ForeSite - Comprehensive System Documentation & User Manual

Table of Contents

1. Product Overview

What is ForeSite?

ForeSite (internally codenamed "CodeFamily") is a **Repository Intelligence Platform** that provides deep, semantic analysis of GitHub repositories. It transforms raw code into actionable insights by:

- **Analyzing every commit** in a repository's history
- **Extracting code dependencies** using Tree-sitter parsing
- **Computing semantic similarity** via AI-generated embeddings
- **Calculating code ownership** based on contribution patterns
- **Detecting potential merge conflicts** in open Pull Requests
- **Generating AI-powered file summaries** using Groq LLMs

Who is ForeSite For?

User Role	Value Proposition
Engineering Managers	Understand who owns what code, identify knowledge silos
Tech Leads	Review PR conflict risks, find the right reviewers

Individual Contributors	Understand file dependencies, navigate unfamiliar codebases
DevOps/Platform Teams	Monitor repository health and team analytics

Core Capabilities

- Clone & Analyze
- Functions & Imports
- Embeddings
- Summaries
- Store
- Query
- Visualize

Key Features:

- **Repository Analytics** - Commit activity, file type distribution, contributor statistics
- **Code Ownership** - Semantic ownership percentages based on embedding contributions
- **Dependency Graphs** - Visual representation of file imports and dependents
- **PR Conflict Detection** - Risk scoring for open PRs that touch overlapping files
- **Similar File Discovery** - AI-powered identification of semantically similar files
- **AI File Summaries** - LLM-generated descriptions of file purpose

2. System Architecture

High-Level Architecture

ForeSite follows a **modular monolith architecture** with three runtime components:

- graph TD
- User[User Interface] -->|HTTP/REST| Backend[Backend .NET 8 API]
- User -->|React Query| Frontend[React Frontend]
-
- subgraph "Core System"
- Frontend
- Backend
- Sidecar[Node.js Sidecar]
- end
-
- Backend -->|gRPC/HTTP| Sidecar
- Backend -->|SQL| DB[(Supabase PostgreSQL)]
- Backend -->|API| GitHub[GitHub API]
- Backend -->|API| Gemini[Gemini AI]
- Backend -->|API| Groq[Groq LLM]

- Backend -->|Webhooks| Slack[Slack]
-
- Sidecar -->|Parsing| TreeSitter[Tree-sitter Parser]

Component Responsibilities

Component	Responsibility	Port
React Frontend	User interface, data visualization, OAuth flow	`5173`
C# Backend API	Business logic, GitHub integration, database operations	`5000`
Node.js Sidecar	Tree-sitter parsing for function/import extraction	`3002`
Supabase	PostgreSQL database with pgvector extension	Cloud

Request/Response Flow

`User Action → React Query → Backend API → Service Layer → Database/External APIs → Response → UI Update`

Typical API Call Lifecycle:

1. User clicks "Analyze Repository"
2. Frontend calls `POST /repositories/{owner}/{repo}/analyze`
3. `RepositoriesController` delegates to `AnalysisService`
4. `AnalysisService` clones repo, walks commits, extracts functions
5. Each function is sent to Sidecar for parsing
6. Embeddings are generated via Gemini AI
7. Results are stored in Supabase
8. Analysis status is updated and returned

3. Repository Structure

Root Directory Overview

- Foresite/
- ├── UPS-Project-Frontend/ # React TypeScript Frontend

- │   ├── src/
- │   │   ├── components/   # Reusable UI components
- │   │   ├── pages/   # Route-level page components
- │   │   ├── hooks/   # Custom React hooks
- │   │   ├── utils/   # API client & utilities
- │   │   ├── lib/   # Query client & shared utils
- │   │   └── App.tsx   # Main application entry
- │   ├── package.json   # Frontend dependencies
- │   ├── vite.config.ts   # Vite build configuration
- │   └── tailwind.config.js   # Tailwind CSS configuration
- │
- └── ups_foresite_backend/   # Backend Services
- ├── .env   # Environment variables (gitignored)
- ├── settings.json   # Application configuration
- ├── backend/   # .NET 8 API
- │   ├── src/
- │   │   ├── Api/   # API layer (controllers, configs)
- │   │   ├── Core/   # Domain layer (services, models)
- │   │   └── Workers/   # Background job workers
- │   ├── sql_migrations/   # Database migration scripts
- │   └── CodeFamily.sln   # Solution file
- │
- └── sidecar/   # Node.js Tree-sitter Service
- ├── src/
- │   ├── index.ts   # Express server entry
- │   └── parser/   # Tree-sitter adapters
- └── package.json   # Sidecar dependencies

