@social.io/social.io

Documentation for @social.io/social.io

- readme.md for @social.io/social.io
- changelog.md for @social.io/social.io

readme.md for @social.io/social.io

> □□ All-in-one communication platform combining email, chat, calls, and video meetings — built with Deno, TypeScript, and the foss.global ecosystem.

Overview

social.io is a modern, self-hosted communication platform designed for organizations that need complete control over their messaging infrastructure. It provides a unified solution for team collaboration with real-time messaging, direct conversations, file sharing, and more.

□ Key Features

Feature	Status	Description
□□ **Real-time Chat**	□ Ready	Channels, DMs, threads, reactions, typing indicators
□□ **Organizations**	□ Ready	Multi-tenant support with workspaces and teams
□□ **Authentication**	□ Ready	JWT-based auth with session management
□□ **File Storage**	□ Ready	S3-compatible storage with thumbnails
□□ **Real-time Events**	□ Ready	WebSocket-powered live updates
□□ **Email Integration**	□□ Coming	IMAP/SMTP integration
□□ **Voice Calls**	□□ Coming	WebRTC peer-to-peer calling
□□ **Video Meetings**	□□ Coming	Scheduled video conferences with screen sharing

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Tech Stack

- **Runtime**: Deno 2.x+ – Secure, modern TypeScript runtime
- **Database**: MongoDB via @push.rocks/smartdata
- **Storage**: S3-compatible via @push.rocks/smartbucket
- **Real-time**: WebSocket with @push.rocks/smartsocket
- **Frontend** (coming): Angular 19 + @design.estate/dees-catalog

Quick Start

Prerequisites

- **Deno 2.x+** – Install Deno
- **MongoDB** – Local instance or cloud (Atlas, etc.)
- **S3-compatible storage** – MinIO (local), AWS S3, Cloudflare R2, etc.

1. Clone & Configure

```
git clone https://code.foss.global/design.estate/social.io.git
cd social.io
```

Create a `.nogit/env.json` file for local development:

```
{
  "MONGODB_URL": "mongodb://localhost:27017/socialio",
  "MONGODB_DB": "socialio",
```

```
  "S3_ENDPOINT": "http://localhost:9000",
  "S3_ACCESS_KEY": "minioadmin",
  "S3_SECRET_KEY": "minioadmin",
  "S3_BUCKET": "socialio",
  "JWT_SECRET": "your-super-secret-key-change-in-production",
  "PORT": "3000"
}
```

2. Start Development Server

```
deno task dev
```

This starts the server with hot-reloading enabled. Open http://localhost:3000 to see the landing page.

3. Production Deployment

Set environment variables and run:

```
deno task start
```

Or compile to standalone binaries:

```
# Build all platform binaries
deno task release

# Or build for specific platforms
deno task compile:linux-x64
deno task compile:linux-arm64
deno task compile:macos-x64
deno task compile:macos-arm64
```

Environment Variables

Variable	Default	Description
`MONGODB_URL`	`mongodb://localhost:27017`	MongoDB connection string

Variable	Default	Description
`MONGODB_DB`	`socialio`	Database name
`S3_ENDPOINT`	`http://localhost:9000`	S3 endpoint URL
`S3_ACCESS_KEY`	`minioadmin`	S3 access key
`S3_SECRET_KEY`	`minioadmin`	S3 secret key
`S3_BUCKET`	`socialio`	S3 bucket name
`S3_REGION`	-	S3 region (optional)
`JWT_SECRET`	(random)	JWT signing secret
`HOST`	`0.0.0.0`	Server bind address
`PORT`	`3000`	HTTP server port
`ENABLE_EMAIL`	`true`	Enable email features
`ENABLE_CHAT`	`true`	Enable chat features
`ENABLE_CALLS`	`true`	Enable call features
`ENABLE_MEETINGS`	`true`	Enable meeting features

API Reference

All API endpoints are prefixed with `/api/v1/`. Authentication uses Bearer tokens in the `Authorization` header.

