Complete Monorepo Setup Guide: Next.js + FastAPI

Architecture Overview

Why This Stack?

- **Turborepo**: Provides intelligent caching, parallel execution, and optimized builds for monorepos
- **pnpm**: Fast, disk-efficient package manager with excellent monorepo support via workspaces
- **Next.js 15**: Latest React framework with App Router, Server Components, and excellent DX
- **FastAPI**: Modern Python web framework with automatic API documentation and type hints
- **Poetry**: Dependency management and packaging for Python projects
- **Vitest**: Lightning-fast unit testing with native ESM support
- **React Testing Library**: Testing utilities focused on user behavior
- **Playwright**: Cross-browser E2E testing
- **Additional Tools**: Husky, lint-staged, commitlint for code quality

Project Structure

```
my-monorepo/
├── apps/
│   ├── web/                  # Next.js frontend
│   │   ├── src/
│   │   │   ├── app/          # App Router pages
│   │   │   ├── components/   # React components
│   │   │   ├── lib/          # Utilities and helpers
│   │   │   └── styles/       # Global styles
│   │   ├── public/
│   │   ├── tests/
│   │   │   ├── unit/         # Vitest + RTL tests
│   │   │   └── e2e/          # Playwright tests
│   │   ├── next.config.js
│   │   ├── package.json
│   │   ├── tsconfig.json
│   │   └── vitest.config.ts
│   │
│   └── api/                  # FastAPI backend
│       ├── app/
│       │   ├── api/
│       │   │   └── v1/
│       │   │       ├── endpoints/
│       │   │       └── router.py
│       │   ├── core/
│       │   │   ├── config.py
│       │   │   └── security.py
│       │   ├── models/
│       │   ├── schemas/
│       │   ├── services/
│       │   └── main.py
│       ├── tests/
│       ├── alembic/          # Database migrations
│       ├── pyproject.toml    # Poetry configuration
│       └── Dockerfile
│
├── packages/
│   ├── ui/                   # Shared UI components
│   │   ├── src/
│   │   ├── package.json
│   │   └── tsconfig.json
│   │
│   ├── config-typescript/    # Shared TS configs
│   │   ├── base.json
│   │   ├── nextjs.json
│   │   └── package.json
│   │
```

```
│   └── config-eslint/        # Shared ESLint configs
│       ├── index.js
│       ├── next.js
│       └── package.json
│
├── .github/
│   └── workflows/
│       ├── ci.yml
│       └── deploy.yml
│
├── .husky/
│   ├── pre-commit
│   └── commit-msg
│
├── turbo.json                # Turborepo configuration
├── pnpm-workspace.yaml       # pnpm workspace config
├── package.json              # Root package.json
├── .gitignore
├── .nvmrc                    # Node version
├── .env.example
├── docker-compose.yml
└── README.md
```

Step-by-Step Setup Guide

Step 1: Initialize the Monorepo

bash

```bash
# Create project directory
mkdir my-monorepo && cd my-monorepo

# Initialize git
git init

# Create .nvmrc for Node version consistency
echo "20.11.0" > .nvmrc

# Initialize pnpm (make sure pnpm is installed globally)
npm install -g pnpm
pnpm init

# Install Turborepo
pnpm add -D turbo
```

Step 2: Configure pnpm Workspaces

Create `pnpm-workspace.yaml`:

```yaml
packages:
  - 'apps/*'
  - 'packages/*'
```

Step 3: Setup Root Configuration

Update root `package.json`:

```json
{
  "name": "my-monorepo",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "turbo dev",
    "build": "turbo build",
    "test": "turbo test",
    "test:e2e": "turbo test:e2e",
    "lint": "turbo lint",
    "format": "prettier --write \"**/*.{ts,tsx,md,json}\"",
    "prepare": "husky install"
  },
  "devDependencies": {
    "@commitlint/cli": "^18.4.4",
    "@commitlint/config-conventional": "^18.4.4",
    "husky": "^8.0.3",
    "lint-staged": "^15.2.0",
    "prettier": "^3.1.1",
    "turbo": "^1.11.2"
  },
  "engines": {
    "node": ">=20.0.0",
    "pnpm": ">=8.0.0"
  },
  "packageManager": "pnpm@8.14.0"
}
```