Frontend Structure (UPS-Project-Frontend)

`/src/pages/` - Route Components

File	Purpose	Route
`Login.tsx`	GitHub OAuth login page	`/login`
`Dashboard.tsx`	Main landing page with repository lists	`/`
`RepoView.tsx`	Repository detail view with tabs	`/repo/:repositoryId`

`CommitView.tsx`	Commit details page	`/commit/:commitId`
`PRView.tsx`	Pull request list view	`/pr/:prId`
`PullRequestView.tsx`	PR details with files, reviewers	`/pr/:owner/:repo/:prNumber`
`FileView.tsx`	File content viewer with commit navigation	`/file/:fileId`
`FileTreeView.tsx`	File tree browser	`/filetree/:fileId`

`/src/components/` - UI Components

Component	Responsibility
`RepositoryAnalytics.tsx`	Charts for commits, file types, activity
`TeamInsights.tsx`	Code ownership, contributor graphs
`FileAnalysis.tsx`	File details, ownership, dependencies
`NetworkGraph.tsx`	SVG dependency visualization
`DependencyGraph.tsx`	React Flow-based dependency graph
`FileViewer.tsx`	Syntax-highlighted file content
`FileTree.tsx`	Hierarchical file browser

`ThemeSelector.tsx`	Theme switching UI
`ui/`	Shadcn-style primitives (Button, Card, etc.)

`/src/hooks/useApiQueries.ts` - Data Fetching Centralized React Query hooks for all API operations:

- `useAllRepositories()` - Fetch user's GitHub repositories
- `useRepository()` - Single repository details
- `useBranchCommits()` - Commits for a branch
- `useFileContent()` - File content at commit/branch
- `useRepositoryAnalytics()` - Analytics data
- `useTeamInsights()` - Team contribution data
- `useFileEnhancedAnalysis()` - Dependency/ownership analysis

`/src/utils/api.ts` - API Client Low-level API functions targeting `http://localhost:5000`:

- OAuth flow: `githubLogin()`, `githubCallback()`
- Repositories: `getRepositories()`, `analyzeRepository()`
- Files: `getFileAnalysis()`
- Pull Requests: `getPullRequestDetails()`

Backend Structure (ups_foresite_backend/backend)

`/src/Api/Controllers/` - REST Endpoints

Controller	Routes	Responsibility
`AuthController.cs`	`/auth/*`	GitHub OAuth login & callback
`RepositoriesController.cs`	`/repositories/*`	Repository CRUD, analysis, analytics
`CommitsController.cs`	`/commits/*`	Commit listing & details

`FilesController.cs`	`/files/*`	File metadata, content, analysis
`PullRequestsController.cs`	`/pullrequests/*`	PR listing & details
`BranchesController.cs`	`/api/repositories/{id}/branches/*`	Branch commits & files
`WebhooksController.cs`	`/webhooks/*`	GitHub webhook ingestion
`RepositoryRefreshController.cs`	`/api/repositories/{id}/refresh`	Incremental repo updates

`/src/Core/Services/` - Business Logic

Service	Purpose
`AnalysisService.cs`	Core ingestion pipeline: cloning, commit walking, function extraction, embedding generation, ownership calculation
`DatabaseService.cs`	PostgreSQL data access via Npgsql
`GitHubService.cs`	Octokit-based GitHub API client
`RepositoryService.cs`	LibGit2Sharp repository operations
`TreeSitterService.cs`	HTTP client to Node.js sidecar
`GeminiService.cs`	Gemini AI embedding generation

`GroqService.cs`	Groq LLM file summary generation
`SlackService.cs`	Slack DM notifications

`/src/Core/Interfaces/` - Service Contracts All services implement corresponding interfaces for dependency injection:

- `IAnalysisService` - Repository analysis operations
- `IDatabaseService` - All database queries
- `IGitHubService` - GitHub API operations
- `IRepositoryService` - Git operations
- `ITreeSitterService` - Code parsing
- `IGeminiService` - AI embeddings
- `IGroqService` - File summaries
- `ISlackService` - Notifications

`/src/Core/Models/` - Domain Models

- `FileContents`
- `DatabaseModels.cs` Entity classes: `User`, `Repository`, `Commit`, `Branch`, `RepositoryFile`, `FileChange`, `FileOwnership`, `Dependency`, `PullRequest`, etc.
- `DTOs.cs` Data transfer objects for API responses
- `AppSettings.cs` Strongly-typed configuration classes

`/src/Workers/IncrementalWorker.cs` - Background Processing

- Polls `webhook_queue` table for GitHub push events
- Processes incremental updates to repositories
- Calculates risk scores and sends Slack alerts for high-risk conflicts

Sidecar Structure (ups_foresite_backend/sidecar)

`/src/index.ts` - Express Server

- `POST /parse` - Parse code and extract functions/imports
- `GET /health` - Health check endpoint

`/src/parser/` - Language Parsers

File	Purpose

`treeSitterAdapter.ts`	Main parser orchestration for JS, TS, Python, Go, Java
`extractFunctions.ts`	AST traversal to find function definitions
`extractImports.ts`	AST traversal to find import statements

Supported Languages:

- JavaScript/JSX
- TypeScript/TSX
- Python
- Go
- Java

4. Technology Stack

Frontend Technologies

Technology	Version	Purpose
React	18.2.0	UI framework
TypeScript	5.2.2	Type safety
Vite	4.5.0	Build tool & dev server
TailwindCSS	3.4.1	Utility-first styling
React Query	5.90.12	Server state management
React Router	6.18.0	Client-side routing

Recharts	3.5.0	Data visualization
ReactFlow	11.10.1	Graph visualization
Framer Motion	11.0.0	Animations
Lucide React	0.556.0	Icons
Radix UI	Various	Accessible primitives

Backend Technologies

Technology	Version	Purpose
.NET	8.0	Runtime framework
ASP.NET Core	8.0	Web API framework
Npgsql	-	PostgreSQL driver
LibGit2Sharp	-	Git operations
Octokit	-	GitHub API client
DotNetEnv	-	Environment variable loading

Sidecar Technologies

Technology	Version	Purpose
Node.js	20.x	Runtime
Express	4.18.2	HTTP server
Tree-sitter	0.21.1	Code parsing
TypeScript	5.2.2	Type safety

External Services

Service	Purpose
Supabase	PostgreSQL database with pgvector
GitHub API	OAuth, repository data, webhooks
Gemini AI	Text embeddings (text-embedding-004)
Groq	LLM file summaries (llama-3.3-70b-versatile)
Slack	Conflict alert notifications

5. Configuration & Environment

Settings File (`settings.json`)

- {
- "ConnectionStrings": {

- "DefaultConnection": "${CONNECTION_STRING}"
- },
- "GitHub": {
- "AppId": "2322543",
- "ClientId": "${GITHUB_CLIENT_ID}",
- "ClientSecret": "${GITHUB_CLIENT_SECRET}",
- "WebhookSecret": "${GITHUB_WEBHOOK_SECRET}",
- "PrivateKeyPath": "/secrets/codefamily.pem"
- },
- "Gemini": {
- "ApiKey": "${GEMINI_API_KEY}"
- },
- "Groq": {
- "ApiKey": "${GROQ_API_KEY}",
- "ApiUrl": "https://api.groq.com/openai/v1/chat/completions",
- "Model": "llama-3.3-70b-versatile",
- "MaxTokens": 300,
- "Temperature": 0.3
- },
- "Supabase": {
- "Url": "${SUPABASE_URL}",
- "ServiceKey": "${SUPABASE_SERVICE_KEY}"
- },
- "Slack": {
- "BotToken": "${SLACK_BOT_TOKEN}"
- },
- "Sidecar": {
- "Url": "http://localhost:3002"
- },
- "WebhookUrl": "https://your-ngrok-url.ngrok-free.dev",
- "CloneBasePath": "./testdata/repos"
- }

Required Environment Variables

Create a `.env` file in `ups_foresite_backend/` with:

