CodeX - Code Collaboration Platform

CodeX - Code Collaboration Platform
Your collaborative coding space, reimagined. Start now, no sign-up required
codex.dulapahv.dev

CodeX is an online code collaboration platform that enables real-time coding, cursor sharing, live UI preview, and video communication with integrated Git support—no sign-up required.

✨ **Try now at codex.dulapahv.dev**

> This project is part of the course "COMPSCI4025P Level 4 Individual Project" at the University of Glasgow.

For detailed usage instructions and feature documentation, please see the **User Manual**.

Features

- **Real-time Collaboration** - Code together in real-time with cursor sharing, highlighting, and follow mode
- **Shared Terminal** - Execute code and see results together with over 80 supported languages
- **Live Preview** - Preview UI changes instantly with loaded libraries like Tailwind CSS, and more
- **GitHub Integrated** - Save your work and open files from your repositories
- **Shared Notepad** - Take notes together in real-time with rich text and markdown support
- **Video & Voice** - Communicate with your team using video and voice chat

Table of Contents

Project Structure

The project is organized as a monorepo using Turborepo:

```
CodeX
├── apps/                    # Application packages
│   ├── client/              # Frontend Next.js application
│   │   ├── public/          # Static assets
│   │   ├── src/             # Source code
│   │   │   ├── app/         # Next.js app router pages and API routes
│   │   │   ├── components/  # React components
│   │   │   ├── hooks/       # Custom React hooks
│   │   │   └── lib/         # Utility functions and services
│   │   └── tests/           # Frontend tests (Playwright)
│   └── server/              # Backend Socket.IO server
│       ├── src/             # Source code
│       │   ├── service/     # Backend services
│       │   └── utils/       # Utility functions
│       └── tests/           # Backend tests (Jest)
├── docs/                    # Documentation assets
├── packages/                # Shared packages
│   └── types/               # Shared TypeScript types and interfaces
├── scripts/                 # Build and maintenance scripts
├── package.json             # Root package.json
└── pnpm-workspace.yaml      # PNPM workspace configuration
```

Prerequisites

Before you begin, ensure you have the following installed:

- Node.js (v18 or higher)
- pnpm (v6 or higher)

If you don't have `pnpm` installed, you can install it globally:

```
npm install -g pnpm
```

Getting Started

After checking the prerequisites above, follow these steps to set up the project:

1. **Clone the repository**

   ```
   git clone https://github.com/dulapahv/CodeX.git
   cd CodeX
   ```

2. **Install dependencies**
   This will install all dependencies for the frontend and backend applications:

   ```
   pnpm install
   ```

   > Note: Git hooks will be automatically installed via Husky when running `pnpm install`

3. **Environment setup**
   Create `apps/client/.env` using the template from `apps/client/.env.example`:

   ```
   BETTERSTACK_API_KEY=
   SENTRY_AUTH_TOKEN=
   GITHUB_CLIENT_SECRET_PROD=
   GITHUB_CLIENT_SECRET_DEV=
   SENTRY_SUPPRESS_TURBOPACK_WARNING="1"
   TURBO_TEAM=
   TURBO_TOKEN=
   ```

   > Note: This is a personal project and the required API keys and secrets are not publicly shared. For local development, you'll need to set up your own credentials for GitHub OAuth, Sentry, etc.

Development

To start the development server for both the frontend and backend applications:

```
pnpm dev
```

You can also start them individually:

```
# Start only the client
pnpm --filter client dev

# Start only the server
pnpm --filter server dev
```

The application will be available at:

- Frontend: http://localhost:3000
- Backend: http://localhost:3001

Test

All test commands can be run from both the root directory and their respective workspaces.

Frontend Test

Both the frontend server and the backend server will start automatically. To run the frontend tests:

```
# In root directory or client workspace
pnpm test:client         # Run all frontend E2E tests
pnpm test:client:ui      # Run frontend tests with UI mode
pnpm test:client:debug   # Debug frontend tests
pnpm test:client:report  # View frontend test report

# Run in client workspace only
pnpm --filter client test:client
```

Backend Test

The backend server will start automatically. To run the backend tests:

```
# In root directory or server workspace
pnpm test:server           # Run backend tests against local server
pnpm test:server:remote    # Run backend tests against remote server
pnpm test:server:watch     # Run backend tests in watch mode (local server)

# Run in server workspace only
pnpm --filter server test:server
```

Build

This project is configured to build both the frontend and backend applications together with caching from Turborepo. To build the entire project:

```
pnpm build
```

However, you can also build them individually:

```
# Build frontend
pnpm build:client

# Build backend
pnpm build:server
```

The build artifacts of the frontend will be available in the `apps/client/.next` directory, and the backend will be available in the `apps/server/dist` directory.

