Onboarding Developers to Codebases:

A Hands-On Guide

GitKraken

Onboarding Developers to Codebases:

A Hands-On Guide

Summary

It is absolutely awesome to have new people join your team. You're going to be able to do more, better, quicker. In the long run.

But there is always that angst associated with those first few weeks. Are they going to fit in? Are they going to be effective? How do you make them effective?

Getting developers up-and-running with a codebase is a challenge. This is true whether they are junior developers new to the company, or senior developers from another team in the organization that need to understand your code. Either way, you need to make them effective contributors to your codebase.

This used to be done with sitting down 1-on-1 with them and walking through everything. But there is a better way to onboard–codebase maps and tours. You can visualize your codebase for new team members, show them what they need to know quickly, and then tailor a step-by-step walkthrough just for them.

In this guide we want to take you through how to do that, using maps and tours, and then show you how to take things further with automation to make your team even more effective.

Onboarding Developers to Codebases:

A Hands-On Guide

GitKraken

3 Barriers to Developer or Contributor Productivity

So what slows everything down in those first few weeks? It can really be condensed down to three things:

1. Not Understanding the Architecture of the Codebase

This is the most obvious issue. New devs just don't know the codebase. If the new developer is familiar with the specific framework they'll know the rough architecture, but every codebase has its own patterns. Not understanding the design then leads to underlying problems:

- **Inconsistent code quality:** The developer may not be aware of the best practices or patterns used in the codebase, leading to inconsistencies in code quality and structure.
- **Increased risk of bugs:** Without understanding the architecture, the developer may make changes that introduce new bugs or break existing functionality.
- **Slow development:** A lack of understanding of the architecture can result in the developer taking longer to complete tasks or making unnecessary changes, slowing down the development process.
- **Poor code maintainability:** Changes made without an understanding of the architecture can result in code that is difficult to maintain, requiring more time and effort to update or fix in the future.

To avoid these problems, it is important to provide developers with access to the necessary information and resources to understand the architecture of the codebase.

2. Slow Code Review Process

In an analysis of 1M code reviews, Linearb found that the average time to review was more than four days. Anything that can speed up the process needs to be done. When developers, junior or senior, are new to a codebase, review cycles get longer

- If the new developer is junior and pushing code, they might not understand the dependencies in the codebase and the implications of their PR. Additionally, the junior developer may also be less familiar with the development process and best practices, which can result in suboptimal solutions being suggested or missed issues being overlooked.

- If the new developer is senior and reviewing code, then they have to grasp the codebase themselves first before they can start to help others. They will have difficulty in understanding the purpose of specific parts of the code, and an initial inability to identify areas of the code that may require attention or improvement. The senior dev will spend more time understanding the code and its implications before providing feedback.

3. Lack of Organizational Knowledge

New developers lack organizational knowledge. Even if they are coming from a different part of the company, they'll still not know exactly how your team works. This results in several problems, including

- **Inefficient work:** The lack of understanding of how the organization operates can result in the person taking longer to complete tasks, making unnecessary changes, or missing important details.

- **Decreased collaboration:** Without understanding the organizational structure, goals, and processes, it can be difficult for the person to effectively collaborate with others or understand their roles and responsibilities.

- **Poor decision making:** The lack of organizational knowledge can result in poor decision making, as the person may not fully understand the context or impact of their actions.

- **Increased risk of mistakes:** The lack of understanding of organizational policies, procedures, and systems can increase the risk of mistakes being made.

So you need to give people more knowledge about your organization and architecture, and you can do that through codebase maps and tours. Over a quarter, Distribute Aid saved an estimated 72 hours in onboarding time with GitKraken code visibility, and received more contributions to its projects than ever before—9 pull requests from new contributors.

Make a Codebase Map

Codebase maps are so much more than just code diagrams. Maps provide developers and teams a means of sharing key information about code—all within the context of the complete codebase.

What is a codebase map?

A codebase map is a visual representation of the structure and organization of a codebase. It helps developers to understand the relationships between different components of the code and how they fit together, making it easier to navigate the codebase and identify areas of interest.

They can be used to represent different aspects of the codebase, such as the relationships between classes, modules, and functions, or the flow of data and control through the code.

By providing a high-level view of the codebase, codebase maps can help developers to quickly and easily understand the code, reducing the time and effort required to familiarize themselves with new codebases or understand complex systems. They can also help to identify potential issues or areas of the code that may require attention or improvement, improving the overall quality and reliability of the code.

