaClient()` - Prevents Prisma Client instantiation
   - `guardDatabaseAccess()` - Blocks any database operations
   - Module loader hooks to catch imports

2. **TypeScript Types** (`apps/web/src/lib/types/web-tier.d.ts`)
   - `ApiOnly<T>` - Type marker for API-tier only values
   - `WebSafe<T>` - Type marker for web-safe values
   - Prisma Client declared as `never` type in web context

3. **ESLint Rules** (`apps/web/.eslintrc.json`)
   - `no-restricted-imports` - Blocks @prisma/client imports
   - Pattern matching for database-related imports
   - Custom error messages explaining violations

4. **Next.js Middleware** (`apps/web/src/middleware.ts`)
   - Runs on every request
   - Checks for suspicious headers
   - Adds security headers (X-Web-Tier-Guard, X-DB-Access)

5. **Documentation** (`apps/web/WEB_TIER_BOUNDARY.md`)
   - Complete guide to web-tier boundaries
   - Examples of correct and incorrect patterns
   - Troubleshooting guide

Testing:
- See `apps/web/__tests__/web-tier-guard.test.ts`
- Verifies runtime protection works
- Tests error messages and types

✅ T2: Minimal API with PostGIS

Goal: Implement geospatial features for community location-based functionality.

Components Implemented:

1. **Database Migration** (`apps/api/prisma/migrations/20241113000000_init_postgis/`)
   - Enables PostGIS extension
   - Creates geography columns for communities
   - Adds GIST spatial indexes
   - All models (users, communities, tracks, events)

2. **Zod Validation Schemas** (`apps/api/src/communities/dto/community.dto.ts`)
   - `LatitudeSchema` - Validates -90 to 90
   - `LongitudeSchema` - Validates -180 to 180
   - `RadiusSchema` - Validates 10m to 50km
   - `CreateCommunityWithGeoSchema` - Full community creation
   - `FindNearbyCommunitiesSchema` - Spatial search params
   - `VerifyLocationSchema` - Location verification

3. **PostGIS Endpoints** (`apps/api/src/communities/`)

POST /api/communities
- Creates community with GPS coordinates
- Converts lat/lng to PostGIS geography point
- Stores geofence and radius

```
typescript
  {
    "name": "SF Music Scene",
    "slug": "sf-music-scene",
    "description": "Electronic music in San Francisco",
    "lat": 37.7749,
    "lng": -122.4194,
    "radius": 5000
  }
```

GET /api/communities/nearby
- Finds communities within radius using ST_DWithin
- Returns results sorted by distance
- Uses ST_Distance to calculate meters

```
GET /api/communities/nearby?lat=37.7749&lng=-122.4194&radius=5000&limit=20
```

POST /api/communities/:id/verify-location
- Checks if user is within community geofence
- Returns boolean + distance in meters

```
typescript
  {
    "lat": 37.7749,
```

```
    "lng": -122.4194
  }
```

1. **Health Check** (`apps/api/src/health/`)
   - `GET /api/health` - Overall system health
   - `GET /api/health/db` - Database connection check
   - `GET /api/health/postgis` - PostGIS verification
   - Tests ST_Distance calculation (SF to LA)
   - Verifies spatial_ref_sys count
   - Returns PostGIS version info

2. **Docker Setup** (`docker-compose.yml`)
   - PostgreSQL 15 with PostGIS 3.3
   - Exposed on port 5432
   - Health checks configured
   - Redis included for future use

PostGIS Queries Used:

```
-- Convert GPS to geography
ST_GeogFromText('POINT(lng lat)')

-- Find within radius (efficient)
ST_DWithin(geofence, point, radius)

-- Calculate distance in meters
ST_Distance(point1, point2)

-- Extract coordinates
ST_X(geofence::geometry), ST_Y(geofence::geometry)
```

Testing:
- See `apps/api/test/communities.test.ts`
- Tests all PostGIS endpoints
- Validates distance calculations
- Health check verification

✅ T3: Real-Time Socket.IO Functionality

Goal: Implement real-time communication for music communities.