Deployment

The project is configured for automatic deployment through Deploy Hooks which trigger after the GitHub Actions CI/CD pipeline completes successfully:

- Frontend (client): Automatically deploys to Vercel
- Backend (server): Automatically deploys to Render

Scripts

These are the available scripts in the project:

```
# Development
pnpm dev              # Start all applications in development mode
pnpm build            # Build all packages
pnpm build:client     # Build frontend
pnpm build:server     # Build backend
pnpm clean            # Clean all builds, caches, test results, and node_modules

# Testing
pnpm test:client          # Run frontend E2E tests (Playwright)
pnpm test:client:ui       # Run frontend tests with UI mode
pnpm test:client:debug    # Debug frontend tests
pnpm test:client:report   # View frontend test report
pnpm test:server          # Run backend tests against local server
pnpm test:server:remote   # Run backend tests against remote server
pnpm test:server:watch    # Run backend tests in watch mode (local server)

# Linting and Formatting
pnpm lint:check       # Run ESLint checks (frontend only)
pnpm lint:fix         # Fix ESLint issues (frontend only)
pnpm format:check     # Check formatting
pnpm format:fix       # Fix formatting issues
```

You can also run scripts in the specific workspaces

> Note: This will not use Turborepo caching

```
# Frontend specific
pnpm --filter client dev
pnpm --filter client build
pnpm --filter client test:e2e

# Backend specific
pnpm --filter server dev
pnpm --filter server build
pnpm --filter server test:socket
```

Tech Stack

- **Frontend:**
  - Next.js
  - TypeScript
  - Tailwind CSS
  - shadcn/ui
  - Monaco Editor (code editor)
  - Socket.IO Client
  - Sandpack (live preview)
  - MDXEditor (notepad)
  - simple-peer (WebRTC)
  - React Hook Form + Zod
- **Backend:**
  - Node.js
  - TypeScript
  - Socket.IO (binded to µWebSockets.js server)
- **Testing:**
  - Playwright (end-to-end testing for frontend)
  - Jest (unit testing for backend)
  - CodeQL (security analysis)
- **Code Quality:**
  - ESLint (static code analysis)
  - Prettier (code formatting)
  - Husky (git hooks)
  - commitlint (commit message linting)
- **Build & DevOps:**
  - Turborepo (monorepo build system)
  - GitHub Actions (CI/CD)
  - Vercel (frontend deployment)
  - Render (backend deployment)
- **Monitoring & Analytics:**
  - Sentry (error tracking)
  - Vercel Analytics (web analytics)
  - Cloudflare Web Analytics (web analytics)

    - Better Stack (uptime monitoring and status page)
- **External Services:**
    - Piston (code execution)
    - GitHub REST API (repository management)

Coding Style

We use several tools to maintain code quality:

- ESLint for static code analysis (frontend only)
- Prettier for code formatting
- prettier-plugin-sort-imports for import statement organization
- prettier-plugin-tailwindcss for Tailwind CSS class sorting (frontend only)
- prettier-plugin-classnames for wrapping long Tailwind CSS class names (frontend only)
- Husky for Git hooks
- lint-staged for running checks on staged files
- commitlint for commit message linting

Check and fix code style:

```
pnpm lint:check    # Check ESLint issues
pnpm lint:fix      # Fix ESLint issues
pnpm format:check  # Check formatting issues
pnpm format:fix    # Fix formatting issues
```

Contributing

Contributions are welcome! To contribute to this project, follow these steps:

1. Create a new branch for your feature:

    ```
    git checkout -b feat/your-feature-name
    ```

2. Commit your changes following **Conventional Commits**:

    ```
    git commit -m "<type>(<optional-scope>): <description>"
    ```

    - `<type>`: Must be one of:
        - `feat`: New features (e.g., "feat: add user authentication")
        - `fix`: Bug fixes (e.g., "fix: resolve memory leak")
        - `docs`: Documentation changes (e.g., "docs: update API guide")
        - `style`: Code style changes (e.g., "style: fix indentation")
        - `refactor`: Code refactoring (e.g., "refactor: simplify auth logic")
        - `perf`: Performance improvements (e.g., "perf: optimize database queries")
        - `test`: Adding/updating tests (e.g., "test: add unit tests for auth")
        - `chore`: Routine tasks/maintenance (e.g., "chore: update dependencies")

    - `ci` : CI/CD changes (e.g., "ci: add GitHub Actions workflow")
    - `revert` : Revert previous changes (e.g., "revert: remove broken feature")

   > For a complete commit message guidelines, see **Conventional Commits**.

3. Push your changes and submit a Pull Request with a description of your changes:

   ```
   git push origin feat/your-feature-name
   ```

User Manual

For detailed usage instructions and feature documentation, please refer to the **User Manual**.

License