# Dispute Resolution Engine

## Coding Standard

Formatting Tools & Configurations ..... 2
- Prettier ..... 2
- Go Format ..... 2

Git Conventions ..... 2
- Branch Naming Rules ..... 2
- Commit Naming Rules ..... 2

File Structure ..... 3

# Formatting Tools & Configurations

## Prettier

We use Prettier for code formatting to ensure consistency across different files and contributors in all our JavaScript/TypeScript projects. Our Prettier configuration is as follows:

```
{
  "tabWidth": 2,
  "useTabs": false,
  "printWidth": 100,
  "singleQuote": false,
  "bracketSameLine": false,
  "arrowParens": "always",
  "endOfLine": "lf"
}
```

## Go Format

For our Go-based projects, we use gofmt along with the opinionated formatting style prescribed by the Go development team. This ensures that all our Go code follows the standard style guidelines set by the Go community.

# Git Conventions

## Branch Naming Rules

Branches in our repository are categorized into the following types:

- feat/xxx: Feature branches for new additions to the codebase
- docs/xxx: Documentation branches for updates or changes to project documentation
- hotfix/xxx: Urgent changes or patches to be applied to the main branch

## Commit Naming Rules

Commit messages should be concise and descriptive, following this format:

- feat: xxx for commits that add new features to the project
- refac: xxx for code refactoring that doesn't add new features
- fix: xxx for bug fixes
- chore: xxx for general maintenance or structural changes without new features
- docs: xxx for updates or changes made to documentation

# File Structure

Our repository follows a monorepo structure, with each component of the project defined in a sub-folder at the root:

Example:
├── api <---- Go API
├── frontend <---- Next.js frontend
└── initdb <---- SQL scripts to initialize the database