Otantist
Developer Guide

Environment Setup & Onboarding

Version 2.0 — Updated February 2026
Includes Windows-specific troubleshooting

Table of Contents

1. Prerequisites

Required Software

Software	Version	Notes
Node.js	20.x LTS	Use nvm for version management
npm	10.x+	Comes with Node.js
Docker Desktop	Latest	Required for PostgreSQL, Redis, Mailhog
Git	Latest	Version control
VS Code	Latest	Recommended IDE

⚠️ **Windows Users**
Docker Desktop requires WSL 2. Make sure it's enabled in Docker Desktop settings under 'General > Use the WSL 2 based engine'. Some Docker caching issues are Windows-specific — see Troubleshooting section.

VS Code Extensions

Install these extensions for the best development experience:

- dbaeumer.vscode-eslint — ESLint integration
- esbenp.prettier-vscode — Code formatting
- bradlc.vscode-tailwindcss — Tailwind CSS IntelliSense
- prisma.prisma — Prisma schema syntax highlighting
- mikestead.dotenv — .env file support

2. Quick Start (Step-by-Step)

💡 **Before You Begin**
This guide assumes you're starting fresh. If you've already partially set up, you may need to reset (see Troubleshooting).

Step 1: Clone the Repository

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

Step 2: Install Dependencies

```
npm install
```

This installs dependencies for all workspaces (api, web, mobile, packages).

Step 3: Set Up Environment Variables

```
# Windows (PowerShell)
Copy-Item .env.example .env

# Mac/Linux
cp .env.example .env
```

The default .env values work for local development. No changes needed initially.

Step 4: Start Docker Services

```
npm run docker:up

# Verify containers are running:
docker ps
```

You should see these containers:

Container	Port	Purpose
otantist-postgres	5432	PostgreSQL database
otantist-redis	6379	Cache and sessions
otantist-mailhog	8025 (web), 1025 (SMTP)	Email testing

⚠️ **pgAdmin Optional**

> The docker-compose.yml includes pgAdmin, but it has known issues on Windows with email validation. Skip it — use Prisma Studio instead (see Step 5b).

Step 5: Initialize the Database

```
cd apps/api

# Generate Prisma client
npx prisma generate

# Run migrations
npx prisma migrate dev

# Seed with test data
npm run db:seed

# Return to root
cd ../..
```

Step 5b: Open Prisma Studio (Database GUI)

```
cd apps/api
npx prisma studio
```

Opens at http://localhost:5555 — this is your database admin interface.

> 💡 **Prisma Studio vs pgAdmin**
> Prisma Studio is purpose-built for your schema and requires no Docker container. It's the recommended way to browse and edit data during development.

Step 6: Start Development Servers

You need TWO terminal windows:

Terminal 1 — API Server:

```
npm run dev:api
```

Wait for: '🚀 Otantist API is running!' message

Terminal 2 — Web Server:

```
npm run dev:web
```

Step 7: Verify Everything Works

Service	URL	What You Should See
API Docs	http://localhost:3001/api/docs	Swagger UI with API endpoints
Web App	http://localhost:3000	Next.js welcome page
Mailhog	http://localhost:8025	Email inbox (empty initially)
Prisma Studio	http://localhost:5555	Database browser

💡 **Success!**
If all four URLs load correctly, your development environment is fully configured. You're ready to start coding!