- # Database
- CONNECTION_STRING=postgresql://user:password@host:5432/database
-
- # GitHub OAuth App
- GITHUB_CLIENT_ID=your_client_id
- GITHUB_CLIENT_SECRET=your_client_secret
- GITHUB_WEBHOOK_SECRET=your_webhook_secret
-
- # GitHub App (for installation tokens)
- GITHUB_APP_CLIENT_ID=your_app_client_id

- GITHUB_APP_CLIENT_SECRET=your_app_client_secret
-
- # AI Services
- GEMINI_API_KEY=your_gemini_key
- GROQ_API_KEY=your_groq_key
-
- # Supabase
- SUPABASE_URL=https://your-project.supabase.co
- SUPABASE_SERVICE_KEY=your_service_key
-
- # Slack (optional)
- SLACK_BOT_TOKEN=xoxb-your-bot-token

GitHub App Private Key Place your GitHub App private key PEM file at: `/secrets/codefamily.pem`

6. Installation & Setup Guide

Prerequisites

Requirement	Minimum Version
Node.js	20.x
.NET SDK	8.0
Git	2.x
PostgreSQL	14.x (via Supabase)

Step-by-Step Setup

1. Clone the Repository

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

2. Configure Environment

- cd ups_foresite_backend
- cp .env.example .env
- # Edit .env with your credentials

3. Run Database Migrations Execute the SQL in `backend/sql_migrations/` via Supabase SQL Editor:

- `multi_user_repository_access.sql` - Creates multi-user access table

4. Start the Sidecar Service

- cd ups_foresite_backend/sidecar
- npm install
- npm run build
- npm start
- # Runs on http://localhost:3002

5. Start the Backend API

- cd ups_foresite_backend/backend/src/Api
- dotnet run
- # Runs on http://localhost:5000

6. Start the Frontend

- cd UPS-Project-Frontend
- npm install
- npm run dev
- # Runs on http://localhost:5173

7. Access the Application Open `http://localhost:5173` in your browser.

Common Setup Errors

Error	Cause	Solution
Connection refused on 5000	Backend not running	Start backend with `dotnet run`
Connection refused on 3002	Sidecar not running	Start sidecar with `npm start`

GitHub OAuth failed	Wrong redirect URI	Ensure GitHub OAuth app has `http://localhost:5173` as callback
PEM file not found	Missing private key	Place `.pem` file at `/secrets/codefamily.pem`
Database connection error	Wrong connection string	Verify `CONNECTION_STRING` in `.env`

7. Runtime Behavior & Execution Flow

Application Startup

Backend Startup (`Program.cs`):

- Loads environment variables from `.env`
- Reads `settings.json` with variable substitution
- Registers all services with DI container
- Starts background worker (`IncrementalWorker`)
- Configures CORS for frontend origin
- Starts listening on port 5000

Frontend Startup (`main.tsx` → `App.tsx`):

- Initializes React Query client
- Checks for OAuth code in URL (GitHub redirect)
- Loads user from localStorage if available
- Renders route based on authentication state

Repository Analysis Flow

- sequenceDiagram
- participant User
- participant Frontend
- participant Backend
- participant Sidecar
- participant Gemini
- participant DB
-
- User->>Frontend: Click "Analyze"
- Frontend->>Backend: POST /repositories/{owner}/{repo}/analyze
- Backend->>Backend: Clone repository (bare)
-

- loop For each commit
- Backend->>Backend: Walk all commits chronologically
- Backend->>Backend: Get changed files
- Backend->>Sidecar: POST /parse (code, language)
- Sidecar-->>Backend: Tree-sitter parse {functions, imports}
-
- Backend->>Gemini: Generate embedding
- Gemini-->>Backend: float[768]
-
- Backend->>DB: Store embedding
- Backend->>Backend: Create dependencies
- Backend->>Backend: Calculate ownership
- end
-
- Backend->>Backend: Register webhook
- Backend-->>Frontend: Analysis complete
- Frontend-->>User: Show results

Incremental Update Flow (Webhooks)

