

[Event dashboard](#) > Lab 2: Steering

Lab 2: Steering

Overview

In this lab, you will learn how to use Kiro's Steering feature to provide contextual guidance and standards to your AI agent. Steering allows you to establish project-specific rules, coding standards, and architectural guidelines that Kiro will follow when generating code and making suggestions.

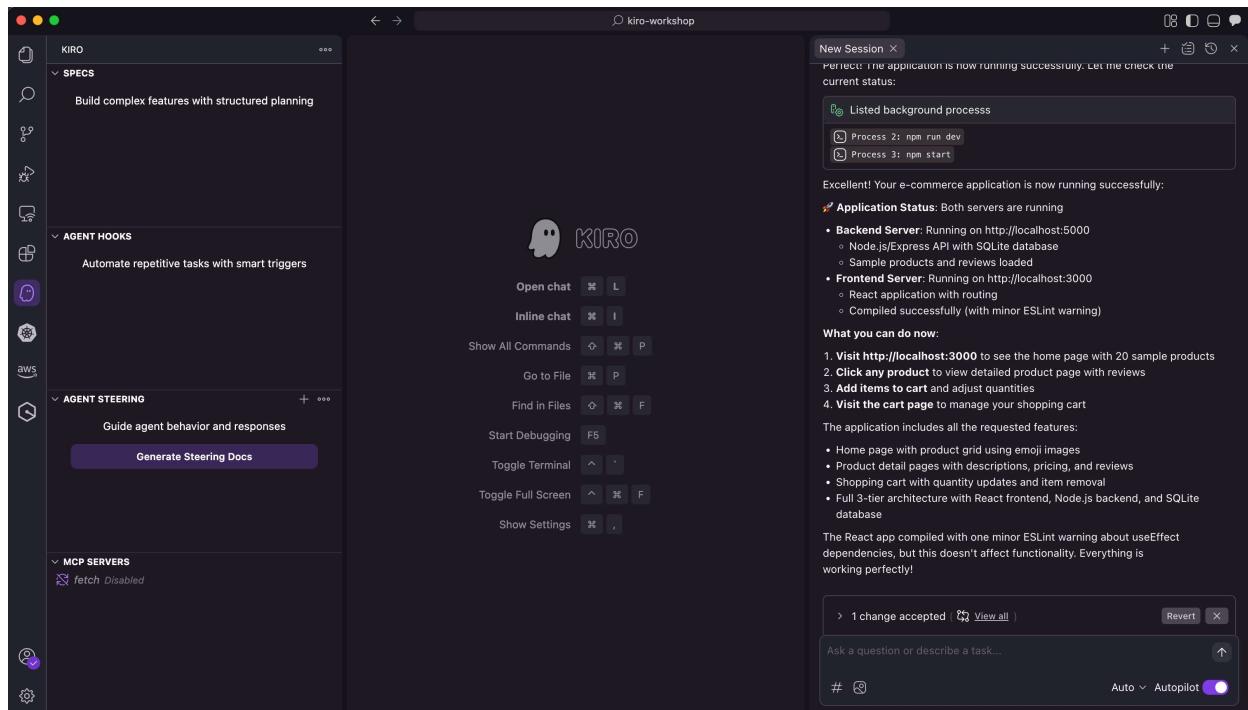
You'll discover how to:

- **Generate foundational steering documents** that define your product overview, technology stack, and project structure
- **Create custom steering files** with development standards and best practices specific to your e-commerce project
- **Configure steering inclusion rules** to control when specific guidance applies
- **Leverage steering to maintain consistency** across your codebase and ensure AI-generated code follows your established patterns

Steering acts as a persistent memory for Kiro, helping it understand your project's context, constraints, and preferences. This results in more relevant suggestions, better code quality, and faster development cycles as the AI agent becomes familiar with your specific requirements and coding standards.