□□ Authentication

```
# Register a new user
POST /api/v1/auth/register
{ "email": "user@example.com", "username": "johndoe", "password": "secret" }

# Login
POST /api/v1/auth/login
{ "email": "user@example.com", "password": "secret" }

# Get current user
GET /api/v1/auth/me

# Refresh access token
```

```
POST /api/v1/auth/refresh
{ "refreshToken": "..." }

# Logout
POST /api/v1/auth/logout

# List active sessions
GET /api/v1/auth/sessions
```

□□ Organizations

```
# List user's organizations
GET /api/v1/organizations

# Create organization
POST /api/v1/organizations
{ "slug": "my-org", "displayName": "My Organization" }

# Get/Update/Delete organization
GET    /api/v1/organizations/:orgId
PUT    /api/v1/organizations/:orgId
DELETE /api/v1/organizations/:orgId

# Manage members
GET    /api/v1/organizations/:orgId/members
PUT    /api/v1/organizations/:orgId/members/:userId
DELETE /api/v1/organizations/:orgId/members/:userId
```

□□ Invitations

```
# List organization invitations
GET /api/v1/organizations/:orgId/invitations

# Send invitation
POST /api/v1/organizations/:orgId/invitations
{ "email": "newuser@example.com", "role": "member" }
```

```
# Get invitation details (public)
GET /api/v1/invitations/:token

# Accept invitation
POST /api/v1/invitations/:token/accept

# Revoke invitation
DELETE /api/v1/organizations/:orgId/invitations/:invitationId
```

□□ Workspaces

```
GET    /api/v1/organizations/:orgId/workspaces
POST   /api/v1/organizations/:orgId/workspaces
GET    /api/v1/workspaces/:workspaceId
PUT    /api/v1/workspaces/:workspaceId
DELETE /api/v1/workspaces/:workspaceId
```

□□ Teams

```
# Team CRUD
GET    /api/v1/organizations/:orgId/teams
POST   /api/v1/organizations/:orgId/teams
GET    /api/v1/teams/:teamId
PUT    /api/v1/teams/:teamId
DELETE /api/v1/teams/:teamId

# Team members
GET    /api/v1/teams/:teamId/members
POST   /api/v1/teams/:teamId/members
PUT    /api/v1/teams/:teamId/members/:userId
DELETE /api/v1/teams/:teamId/members/:userId
```

□□ Channels

```
# Channel CRUD
GET    /api/v1/organizations/:orgId/channels
```

```
POST   /api/v1/organizations/:orgId/channels
GET    /api/v1/channels/:channelId
PUT    /api/v1/channels/:channelId
DELETE /api/v1/channels/:channelId

# Archive/Unarchive
POST /api/v1/channels/:channelId/archive
POST /api/v1/channels/:channelId/unarchive

# Channel membership
GET    /api/v1/channels/:channelId/members
POST   /api/v1/channels/:channelId/join
POST   /api/v1/channels/:channelId/leave
POST   /api/v1/channels/:channelId/members
PUT    /api/v1/channels/:channelId/members/:userId
DELETE /api/v1/channels/:channelId/members/:userId

# Notification settings
POST /api/v1/channels/:channelId/read
PUT  /api/v1/channels/:channelId/notifications
POST /api/v1/channels/:channelId/mute
POST /api/v1/channels/:channelId/unmute

# Messages
GET  /api/v1/channels/:channelId/messages
POST /api/v1/channels/:channelId/messages
GET  /api/v1/channels/:channelId/messages/search?q=keyword

# Pin messages
POST   /api/v1/channels/:channelId/pin/:messageId
DELETE /api/v1/channels/:channelId/pin/:messageId
```

✉ Messages

```
# Get/Edit/Delete message
GET    /api/v1/messages/:messageId
PUT    /api/v1/messages/:messageId
DELETE /api/v1/messages/:messageId
```

```
# Thread replies
GET /api/v1/messages/:messageId/replies

# Reactions
GET    /api/v1/messages/:messageId/reactions
POST   /api/v1/messages/:messageId/reactions
DELETE /api/v1/messages/:messageId/reactions/:emoji
```

□□ Direct Messages (Conversations)

```
# List conversations
GET /api/v1/conversations

# Start direct conversation
POST /api/v1/conversations/direct
{ "userId": "..." }

# Create group conversation
POST /api/v1/conversations/group
{ "participantIds": ["...", "..."], "name": "My Group" }

# Conversation management
GET  /api/v1/conversations/:conversationId
PUT  /api/v1/conversations/:conversationId
POST /api/v1/conversations/:conversationId/leave

# Participants
GET    /api/v1/conversations/:conversationId/participants
POST   /api/v1/conversations/:conversationId/participants
DELETE /api/v1/conversations/:conversationId/participants/:userId

# Messages
GET  /api/v1/conversations/:conversationId/messages
POST /api/v1/conversations/:conversationId/messages
```