Step 4: Configure Turborepo

Create `turbo.json`:

json

```json
{
  "$schema": "https://turbo.build/schema.json",
  "globalDependencies": ["**/.env.*local"],
  "pipeline": {
    "build": {
      "dependsOn": ["^build"],
      "outputs": [".next/**", "!.next/cache/**", "dist/**"],
      "env": ["NODE_ENV", "NEXT_PUBLIC_API_URL"]
    },
    "dev": {
      "cache": false,
      "persistent": true
    },
    "lint": {
      "dependsOn": ["^build"]
    },
    "test": {
      "dependsOn": ["build"],
      "outputs": ["coverage/**"],
      "cache": false
    },
    "test:e2e": {
      "dependsOn": ["build"],
      "cache": false
    },
    "type-check": {
      "dependsOn": ["^build"]
    }
  }
}
```

Step 5: Setup Next.js Frontend

bash

```bash
# Create Next.js app
mkdir -p apps/web
cd apps/web
pnpm init

# Install Next.js and dependencies
pnpm add next@latest react@latest react-dom@latest
pnpm add -D @types/node @types/react @types/react-dom typescript
pnpm add -D vitest @vitejs/plugin-react @testing-library/react @testing-library/user-event @testing-library/jest-dom
pnpm add -D playwright @playwright/test
pnpm add -D eslint eslint-config-next
pnpm add -D tailwindcss postcss autoprefixer
pnpm add axios swr zustand @tanstack/react-query
```

Create `apps/web/package.json`:

json

```json
{
  "name": "@monorepo/web",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint",
    "test": "vitest",
    "test:e2e": "playwright test",
    "type-check": "tsc --noEmit"
  }
}
```

Create `apps/web/next.config.js`:

javascript

```javascript
/** @type {import('next').NextConfig} */
const nextConfig = {
  reactStrictMode: true,
  swcMinify: true,
  experimental: {
    serverActions: true,
  },
  images: {
    domains: ['api.example.com'],
  },
  async rewrites() {
    return [
      {
        source: '/api/v1/:path*',
        destination: `${process.env.NEXT_PUBLIC_API_URL}/api/v1/:path*`,
      },
    ];
  },
};

module.exports = nextConfig;
```

Create `apps/web/vitest.config.ts`:

typescript

```typescript
import { defineConfig } from 'vitest/config';
import react from '@vitejs/plugin-react';
import path from 'path';

export default defineConfig({
  plugins: [react()],
  test: {
    environment: 'jsdom',
    globals: true,
    setupFiles: './tests/setup.ts',
    coverage: {
      reporter: ['text', 'json', 'html'],
      exclude: [
        'node_modules/',
        '.next/',
        'tests/',
      ],
    },
  },
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src'),
    },
  },
});
```

Create `apps/web/playwright.config.ts`:

typescript

```typescript
import { defineConfig, devices } from '@playwright/test';

export default defineConfig({
  testDir: './tests/e2e',
  fullyParallel: true,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: 'html',
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',
  },
  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },
    {
      name: 'firefox',
      use: { ...devices['Desktop Firefox'] },
    },
    {
      name: 'webkit',
      use: { ...devices['Desktop Safari'] },
    },
  ],
  webServer: {
    command: 'pnpm dev',
    url: 'http://localhost:3000',
    reuseExistingServer: !process.env.CI,
  },
});
```

Step 6: Setup FastAPI Backend

```bash
# Create API directory
mkdir -p apps/api
cd apps/api

# Initialize Poetry
poetry init --name api --dependency fastapi --dependency uvicorn --dependency python-dotenv --dependency httpx

# Add dev dependencies
poetry add --group dev pytest pytest-asyncio pytest-cov black ruff mypy
```