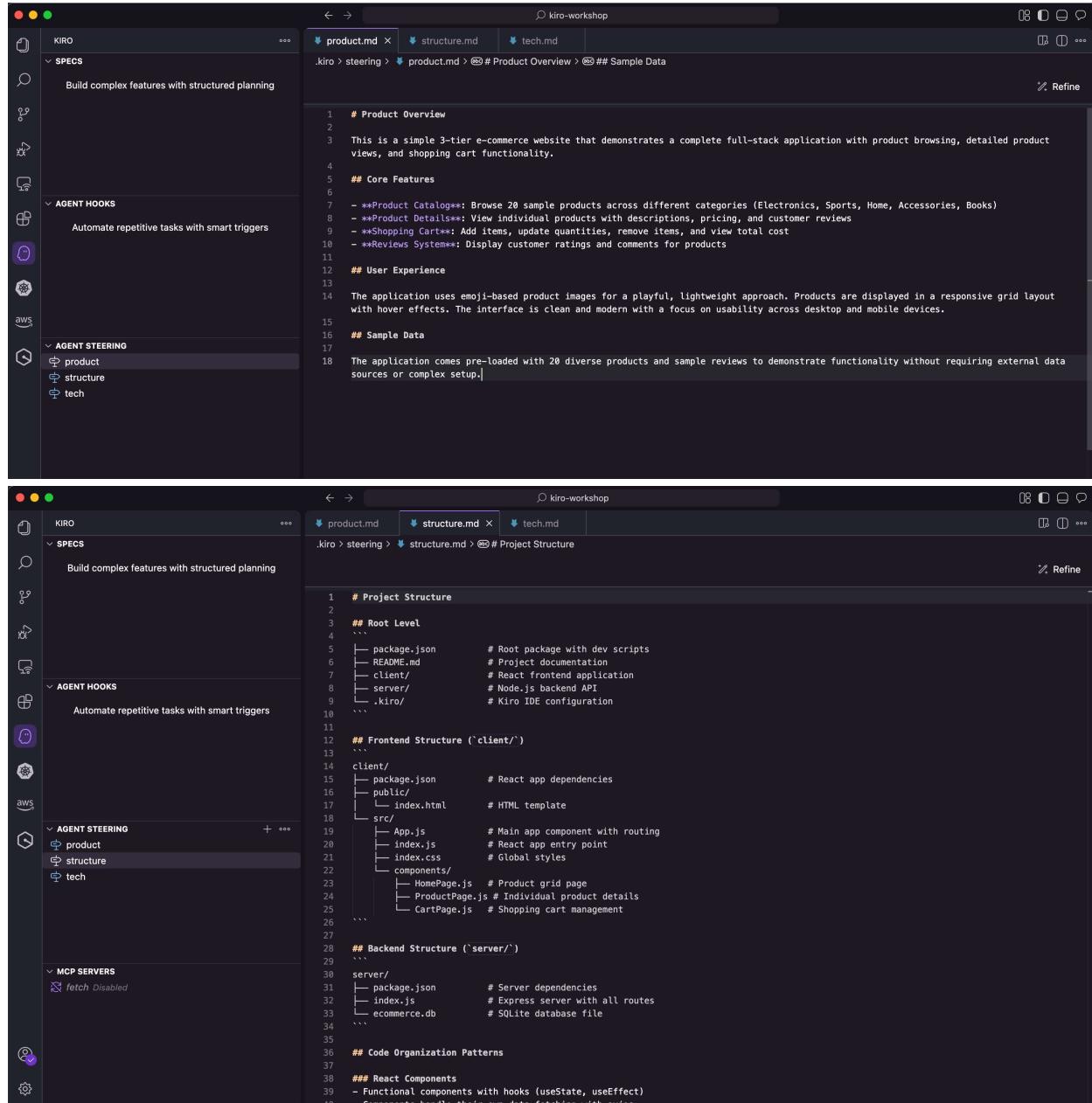
Generate Steering Docs

Click the Kiro Ghost icon in the activity bar (left sidebar). In **AGENT STEERING** section, click **Generate Steering Docs** button for foundational steering files to establish core project context.