Why should you create a codebase map?

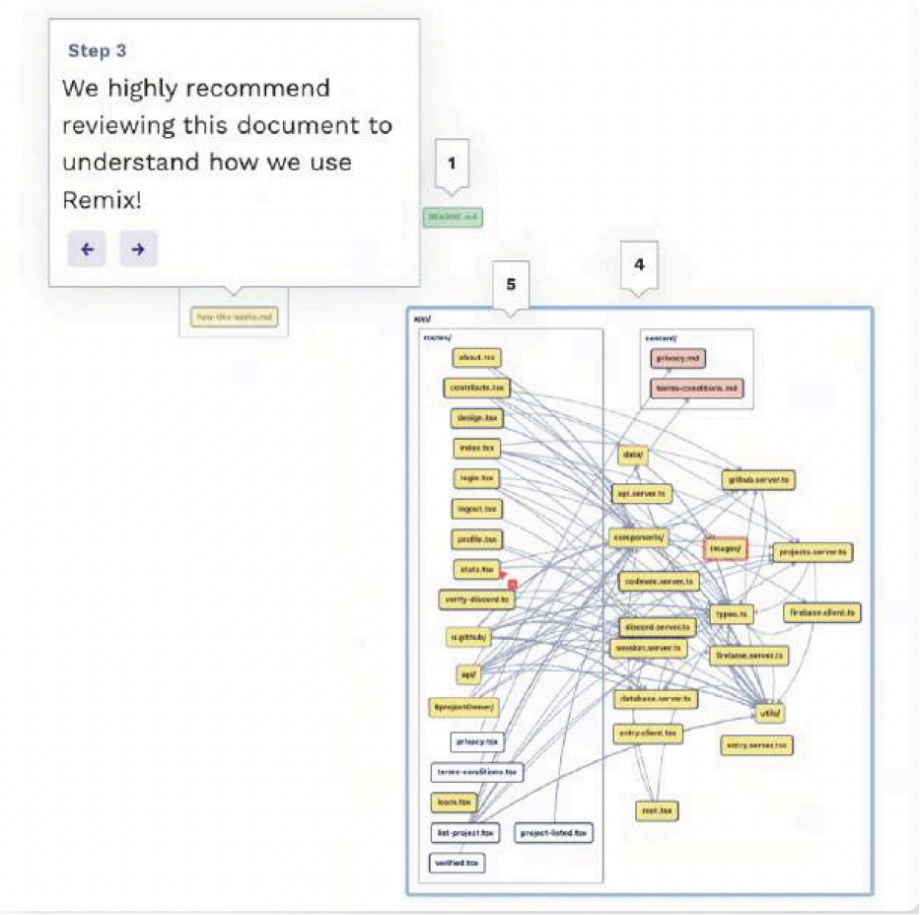
Creating a codebase map can provide several benefits, including

- **Improved code understanding:** A codebase map can help developers to quickly and easily understand the structure and organization of a codebase, reducing the time and effort required to familiarize themselves with new codebases or understand complex systems.
- **Increased collaboration:** By providing a shared understanding of the codebase, codebase maps can help to improve collaboration between developers, making it easier to work together on complex systems.
- **Better code maintainability:** Codebase maps can help to identify areas of the code that may require attention or improvement, improving the overall quality and maintainability of the code.
- **Faster bug fixing:** By providing a clear understanding of the relationships between different components of the code, codebase maps can help developers to identify and fix bugs more quickly.
- **Reduced development time:** A clear understanding of the codebase can help developers to complete tasks more quickly, reducing the time required to develop new features or fix existing issues.
- **Improved code reviews:** Codebase maps can help to make code reviews more efficient, by making it easier to identify potential issues or the impact of changes.

Creating a codebase map improves the quality, maintainability, and efficiency of a codebase, making it easier to develop high-quality products. It also makes people quicker. As Joan M. Davis told us:

> “Using GitKraken to take on a new codebase, I’m able to save up to 4 hours a week across my open-source projects.”
>
> **Joan M. Davis**

Let’s quickly go through what a codebase map looks like, particularly one focused on onboarding new developers. Here’s an example onboarding codebase map:



This map is for a Remix project. At a glance, a new developer can:

- Understand the design of the codebase. It’s clear how the code is organized. This is a route and component-based architecture with different concerns separated into different folders.
- See where the docs are to help them get started and orient them into the code.
- Follow the walkthrough to know the steps they need to follow to work on the code.