Create `apps/api/pyproject.toml`:

toml

```toml
[tool.poetry]
name = "api"
version = "0.1.0"
description = "FastAPI backend"
authors = ["Your Name <email@example.com>"]

[tool.poetry.dependencies]
python = "^3.11"
fastapi = "^0.109.0"
uvicorn = {extras = ["standard"], version = "^0.27.0"}
pydantic = "^2.5.0"
pydantic-settings = "^2.1.0"
python-dotenv = "^1.0.0"
httpx = "^0.26.0"
sqlalchemy = "^2.0.0"
alembic = "^1.13.0"
asyncpg = "^0.29.0"
redis = "^5.0.0"
python-jose = {extras = ["cryptography"], version = "^3.3.0"}
passlib = {extras = ["bcrypt"], version = "^1.7.4"}
python-multipart = "^0.0.6"

[tool.poetry.group.dev.dependencies]
pytest = "^7.4.0"
pytest-asyncio = "^0.21.0"
pytest-cov = "^4.1.0"
black = "^23.12.0"
ruff = "^0.1.9"
mypy = "^1.8.0"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"

[tool.black]
line-length = 88
target-version = ['py311']

[tool.ruff]
line-length = 88
select = ["E", "F", "I", "N", "UP", "S", "B", "A", "C4", "T20"]
ignore = ["E501"]

[tool.mypy]
```

```
python_version = "3.11"
strict = true
```

Create `apps/api/app/main.py`:

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.core.config import settings
from app.api.v1.router import api_router

app = FastAPI(
    title=settings.PROJECT_NAME,
    version=settings.VERSION,
    openapi_url=f"{settings.API_V1_STR}/openapi.json",
)

# Configure CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Include routers
app.include_router(api_router, prefix=settings.API_V1_STR)

@app.get("/health")
async def health_check():
    return {"status": "healthy"}
```

Step 7: Setup Shared Packages

Create shared UI package:

```bash
mkdir -p packages/ui
cd packages/ui
pnpm init
```

Create `packages/ui/package.json`:

json

```json
{
  "name": "@monorepo/ui",
  "version": "0.1.0",
  "main": "./dist/index.js",
  "module": "./dist/index.mjs",
  "types": "./dist/index.d.ts",
  "scripts": {
    "build": "tsup",
    "dev": "tsup --watch",
    "lint": "eslint .",
    "type-check": "tsc --noEmit"
  },
  "devDependencies": {
    "@types/react": "^18.2.0",
    "tsup": "^8.0.0",
    "typescript": "^5.3.0"
  },
  "peerDependencies": {
    "react": "^18.2.0"
  }
}
```

Step 8: Setup Git Hooks with Husky

bash

```bash
# Install Husky
pnpm add -D husky lint-staged @commitlint/cli @commitlint/config-conventional

# Initialize Husky
pnpm exec husky install

# Add pre-commit hook
pnpm exec husky add .husky/pre-commit "pnpm exec lint-staged"

# Add commit-msg hook
pnpm exec husky add .husky/commit-msg "pnpm exec commitlint --edit $1"
```

Create `.lintstagedrc.json`:

```json
{
  "*.{js,jsx,ts,tsx}": ["eslint --fix", "prettier --write"],
  "*.{json,md,yml,yaml}": ["prettier --write"],
  "*.py": ["black", "ruff --fix"]
}
```

Create `commitlint.config.js`:

```javascript
module.exports = {
  extends: ['@commitlint/config-conventional'],
  rules: {
    'type-enum': [
      2,
      'always',
      [
        'feat',
        'fix',
        'docs',
        'style',
        'refactor',
        'perf',
        'test',
        'chore',
        'revert',
        'ci',
        'build',
      ],
    ],
  },
};
```

Step 9: Environment Configuration

Create `.env.example`:

```bash
# Frontend
NEXT_PUBLIC_API_URL=http://localhost:8000
NEXT_PUBLIC_APP_NAME=My Monorepo App

# Backend
DATABASE_URL=postgresql://user:password@localhost:5432/myapp
REDIS_URL=redis://localhost:6379
SECRET_KEY=your-secret-key-here
BACKEND_CORS_ORIGINS=["http://localhost:3000"]
```

Step 10: Docker Configuration

Create `docker-compose.yml`:

yaml