Components Implemented:

1. **JWT Authentication Middleware** (`apps/socket/src/middleware/auth.ts`)
   - Validates JWT from handshake.auth or query params
   - Attaches user data to socket
   - Logs authentication attempts (success/failure)
   - `requireAuth()` helper for protected handlers

2. **Logger Utility** (`apps/socket/src/utils/logger.ts`)
   - Structured logging with context

- Log levels: INFO, WARN, ERROR, DEBUG
- Helper methods:
  - `logConnection()` - Socket connections
  - `logDisconnection()` - Socket disconnections
  - `logEvent()` - Event emissions
  - `logAuthSuccess/Failure()` - Authentication
  - `logJoinRoom/LeaveRoom()` - Room operations

3. **Community Namespace Handlers** (`apps/socket/src/namespaces/communities.ts`)

Dynamic Namespaces: `/community/:communityId`

Events Implemented:

- `join-community` - Explicit join with optional location
- `leave-community` - Explicit leave
- `community-message` - Send message to all members
- `track-share` - Share a track with community
- `track-reaction` - React to a track (emoji/like)
- `typing:start/stop` - Typing indicators
- `request-sync` - Request playback state

Auto-emitted Events:
- `user:joined` - When user connects to namespace
- `user:left` - When user disconnects
- `community:member-active` - After explicit join
- `community:member-inactive` - After explicit leave
- `community-message:new` - New message broadcast
- `track:shared` - Track shared notification
- `track:reaction` - Reaction notification

1. **Root Namespace Handlers** (`apps/socket/src/handlers/index.ts`)
   - `presence:update` - User status changes
   - `direct-message` - Send DM to another user
   - `notification-read` - Mark notification as read
   - `ping/pong` - Keep-alive health check

2. **Server Configuration** (`apps/socket/src/index.ts`)
   - CORS configuration
   - Ping/pong intervals (25s/60s)
   - Transport: websocket + polling fallback
   - Graceful shutdown with 10s timeout
   - Stats logging every minute
   - Error handling for uncaught exceptions

Usage Example:

```
// Client-side
import { io } from 'socket.io-client';

const socket = io('http://localhost:4001/community/sf-music', {
  auth: { token: jwtToken }
});

// Join community
socket.emit('join-community', {
  communityId: 'sf-music',
  location: { lat: 37.7749, lng: -122.4194 }
});

// Send message
socket.emit('community-message', {
  content: 'Hello everyone!',
  type: 'text'
});

// Listen for messages
socket.on('community-message:new', (message) => {
  console.log(`${message.username}: ${message.content}`);
});

// Leave community
socket.emit('leave-community', { communityId: 'sf-music' });
```

Testing:
- See `apps/socket/test/socket.test.ts`
- Tests JWT authentication flow
- Verifies event emission/reception
- Tests connection/disconnection logging

✅ T4: Smoke Tests

Goal: Create test suites to verify all T1-T3 functionality.

Components Implemented:

1. **Web-Tier Boundary Tests** (`apps/web/__tests__/web-tier-guard.test.ts`)
   - Tests runtime guard detection
   - Verifies errors are thrown on DB access
   - Validates TypeScript types
   - Tests API client is accessible
   - 8 test cases covering all guard functions

2. **API Integration Tests** (`apps/api/test/`)

communities.test.ts
- POST /api/communities endpoint
- GET /api/communities/nearby spatial queries
- POST /api/communities/:id/verify-location
- PostGIS extension verification
- Distance calculation accuracy
- 8+ test cases

health.test.ts
- GET /api/health overall status
- GET /api/health/postgis extension check
- GET /api/health/db connection
- Functionality test (SF to LA distance)
- 4+ test cases

1. **Socket.IO Tests** (`apps/socket/test/socket.test.ts`)
   - JWT authentication (valid/invalid/missing)
   - join-community event
   - community-message event
   - leave-community event
   - Connection logging verification
   - Disconnection handling
   - 6+ test cases

2. **Test Configuration**
   - Jest configured for all apps
   - `jsdom` for web tier (browser simulation)
   - `node` for API and Socket tiers
   - Coverage collection enabled
   - Watch mode support
   - Setup files for environment configuration

3. **Package.json Scripts**
   `json`
   `  {`
   `    "test": "jest",`
   `    "test:watch": "jest --watch",`
   `    "test:coverage": "jest --coverage"`
   `  }`