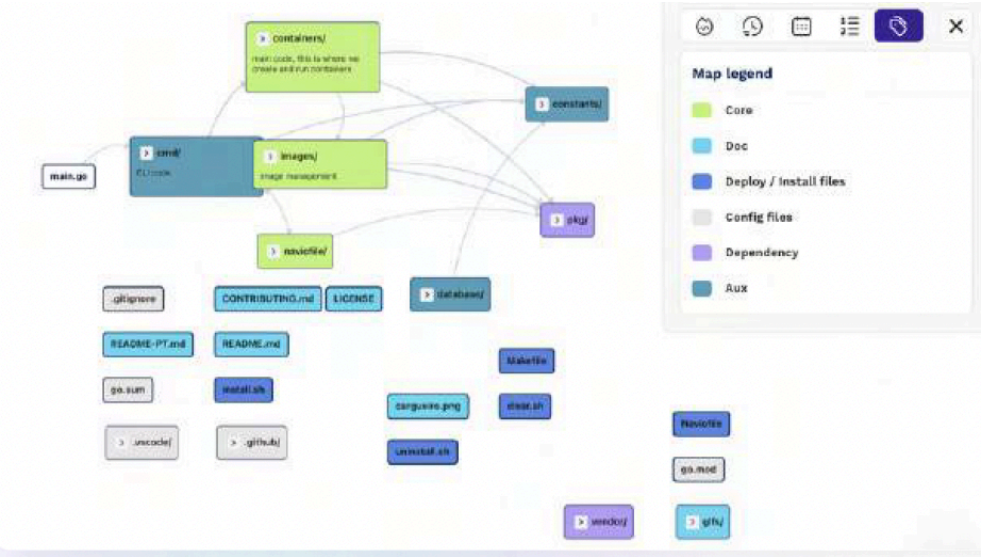
This requires only a 5-minute setup but can save hours (or days) of onboarding time further down the road. Let’s go through how you can set up a codebase map and tour on your own codebase to help new engineers onboard properly.

Customize Your Codebase Map

We won't go through the installation process here, you can see that in our docs. Here we're going to use the Navio project as an example. Let's say we're using this within our organization to create and manage linux containers and it is an integral part of our workflow. Any new engineer has to understand this codebase. What we want them to know immediately is

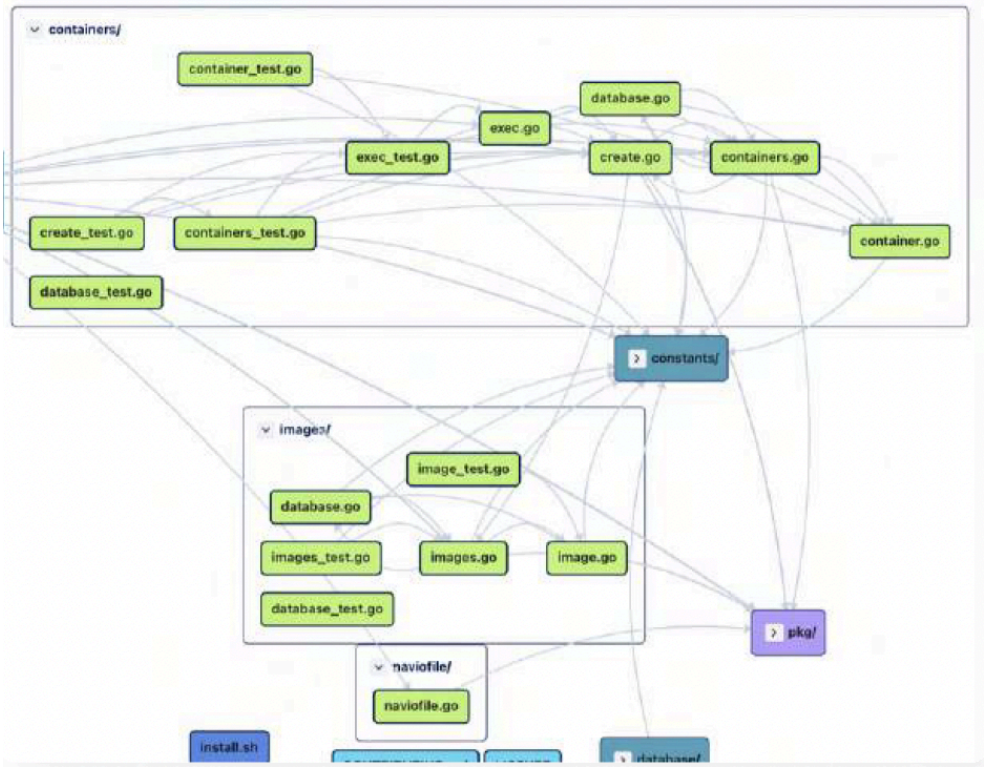
- Where does the main 'work' of the application get done?
- What imports does this main piece of code rely on?
- Where do testing files live?
- Where do configuration and package management files live?