```yaml
version: '3.8'

services:
  web:
    build:
      context: ./apps/web
      dockerfile: Dockerfile
    ports:
      - "3000:3000"
    environment:
      - NEXT_PUBLIC_API_URL=http://api:8000
    depends_on:
      - api

  api:
    build:
      context: ./apps/api
      dockerfile: Dockerfile
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://user:password@postgres:5432/myapp
      - REDIS_URL=redis://redis:6379
    depends_on:
      - postgres
      - redis

  postgres:
    image: postgres:15
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=myapp
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

volumes:
```

```
postgres_data:
```

Step 11: CI/CD with GitHub Actions

Create `.github/workflows/ci.yml`:

yaml

```yaml
name: CI

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main, develop]

jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: pnpm/action-setup@v2
        with:
          version: 8
      - uses: actions/setup-node@v3
        with:
          node-version: 20
          cache: 'pnpm'
      - run: pnpm install
      - run: pnpm lint

  test-frontend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: pnpm/action-setup@v2
        with:
          version: 8
      - uses: actions/setup-node@v3
        with:
          node-version: 20
          cache: 'pnpm'
      - run: pnpm install
      - run: pnpm test
      - run: pnpm exec playwright install
      - run: pnpm test:e2e

  test-backend:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v4
```

```
      with:
        python-version: '3.11'
    - uses: snok/install-poetry@v1
    - working-directory: ./apps/api
      run: |
        poetry install
        poetry run pytest --cov

  build:
    runs-on: ubuntu-latest
    needs: [lint, test-frontend, test-backend]
    steps:
      - uses: actions/checkout@v3
      - uses: pnpm/action-setup@v2
        with:
          version: 8
      - uses: actions/setup-node@v3
        with:
          node-version: 20
          cache: 'pnpm'
      - run: pnpm install
      - run: pnpm build
```

Development Workflow

Starting Development

```bash
# Install all dependencies
pnpm install

# Start development servers (frontend + backend)
pnpm dev

# Run tests
pnpm test

# Run E2E tests
pnpm test:e2e

# Build for production
pnpm build

# Lint and format
pnpm lint
pnpm format
```

Adding New Dependencies

```bash
# Add to specific workspace
pnpm add axios --filter @monorepo/web
pnpm add -D @types/node --filter @monorepo/web

# Add to root
pnpm add -D prettier -w

# For Python/FastAPI
cd apps/api && poetry add package-name
```

Best Practices Implemented

1. **Type Safety**

- TypeScript for frontend with strict mode
- Pydantic for backend data validation
- Shared types between frontend and backend (can generate from OpenAPI schema)

2. **Code Quality**

- ESLint + Prettier for JavaScript/TypeScript
- Black + Ruff for Python
- Pre-commit hooks with Husky
- Conventional commits with commitlint

3. **Testing Strategy**

- Unit tests with Vitest for fast feedback
- Integration tests with React Testing Library
- E2E tests with Playwright
- API tests with pytest

4. **Performance**

- Turborepo caching for faster builds
- pnpm for efficient dependency management
- Next.js optimizations (Image, Font, Script components)
- FastAPI async endpoints

5. **Developer Experience**

- Hot reload in development
- Shared configurations
- Consistent code style
- Automated workflows

6. **Security**

- Environment variables for secrets
- CORS configuration
- Input validation
- SQL injection prevention with SQLAlchemy ORM

Public APIs to Integrate

Here are some fun public APIs you can use:

1. **Weather**: OpenWeatherMap API
2. **News**: NewsAPI or The Guardian API
3. **Movies**: TMDB API
4. **Music**: Spotify Web API or Last.fm
5. **Games**: RAWG Video Games Database
6. **Jokes**: JokeAPI
7. **Facts**: Numbers API or Fun Facts API
8. **Images**: Unsplash API or Pexels API
9. **Quotes**: Quotable API
10. **Space**: NASA API or SpaceX API

Next Steps

1. Set up authentication (NextAuth.js + FastAPI JWT)
2. Add database migrations with Alembic
3. Implement caching strategy (Redis)
4. Set up monitoring (Sentry, DataDog)
5. Add API documentation (FastAPI auto-generates OpenAPI)
6. Implement rate limiting
7. Add WebSocket support for real-time features
8. Set up deployment (Vercel for frontend, Cloud Run/ECS for backend)

This setup provides a solid foundation for building scalable, maintainable applications with modern best practices!