3. Project Structure

```
otantist/
├── apps/
│   ├── api/                 # NestJS backend
│   │   ├── prisma/
│   │   │   ├── schema.prisma    # Database schema
│   │   │   ├── migrations/      # Database migrations
│   │   │   └── seed.ts          # Test data seeder
│   │   ├── src/
│   │   │   ├── auth/            # Authentication module
│   │   │   ├── users/           # User management
│   │   │   ├── preferences/     # User preferences
│   │   │   ├── messaging/       # 1:1 messaging
│   │   │   ├── moderation/      # Moderation tools
│   │   │   ├── parent-dashboard/
│   │   │   ├── prisma/          # Prisma service
│   │   │   ├── common/          # Shared utilities
│   │   │   ├── app.module.ts
│   │   │   └── main.ts
│   │   └── test/
│   │
│   ├── web/                 # Next.js web app
│   │   ├── app/                 # App router pages
│   │   ├── components/
│   │   ├── lib/
│   │   └── public/
│   │
│   └── mobile/              # React Native + Expo
│       ├── app/                 # Expo router
│       ├── components/
│       └── lib/
│
├── packages/
│   ├── shared/              # Shared types & constants
│   └── ui/                  # Shared UI components
│
├── scripts/                 # Utility scripts
├── docs/                    # Documentation
├── docker-compose.yml
├── package.json             # Root (workspaces)
└── .env.example
```

4. Development Workflow

Branch Naming

```
feature/OT-123-add-calm-mode
bugfix/OT-456-fix-message-queue
hotfix/OT-789-security-patch
chore/update-dependencies
```

Commit Messages

Follow Conventional Commits:

```
feat(messaging): add time boundary enforcement
fix(auth): resolve token refresh race condition
docs(api): update swagger documentation
chore(deps): upgrade nestjs to v10.3
```

Pull Request Process

1. Create feature branch from main
2. Make changes with meaningful commits
3. Run tests: npm test
4. Run linting: npm run lint
5. Create PR with description
6. Request review
7. Squash and merge

5. Database Management

Prisma Commands

```
cd apps/api

# Generate Prisma client after schema changes
npx prisma generate

# Create a new migration
npx prisma migrate dev --name add_user_preferences

# Apply migrations (production)
npx prisma migrate deploy

# Reset database (WARNING: deletes all data)
npx prisma migrate reset

# Open Prisma Studio (database GUI)
npx prisma studio
```

Schema Changes Workflow

8. Edit apps/api/prisma/schema.prisma
9. Run: npx prisma migrate dev --name descriptive_name
10. Prisma generates migration SQL and updates client
11. Commit both schema and migration files

Direct Database Access

If you need raw SQL access:

```
# Via Docker exec
docker exec -it otantist-postgres psql -U otantist -d otantist_dev

# Via connection string
psql postgresql://otantist:otantist_dev@localhost:5432/otantist_dev
```

6. API Development

Creating a New Module

```
cd apps/api

# Generate module scaffolding
npx nest generate module feature-name
npx nest generate controller feature-name
npx nest generate service feature-name
```

API Documentation

Swagger UI is available at http://localhost:3001/api/docs

Use decorators to document endpoints:

```
@ApiTags('messaging')
@ApiOperation({ summary: 'Send a message' })
@ApiResponse({ status: 201, description: 'Message sent' })
@ApiBearerAuth()
@Post()
async sendMessage(@Body() dto: SendMessageDto) {
  // ...
}
```

Authentication

Protected routes use JWT:

```
import { UseGuards } from '@nestjs/common';
import { JwtAuthGuard } from '../auth/guards/jwt-auth.guard';

@UseGuards(JwtAuthGuard)
@Get('me')
async getProfile(@Request() req) {
  return req.user;
}
```

7. Testing

Running Tests

```
# All tests
npm test

# API tests only
npm test -w @otantist/api

# Watch mode
npm test -- --watch

# Coverage report
npm test -- --coverage
```

Test Structure

```
apps/api/
├── src/
│   └── auth/
│       ├── auth.service.ts
│       └── auth.service.spec.ts   # Unit test
└── test/
    ├── auth.e2e-spec.ts           # E2E test
    └── jest-e2e.json
```

8. Coding Standards

TypeScript

- Strict mode enabled
- No 'any' types (use 'unknown' if needed)
- Explicit return types on functions
- Use interfaces over types when possible

Formatting

- Prettier handles formatting
- 2 space indentation
- Single quotes
- Trailing commas