□□ Attachments

```
# Upload (multipart/form-data or application/json with base64)
POST /api/v1/attachments/upload

# Get attachment info
GET /api/v1/attachments/:attachmentId

# Download
GET /api/v1/attachments/:attachmentId/download
GET /api/v1/attachments/:attachmentId/thumbnail

# Delete
DELETE /api/v1/attachments/:attachmentId

# Storage usage
GET /api/v1/organizations/:orgId/storage
```

□□ Health Check

```
GET /health    # Full health status with service states
GET /healthz   # Kubernetes-style health probe
```

WebSocket API

Connect to `/ws` for real-time updates.

Client → Server Events

```
// Connect and authenticate
socket.send(JSON.stringify({
  type: 'authenticate',
  userId: 'user-id',
  sessionId: 'session-id'
}));

// Subscribe to a channel
socket.send(JSON.stringify({
```

```
  type: 'subscribe',
  channelId: 'channel-id'
}));

// Typing indicators
socket.send(JSON.stringify({ type: 'typing_start', channelId: '...' }));
socket.send(JSON.stringify({ type: 'typing_stop', channelId: '...' }));

// Presence update
socket.send(JSON.stringify({ type: 'presence_update', status: 'online' }));

// Keep-alive
socket.send(JSON.stringify({ type: 'ping' }));
```

Server → Client Events

Event	Description
`connected`	Connection established
`authenticated`	Authentication successful
`subscribed`	Subscribed to channel
`message:new`	New message in channel
`typing:start` / `typing:stop`	Typing indicators
`presence:update`	User presence change
`pong`	Keep-alive response

Project Structure

```
social.io/
├── mod.ts                  # Entry point
├── deno.json               # Deno configuration
├── ts/
│   ├── index.ts            # Library exports
│   ├── plugins.ts          # Centralized dependencies
│   ├── socialio.ts         # Main SocialIO class
│   ├── cli.ts              # CLI commands
```

```
│   ├── interfaces/          # TypeScript interfaces
│   ├── models/              # MongoDB models (smartdata)
│   ├── services/            # Business logic services
│   ├── api/                 # REST API handlers
│   └── realtime/            # WebSocket managers
├── ui/                      # Angular 19 frontend (coming soon)
└── scripts/                 # Build scripts
```

Programmatic Usage

You can also use social.io as a library in your Deno project:

```
import { SocialIO, createSocialIOFromEnv } from '@social.io/social.io';

// Create from environment variables
const app = createSocialIOFromEnv();
await app.start();

// Or with explicit configuration
const app = new SocialIO({
  mongoUrl: 'mongodb://localhost:27017',
  mongoDb: 'socialio',
  s3Endpoint: 'http://localhost:9000',
  s3AccessKey: 'minioadmin',
  s3SecretKey: 'minioadmin',
  s3Bucket: 'socialio',
  port: 3000,
});

await app.init();
await app.start();

// Clean shutdown
await app.stop();
```

Development Commands

```
deno task dev          # Start dev server with hot-reload
deno task start        # Start production server
deno task test         # Run tests
deno task build        # Build Angular UI
deno task bundle-ui    # Bundle UI into embedded assets
deno task compile      # Compile to native binary
deno task release      # Full release build (UI + all binaries)
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

changelog.md for @social.io/social.io

2025-12-09 - 1.1.0 - feat(email)

Add email subsystem: interfaces and models for accounts, folders, threads and messages; integrate smartimap plugin; update exports and documentation

- Add comprehensive email interfaces (ts/interfaces/email.interfaces.ts) describing accounts, IMAP config, folders, threads, messages, drafts and realtime email events
- Introduce Email models: EmailAccount, EmailFolder, EmailThread, Email with persistence helpers, safe-object serializers and helper methods (ts/models/email.account.ts, ts/models/email.folder.ts, ts/models/email.thread.ts, ts/models/email.ts)
- Integrate smartimap into centralized plugins (ts/plugins.ts) and export it for use by email functionality
- Export email models from models index (ts/models/index.ts) so they are available through the central model exports
- Update README with project overview, quick start and programmatic usage documentation

2025-12-09 - 1.0.0 - initial

Initial release: base project skeleton and first commit.

- Added initial project files and repository baseline
- Establishes starting point for development