Kiro will create three foundational files:

- 1. Product Overview** (`product.md`) - Defines your product's purpose, target users, key features, and business objectives. This helps Kiro understand the "why" behind technical decisions and suggest solutions aligned with your product goals.
- 2. Technology Stack** (`tech.md`) - Documents your chosen frameworks, libraries, development tools, and technical constraints. When Kiro suggests implementations, it will prefer your established stack over alternatives.
- 3. Project Structure** (`structure.md`) - Outlines file organization, naming conventions, import patterns, and architectural decisions. This ensures generated code fits seamlessly into your existing codebase.



The screenshot shows the Kiro IDE interface with two tabs open: `product.md` and `structure.md`. The left sidebar contains sections for **SPECS**, **AGENT HOOKS**, and **AGENT STEERING**. The **AGENT STEERING** section has three sub-options: `product`, `structure`, and `tech`.

product.md Content:

```

1 # Product Overview
2
3 This is a simple 3-tier e-commerce website that demonstrates a complete full-stack application with product browsing, detailed product views, and shopping cart functionality.
4
5 ## Core Features
6
7 - **Product Catalog**: Browse 20 sample products across different categories (Electronics, Sports, Home, Accessories, Books)
8 - **Product Details**: View individual products with descriptions, pricing, and customer reviews
9 - **Shopping Cart**: Add items, update quantities, remove items, and view total cost
10 - **Reviews System**: Display customer ratings and comments for products
11
12 ## User Experience
13
14 The application uses emoji-based product images for a playful, lightweight approach. Products are displayed in a responsive grid layout with hover effects. The interface is clean and modern with a focus on usability across desktop and mobile devices.
15
16 ## Sample Data
17
18 The application comes pre-loaded with 20 diverse products and sample reviews to demonstrate functionality without requiring external data sources or complex setup.

```

structure.md Content:

```

1 # Project Structure
2
3 ## Root Level
4
5 package.json      # Root package with dev scripts
6 README.md         # Project documentation
7 client/           # React frontend application
8 server/           # Node.js backend API
9 .kiro             # Kiro IDE configuration
10 ...
11
12 ## Frontend Structure ('client')
13 ...
14 client/
15   package.json     # React app dependencies
16   public/
17     index.html    # HTML template
18   src/
19     App.js        # Main app component with routing
20     index.js      # React app entry point
21     index.css     # Global styles
22     components/
23       HomePage.js # Product grid page
24       ProductPage.js # Individual product details
25       CartPage.js # Shopping cart management
26 ...
27
28 ## Backend Structure ('server')
29 ...
30 server/
31   package.json     # Server dependencies
32   index.js        # Express server with all routes
33   ecommerce.db    # SQLite database file
34 ...
35
36 ## Code Organization Patterns
37
38 ## React Components
39 - Functional components with hooks (useState, useEffect)
40   Components handle their own data fetching with axios

```

The screenshot shows the Kiro application interface. On the left, there's a sidebar with various icons and sections like 'SPECS', 'AGENT HOOKS', 'AGENT STEERING', and 'MCP SERVERS'. The main area displays a file named 'tech.md' with the following content:

```

1 # Technology Stack
2
3 ## Architecture
4 - **Frontend**: React 18 with React Router for client-side routing
5 - **Backend**: Node.js with Express.js REST API
6 - **Database**: SQLite with sqlite3 driver
7 - **Development**: Concurrent development servers using concurrently
8
9 ## Key Dependencies
10
11 ### Frontend (React)
12 - `react` & `react-dom` - Core React framework
13 - `react-router-dom` - Client-side routing
14 - `axios` - HTTP client for API calls
15 - `react-scripts` - Create React App tooling
16
17 ### Backend (Node.js)
18 - `express` - Web framework
19 - `cors` - Cross-origin resource sharing
20 - `sqlite3` - SQLite database driver
21 - `uuid` - Unique ID generation for cart items
22 - `nodemon` - Development auto-reload
23
24 ## Common Commands
25
26 ### Development Setup
27 ````bash
28 # Install all dependencies (root, server, client)
29 npm run install-all
30
31 # Start both servers concurrently
32 npm run dev
33
34
35 ### Individual Services
36 ````bash
37 # Backend only (runs on port 5000)
38 cd server && npm run dev
39
40 # Frontend only (runs on port 3001)