Naming Conventions

Type	Convention	Example
Files	kebab-case	user-preferences.service.ts
Classes	PascalCase	UserPreferencesService
Functions/Methods	camelCase	getUserPreferences()
Constants	SCREAMING_SNAKE	MAX_MESSAGE_LENGTH
Interfaces	PascalCase with I prefix	IUserPreferences

Bilingual Content

All user-facing strings must support FR/EN:

```
// ❌ Bad
throw new BadRequestException('Invalid email');

// ✅ Good
throw new BadRequestException({
  code: 'INVALID_EMAIL',
  message_en: 'Invalid email address',
  message_fr: 'Adresse courriel invalide',
});
```

9. Troubleshooting

⚠️ **Windows-Specific Issues**
Docker on Windows has aggressive caching that can cause frustrating issues. The solutions below were tested on Windows 11 with Docker Desktop.

Docker Containers Won't Start

```
# Check container logs
docker logs otantist-postgres
docker logs otantist-redis

# Full restart
npm run docker:down
npm run docker:up
```

Port Already in Use

```
# Windows - find process using port 5432
netstat -ano | findstr :5432

# Kill the process (replace PID)
taskkill /PID <pid> /F

# Mac/Linux
lsof -i :5432
kill -9 <pid>
```

TypeScript Compilation Errors in seed.ts

If you see 'declared but never read' errors for variables like 'sam' or 'parent', these are warnings that won't prevent the server from starting. However, if the API doesn't start:

```
# Option 1: Prefix unused variables with underscore
const _sam = await prisma.user.create({...});

# Option 2: Use the variables (e.g., log them)
console.log('Created users:', alex.id, sam.id);
```

Database Migration Fails

```
cd apps/api

# Reset database (development only!)
npx prisma migrate reset

# Or resolve specific migration
npx prisma migrate resolve --rolled-back migration_name
```

Prisma Client Out of Sync

```
cd apps/api
npx prisma generate
```

pgAdmin Not Working (Windows)

pgAdmin has email validation issues on Windows with .local domains. Skip it entirely and use Prisma Studio instead:

```
cd apps/api
npx prisma studio    # Opens at http://localhost:5555
```

Prisma Studio provides the same functionality without Docker container issues.

Docker Caching Issues (Windows)

If Docker keeps using old configurations despite changes:

```
# Nuclear option - remove all volumes and rebuild
docker compose down -v
docker system prune -a --volumes
docker compose up -d --build
```

⚠ **Data Loss Warning**
The commands above will delete all Docker data including your database. Only use in development when you need a completely fresh start.

Node/npm Issues

```
# Clear everything and reinstall
npm run clean
Remove-Item package-lock.json   # Windows
rm package-lock.json            # Mac/Linux
npm install

# Wrong Node version? Use nvm
nvm install 20
nvm use 20
```

10. Quick Reference Commands

Daily Development

```
# Start all services (run in order)
npm run docker:up       # 1. Start Docker containers
npm run dev:api         # 2. Start API (Terminal 1)
npm run dev:web         # 3. Start Web (Terminal 2)
```

Database Commands

```
npm run db:migrate      # Run migrations
npm run db:studio       # Open Prisma Studio
npm run db:seed         # Seed test data
```

Testing & Quality

```
npm test                # Run all tests
npm run lint            # Lint all code
npm run build           # Build all apps
```

Docker Management

```
npm run docker:up       # Start containers
npm run docker:down     # Stop containers
npm run docker:logs     # View logs
docker ps               # List running containers
```

Environment URLs

Service	URL
API Server	http://localhost:3001
API Documentation	http://localhost:3001/api/docs
Web Application	http://localhost:3000
Mailhog (Email Testing)	http://localhost:8025
Prisma Studio (DB Admin)	http://localhost:5555

💡 **Need Help?**
Check the docs/ folder for additional documentation, review existing code for patterns, or create a GitHub issue for bugs.

Happy coding! 🚀