Running Tests:

```
# All tests
pnpm test

# Specific app
pnpm --filter web test
pnpm --filter api test
pnpm --filter socket test

# Watch mode
pnpm --filter web test:watch

# Coverage
pnpm --filter api test:coverage
```

File Structure

```
UPRISE_NEXT/
├── apps/
│   ├── web/
│   │   ├── src/
│   │   │   ├── lib/
│   │   │   │   ├── web-tier-guard.ts       # T1: Runtime guard
│   │   │   │   └── types/
│   │   │   │       └── web-tier.d.ts       # T1: TypeScript types
│   │   │   ├── middleware.ts                # T1: Next.js middleware
│   │   │   └── app/
│   │   ├── __tests__/
│   │   │   └── web-tier-guard.test.ts      # T4: Web boundary tests
│   │   ├── .eslintrc.json                   # T1: ESLint rules
│   │   ├── jest.config.js                   # T4: Jest config
│   │   └── WEB_TIER_BOUNDARY.md             # T1: Documentation
│   │
│   ├── api/
│   │   ├── prisma/
│   │   │   ├── schema.prisma                # T2: PostGIS schema
│   │   │   └── migrations/
│   │   │       └── 20241113000000_init_postgis/  # T2: Migration
│   │   ├── src/
│   │   │   ├── communities/
│   │   │   │   ├── dto/
│   │   │   │   │   └── community.dto.ts    # T2: Zod schemas
│   │   │   │   ├── communities.service.ts  # T2: PostGIS queries
│   │   │   │   └── communities.controller.ts # T2: Endpoints
│   │   │   ├── health/
│   │   │   │   ├── health.service.ts       # T2: Health checks
│   │   │   │   └── health.controller.ts
│   │   │   └── common/
│   │   │       └── decorators/
│   │   │           └── zod-query.decorator.ts
│   │   ├── test/
│   │   │   ├── communities.test.ts         # T4: API tests
│   │   │   └── health.test.ts              # T4: Health tests
│   │   └── jest.config.js                   # T4: Jest config
│   │
│   └── socket/
│       ├── src/
│       │   ├── middleware/
│       │   │   └── auth.ts                  # T3: JWT auth
│       │   ├── utils/
│       │   │   └── logger.ts                # T3: Structured logging
│       │   ├── namespaces/
│       │   │   └── communities.ts           # T3: Community handlers
│       │   ├── handlers/
│       │   │   └── index.ts                 # T3: Root handlers
│       │   └── index.ts                     # T3: Server setup
│       ├── test/
│       │   └── socket.test.ts               # T4: Socket tests
│       └── jest.config.js                   # T4: Jest config
│
├── docker-compose.yml                       # T2: PostGIS database
├── README.md                                # Updated documentation
└── T1-T4_IMPLEMENTATION_GUIDE.md            # This file
```

Database Setup

Using Docker (Recommended for Development)

```
# Start PostgreSQL with PostGIS
docker-compose up -d

# Verify it's running
docker ps

# Check logs
docker logs uprise_postgres

# Connect to database
docker exec -it uprise_postgres psql -U uprise -d uprise_dev
```

Manual PostgreSQL Setup

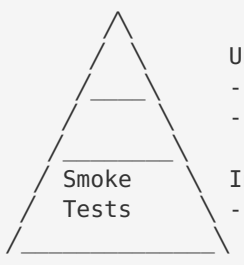
```
# Install PostgreSQL 15+
sudo apt-get install postgresql-15 postgresql-15-postgis-3

# Create database
sudo -u postgres createdb uprise_dev

# Enable PostGIS
sudo -u postgres psql -d uprise_dev -c "CREATE EXTENSION postgis;"
```

Run Migrations

```
cd apps/api

# Generate Prisma Client
pnpm prisma generate

# Run migrations
pnpm prisma migrate dev

# Verify PostGIS
curl http://localhost:4000/api/health/postgis
```

Testing Strategy

Test Pyramid




```
Unit Tests (Future)
- Pure functions
- Utility functions

Integration Tests (T4 - IMPLEMENTED)
- API endpoints
- Socket.IO connections
- Web-tier boundaries
```