```

Add Custom Steering Docs

Create Custom Steering on Your Own

© 2008 – 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy policy](#) [Terms of use](#) [Cookie preferences](#)

needs. For example:

1. Navigate to the Steering section in the Kiro panel
2. Click the + button to create a new .md file
3. Choose a descriptive filename (e.g. python-venv.md)
4. Copy the below guidance which is standard markdown syntax, and Save

1 # Technology Stack and Conventions
2
3 ## Python Environment Management
4 - **Virtual Environments:** Always use Python virtual environments (venv) for dependency management.
5 - **Preferred Tool:** Utilize `venv` for creating and managing virtual environments.
6 - **Activation:** Ensure virtual environments are activated before running Python scripts or install
7 - **Dependency Installation:** Install project dependencies within the activated virtual environment



The screenshot shows the Kiro IDE interface with two tabs open: 'python-venv.md' and 'python-venv'. The left sidebar contains sections for 'SPECS', 'AGENT HOOKS', 'AGENT STEERING', and 'MCP SERVERS'. The right pane displays the contents of the selected file.

```

1 # Technology Stack and Conventions
2
3 ## Python Environment Management
4 - **Virtual Environments:** Always use Python virtual environments (venv) for dependency management.
5 - **Preferred Tool:** Utilize 'venv' for creating and managing virtual environments.
6 - **Activation:** Ensure virtual environments are activated before running Python scripts or installing packages.
7 - **Dependency Installation:** Install project dependencies within the activated virtual environment using 'pip'.

```

Upload Custom Steering File

You can also upload the steering file to `.kiro/steerings` directly. Download the two sample steering files:

- [development-standards.md ↗](#)
- [python-best-practices.md ↗](#)

and then copy the file to `.kiro/steerings`.

```

1  ---
2  #title: Development Standards
3  #inclusion: always
4  ---
5
6  # Development Standards
7
8  ## Dependency Management
9  - Use latest stable versions of all libraries and dependencies
10 - Leverage Context7 MCP server to verify compatibility before adding dependencies
11 - Justify each new dependency with clear business or technical value
12 - Prefer well-maintained libraries with active communities
13 - Document version constraints in project files
14 - Remove unused dependencies regularly
15 - Use lock files to ensure consistent installations across environments
16
17  ## Code Quality Standards
18  - Never create duplicate files with suffixes like '_fixed', '_clean', '_backup', etc.
19  - Work iteratively on existing files (hooks handle commits automatically)
20  - Include relevant documentation links in code comments
21  - Follow language-specific conventions (TypeScript for CDK, Python for Lambda)
22  - Use meaningful variable and function names
23  - Keep functions small and focused on single responsibilities
24  - Implement proper error handling and logging
25
26  ## File Management
27  - Maintain clean directory structures
28  - Use consistent naming conventions across the project
29  - Avoid temporary or backup files in version control
30  - Organize code logically by feature or domain
31  - Keep configuration files at appropriate levels (project vs user)
32
33  ## Documentation Approach
34  - Maintain single comprehensive README covering all aspects including deployment
35  - Reference official sources through MCP servers when available
36  - Update documentation when upgrading dependencies
37  - Keep documentation close to relevant code
38  - Use inline comments for complex business logic
39  - Document API endpoints and data structures
40  - Include setup and deployment instructions
41
42
43
44
45
46
47
48

```

Steering Examples

You may find more steering examples for reference here:

- Kiro Best Practice - Steering Example ↗

[Previous](#)
[Next](#)