- sequenceDiagram
- participant GitHub
- participant WebhookEndpoint
- participant WebhookQueue
- participant IncrementalWorker
- participant AnalysisService
- participant Slack
-
- GitHub->>WebhookEndpoint: POST /webhooks/github
- WebhookEndpoint->>WebhookQueue: Insert payload
- WebhookEndpoint-->>GitHub: 202 Accepted
-
- IncrementalWorker->>WebhookQueue: Poll for pending
- IncrementalWorker->>AnalysisService: ProcessIncrementalUpdate
- AnalysisService->>AnalysisService: Analyze changed files
- AnalysisService->>AnalysisService: Calculate risk score
-
- alt Risk >= 80%
- AnalysisService->>GitHub: Create commit status (failure)
- AnalysisService->>Slack: Send DM alert
- end
-
- IncrementalWorker->>WebhookQueue: Mark completed

8. Data Layer & Persistence

Database Schema

ForeSite uses **Supabase PostgreSQL** with the following core tables:

- erDiagram
- users ||--o{ repositories : connects
- users ||--o{ commits : has
- repositories ||--o{ branches : contains
- repositories ||--o{ commits : contains
- commits ||--o{ file_changes : modifies
- repository_files ||--o{ file_changes : changed_in
- repository_files ||--o{ file_ownership : owned_by
- repository_files ||--o{ dependencies : depends_on
- repository_files ||--o{ code_embeddings : has
- repositories ||--o{ pull_requests : has
-
- users {
- uuid id PK
- bigint github_id
- string author_name
- string email
- string avatar_url
- }
- repositories {
- uuid id PK
- string name
- string owner_username
- string status
- uuid connected_by_user_id FK
- string last_analyzed_commit_sha
- }
- branches {
- uuid id PK
- uuid repository_id FK
- string name
- string head_commit_sha
- boolean is_default
- }
- commits {
- uuid id PK
- uuid repository_id FK
- string sha
- string message
- string author_name
- uuid author_user_id FK
- timestamp committed_at
- }
- repository_files {
- uuid id PK

- uuid repository_id FK
- string file_path
- int total_lines
- }
- file_changes {
- uuid commit_id FK
- uuid file_id FK
- int additions
- int deletions
- }
- file_ownership {
- uuid file_id FK
- string author_name
- decimal semantic_score
- }
- dependencies {
- uuid source_file_id FK
- uuid target_file_id FK
- string dependency_type
- }
- code_embeddings {
- uuid id PK
- uuid file_id FK
- vector embedding
- string chunk_content
- }
- pull_requests {
- uuid id PK
- uuid repository_id FK
- int pr_number
- string title
- string state
- }

Embedding Storage

Code embeddings are stored using `pgvector` extension:

- Model: Gemini `text-embedding-004`
- Dimensions: 768 floats
- Used for: Semantic similarity search, ownership calculation

Data Access Patterns

`DatabaseService.cs` uses raw Npgsql for:

- Direct SQL queries with parameterization

- Efficient bulk inserts
- Vector similarity search via `<=>` operator

9. Feature Documentation

9.1 Repository Analysis

What it does:

- Clones repository using LibGit2Sharp
- Walks commits in chronological order
- Extracts functions/imports via Tree-sitter
- Generates embeddings per code chunk
- Calculates semantic ownership

API Endpoint: `POST /repositories/{owner}/{repo}/analyze`

9.2 Code Ownership

Calculation Method:

1. For each commit, calculate the embedding delta (Euclidean distance)
2. Assign delta contribution to commit author
3. Aggregate contributions per file
4. Normalize to percentages

API Endpoint: `GET /files/{fileId}` → `ownership` array

9.3 Dependency Detection

Detection Strategy:

- Parse imports using Tree-sitter
- Resolve import paths to repository files
- Distinguish internal vs external dependencies
- Store bidirectional relationships

Supported Import Patterns:

- // JavaScript/TypeScript
- import { foo } from './utils'
- const bar = require('./helper')

- # Python
- import os
- from django.db import models

- // Go

- import "github.com/user/repo"

- // Java
- import com.example.package.Class

9.4 AI File Summaries

Technology: Groq + Llama 3.3 70B **Process:**

1. Retrieve code chunks from embeddings table
2. Limit to 8 chunks, 6000 chars max
3. Send to Groq with structured prompt
4. Cache result for subsequent requests