Coverage Goals

- **Web Tier:** 80%+ coverage of guards and middleware
- **API Tier:** 70%+ coverage of controllers and services
- **Socket Tier:** 70%+ coverage of handlers and middleware

CI/CD Integration

Add to GitHub Actions / CI pipeline:

```yaml
- name: Run Tests
  run: pnpm test

- name: Check Coverage
  run: pnpm test:coverage

- name: Type Check
  run: pnpm typecheck

- name: Lint
  run: pnpm lint
```

Deployment Checklist

Pre-Deployment

- [ ] All tests passing (`pnpm test`)
- [ ] Type checking passes (`pnpm typecheck`)
- [ ] Linting passes (`pnpm lint`)
- [ ] Environment variables configured
- [ ] PostGIS extension enabled on production DB
- [ ] Migrations run on production DB

Environment Variables

API (.env):

```env
DATABASE_URL="postgresql://..."  # Must have PostGIS
JWT_SECRET="..."  # Strong secret
PORT=4000
NODE_ENV=production
CORS_ORIGIN="https://yourdomain.com"
```

Socket (.env):

```env
JWT_SECRET="..."  # Same as API
PORT=4001
NODE_ENV=production
CORS_ORIGIN="https://yourdomain.com"
```

Web (.env.local):

```
NEXT_PUBLIC_API_URL="https://api.yourdomain.com"
NEXT_PUBLIC_SOCKET_URL="https://socket.yourdomain.com"
```

Deployment Targets

- **Web:** Vercel (automatic from main branch)
- **API:** Fly.io or AWS App Runner
- **Socket:** Fly.io or AWS App Runner
- **Database:** Managed PostgreSQL with PostGIS (AWS RDS, DigitalOcean, etc.)

Common Issues & Solutions

Issue: PostGIS Extension Not Found

```
Error: PostGIS extension is not installed
```

Solution:

```
# Connect to database
psql $DATABASE_URL

# Enable extension
CREATE EXTENSION IF NOT EXISTS postgis;

# Verify
SELECT PostGIS_Version();
```

Issue: Web-Tier Boundary Violation

```
WEB-TIER BOUNDARY VIOLATION: Attempted to import @prisma/client
```

Solution:
- Remove any `@prisma/client` imports from `apps/web`
- Use `api.get/post/put/delete` from `@/lib/api` instead
- Check ESLint output for specific file/line

Issue: Socket Authentication Fails

```
Error: Invalid authentication token
```

Solution:
- Verify JWT_SECRET matches between API and Socket servers
- Check token format: `{ sub, email, username }`
- Ensure token is passed in `auth.token` or `query.token`

Issue: Tests Fail to Connect to Database

```
Error: Can't reach database server
```

Solution:

```
# Start database
docker-compose up -d

# Or set test database URL
export DATABASE_URL="postgresql://localhost:5432/uprise_test"

# Run migrations
cd apps/api && pnpm prisma migrate dev
```

Next Steps (T5-T8)

The T1-T4 implementation provides the foundation for:

- **T5:** Advanced real-time features (synchronized playback, live DJ sessions)
- **T6:** Media transcoding pipeline (FFmpeg worker, S3 integration)
- **T7:** Event system (geofenced events, RSVP, check-in)
- **T8:** Analytics and monitoring (Sentry, PostHog, metrics)

Git Commit History

```
git log --oneline --graph

* c1a51ae docs: Add comprehensive T1-T4 feature documentation
* 13da88b chore: Update web app layout and globals CSS
* 472bac7 feat(T4): Add Comprehensive Test Suites
* 842cdf2 feat(T3): Implement Real-Time Socket.IO with Community Namespaces
* 3682142 feat(T2): Implement PostGIS API with Geospatial Features
* 3d32638 feat(T1): Implement Web-Tier Contract Guard
```

Support & Documentation

- **Web-Tier Boundaries:** `apps/web/WEB_TIER_BOUNDARY.md`
- **Main README:** `README.md`
- **This Guide:** `T1-T4_IMPLEMENTATION_GUIDE.md`

For questions or issues, refer to the documentation or check the test files for usage examples.