API Endpoint: `GET /files/{fileId}/summary`

9.5 PR Conflict Detection

Risk Calculation: `Risk = (Structural Overlap × 0.4) + (Semantic Overlap × 0.6)`

- **Structural Overlap:** Files modified in both push and open PR
- **Semantic Overlap:** Cosine similarity of affected embeddings

Threshold: Risk >= 80% triggers:

- GitHub commit status (failure/blocked)
- Slack DM to pusher

10. User Guide

10.1 Logging In

1. Navigate to `http://localhost:5173`
2. Click **Login with GitHub** (if OAuth is configured)
3. Or click **Get Started** to continue without OAuth

10.2 Viewing Your Repositories

Dashboard Tab: "Your Repositories"

- Lists all GitHub repositories accessible to your account
- Shows analysis status (Not Analyzed / Ready)
- Pagination for large repository counts

Dashboard Tab: "Analyzed Repositories"

- Repositories that have been fully analyzed
- Filter: Your repos / Others / All

10.3 Analyzing a Repository

Option 1: From Your Repositories

1. Find the repository in the list
2. Click **Analyze** button
3. Wait for analysis to complete (progress shown)

Option 2: By URL

1. Click **Add Repository** tab
2. Enter GitHub repository URL
3. Click **Analyze Repository**

10.4 Exploring Repository Details

After analysis, click **View Analysis** to access:

- **Commits Tab:** List of all commits with author, message, date. Click commit to see changed files.
- **Pull Requests Tab:** Open and closed PRs. Click to see files changed, recommended reviewers.
- **Files Tab:** Full file tree of repository. Click any file for detailed analysis.
- **Analytics Tab:** Commit activity over time, File type distribution, Top contributors, AI-generated summary.

10.5 File Analysis

When viewing a file, you see:

Section	Information
File Info	Path, lines, last modified
Ownership	Contributors with percentage ownership
Dependencies	Files this file imports
Dependents	Files that import this file

Similar Files	Semantically related files
Dependency Graph	Visual network of relationships
AI Summary	LLM-generated description

10.6 Theme Selection

Click the theme icon in the header to choose:

- **Black & Beige** - Warm, earthy tones
- **Light** - Clean white interface
- **Dark** - Deep blue-black
- **Night** - Warm sandy palette
- **Light Palette** - Cream and tan

11. Troubleshooting

Common Problems

Problem	Possible Cause	Solution
Login redirects to blank page	OAuth misconfigured	Check GitHub OAuth app callback URL
Analysis hangs at "Cloning"	Large repository	Wait longer; check backend logs
No dependencies shown	Unsupported language	Only JS/TS/Python/Go/Java supported
Empty ownership data	No embeddings generated	Check Gemini API key
"Rate limited" on summary	Groq API limit hit	Wait 60 seconds and retry

WebSocket connection failed	Supabase realtime issues	Refresh page

Checking Service Health

Backend: `curl http://localhost:5000/health`

- { "status": "healthy", "timestamp": "..." }

Sidecar: `curl http://localhost:3002/health`

- { "status": "healthy", "timestamp": "..." }

Debugging Tips

- **Backend Logs:** View console output from `dotnet run`
- **Frontend Console:** Open browser DevTools → Console
- **Network Tab:** Check for failed API requests
- **Database:** Query Supabase directly for data issues

Resetting Analysis

To re-analyze a repository:

1. Delete repository record from database
2. Remove cloned files from `./testdata/repos/`
3. Re-trigger analysis from frontend

Appendix A: API Reference Summary

Method	Endpoint	Description
`GET`	`/health`	Health check
`GET`	`/auth/github`	Start OAuth flow
`POST`	`/auth/github/callback`	Complete OAuth

GET	/repositories	List user repositories
POST	/repositories/{owner}/{repo}/analyze	Start analysis
GET	/repositories/{id}	Get repository
GET	/repositories/{id}/analytics	Get analytics
GET	/repositories/{id}/team-insights	Get team data
GET	/commits/{id}	Get commit
GET	/commits/{id}/github-details	Get GitHub commit data
GET	/files/{id}	Get file analysis
GET	/files/{id}/content	Get file content
GET	/files/{id}/commits	Get file history
GET	/files/{id}/summary	Get AI summary
GET	/pullrequests/repository/{id}	List PRs
GET	/pullrequests/{owner}/{repo}/{number}/details	Get PR details