Let's customize our codebase map to help highlight these factors:



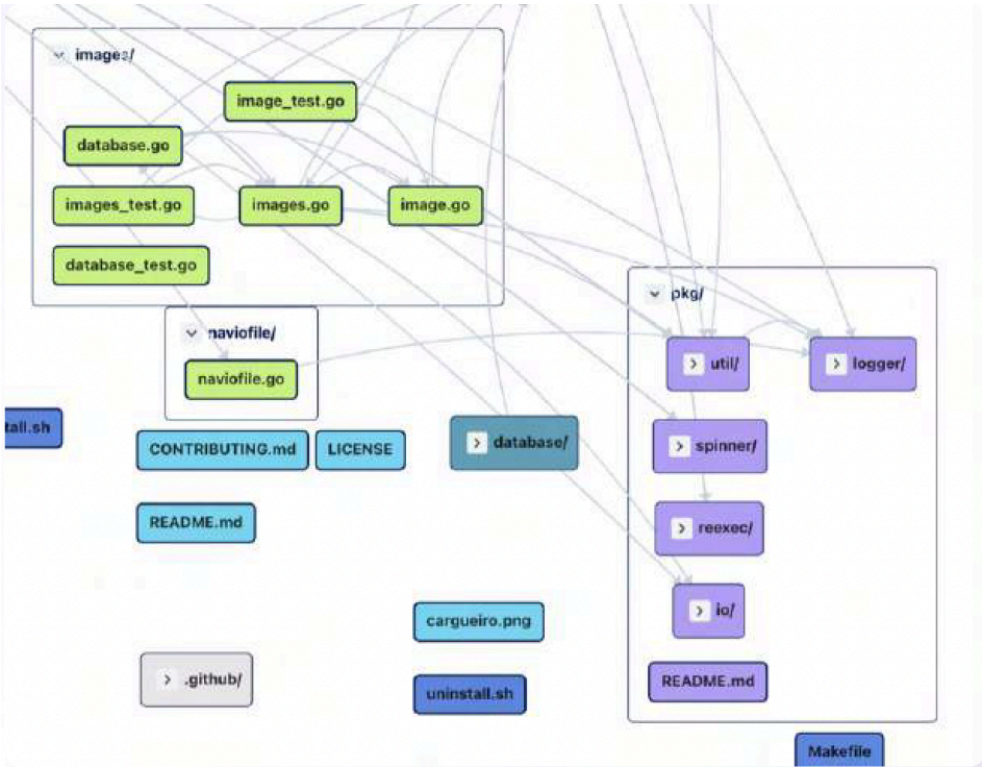
We can color-code our codebase map according to what an onboarding developer would need to know.

Where does the main 'work' of the application get done? The main 'work' is colored light green. The 'Core' modules are in containers/, images/, and naviofile/. Now the new developer knows these are where the fun happens and can explore from there, expanding these folders to see individual files:

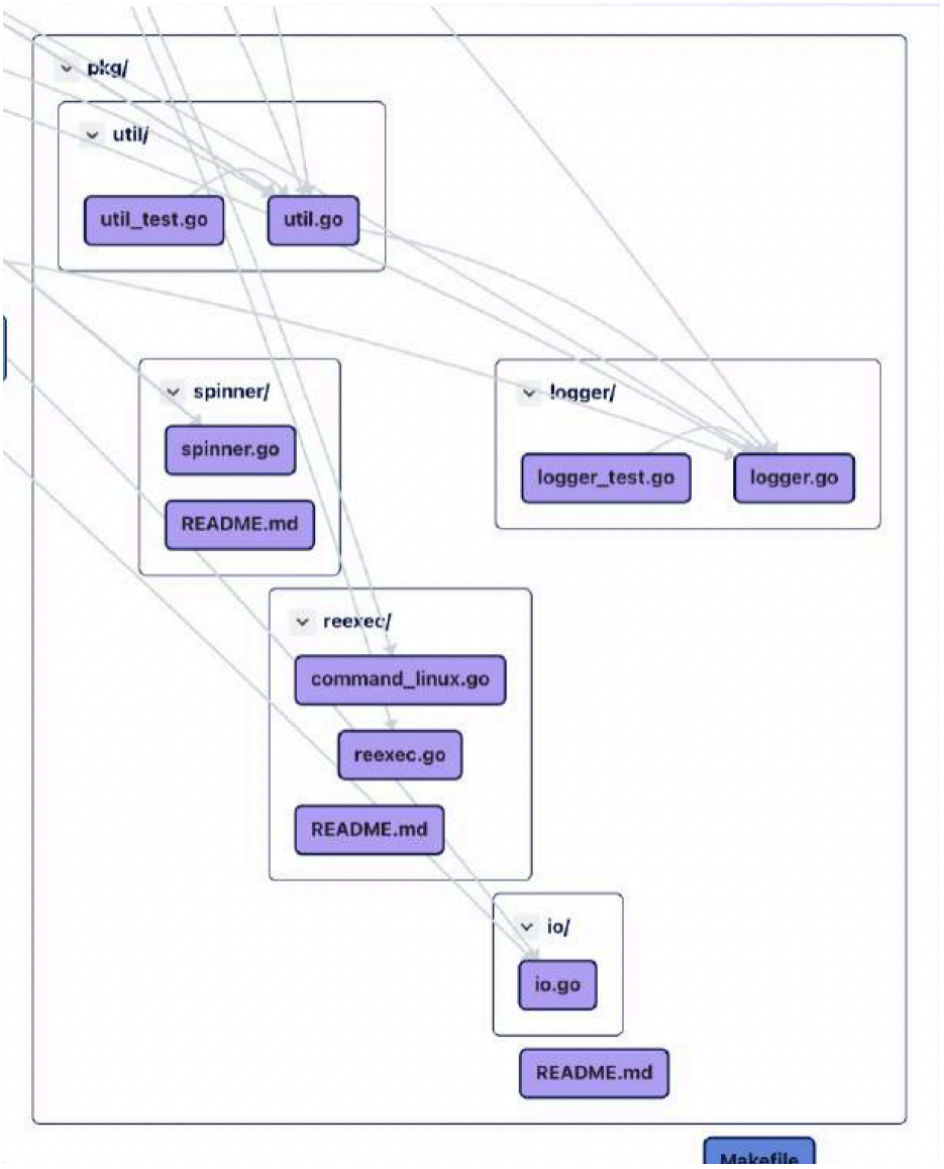


That's still a lot of files to understand, but at least the new developer knows these are the core. They can then delve into the documentation (colored light blue) to learn more.

What imports does this main piece of code rely on? Now we know the core files, what are the dependencies? Here, we've color-coded 'Dependency' as purple. That coloring, plus the arrows going from green to purple, make it easy to see how important pkg/ is to the core files. Again, we can expand this out to see what files are in the folder:



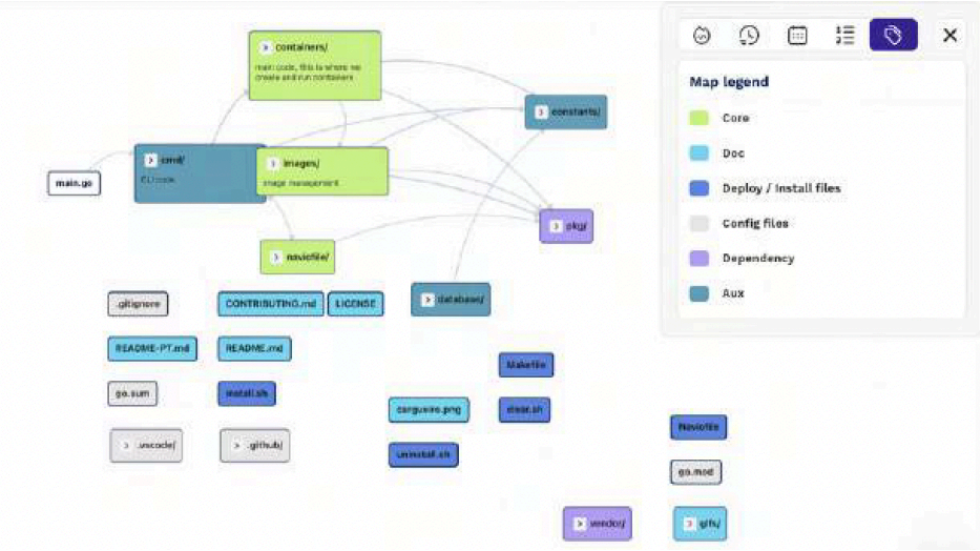
We've got some utils, a logger, and some io. Plus a README. Excellent news for the new developer. Wonder how long it would have taken them to find that otherwise? Let's open this up further to see more about these files:



Oh, awesome, we now also have the answer to our next question.

Where do testing files live? In this design, tests live next to files in the same folder– no separate test folder. Finally, we want to know,

Where do configuration and package management files live? We can go back to our map legend, see config files are colored gray, and see them in the root of the project:



We can see the go.mod file that manages dependencies and go.sum with the checksums for downloaded files. So just through thoughtful color-coding, we've helped an onboarding developer a) find the main files they need to know about to get started with the codebase, and b) understand their different functions within the codebase.

Create a Codebase Tour

This color mapping gives them the information they need, but they are still lacking guidance. For that we want to create a codebase tour.

What is an interactive codebase tour?

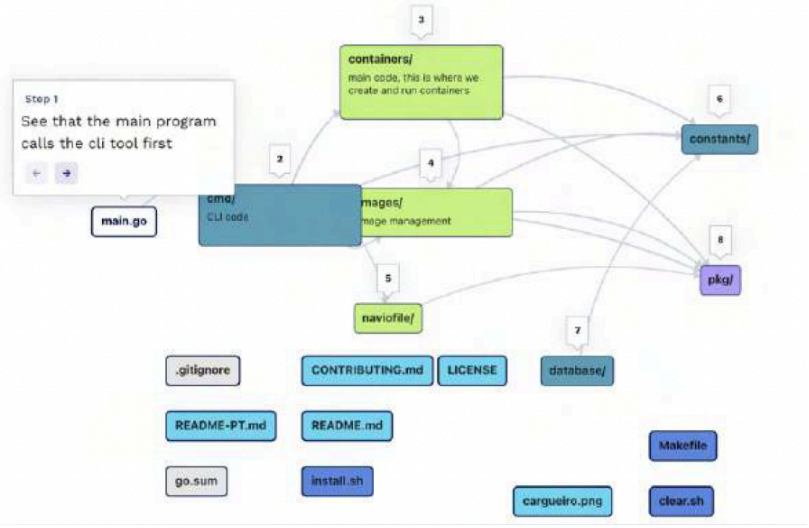
An interactive codebase tour is a guided tour of a codebase that walks a developer through the codebase step-by-step for a particular function. For instance, if you are the team lead on a code refactor, you could create a step-by-step tour through the codebase showing the elements you think need work. Or it could be something as simple as taking a new hire through how each component of the code interacts with others.

Why should I create a codebase tour?

Codebase tours help further the goals of a codebase map: **improved code understanding, increased collaboration, better code maintainability, faster bug fixing, reduced development time, and improved code reviews**. But they take it a step further. By providing a personalized experience, codebase tours can help developers to quickly and effectively understand the codebase in relation to the role they are performing.

In our scenario, a codebase tour might look different for an onboarding junior developer than an onboarding senior developer. Or it might look different for a new hire working on the frontend to the backend. Codebase tours let you guide a user through your codebase in a way that best facilitates their job, getting them the right information to be productive.

Here's a sample tour from our earlier example:



In this tour there are eight steps, taking the new developer logically through the program

- See that the main program calls the cli tool firs
- The cmd package contains several files where each one refers to a Navio command
- Here is the main code, where containers are created and execute
- Provides images for cmd or containers packag
- Create new Images using Naviofil
- Stores all application constant
- Create and configure the database. Creates the default image
- Collection of utility packages

You can see how this corresponds to how an experienced dev might take someone through the codebase in person. But here it's all laid out for anyone to see and use. Build once; use forever. This helps onboard new hires quicker, but also frees up time for the experienced members of the team as well.

Customize Your Codebase Tour

There are a huge array of ways to slice and dice your codebase tours. But here are three that we recommend to start:

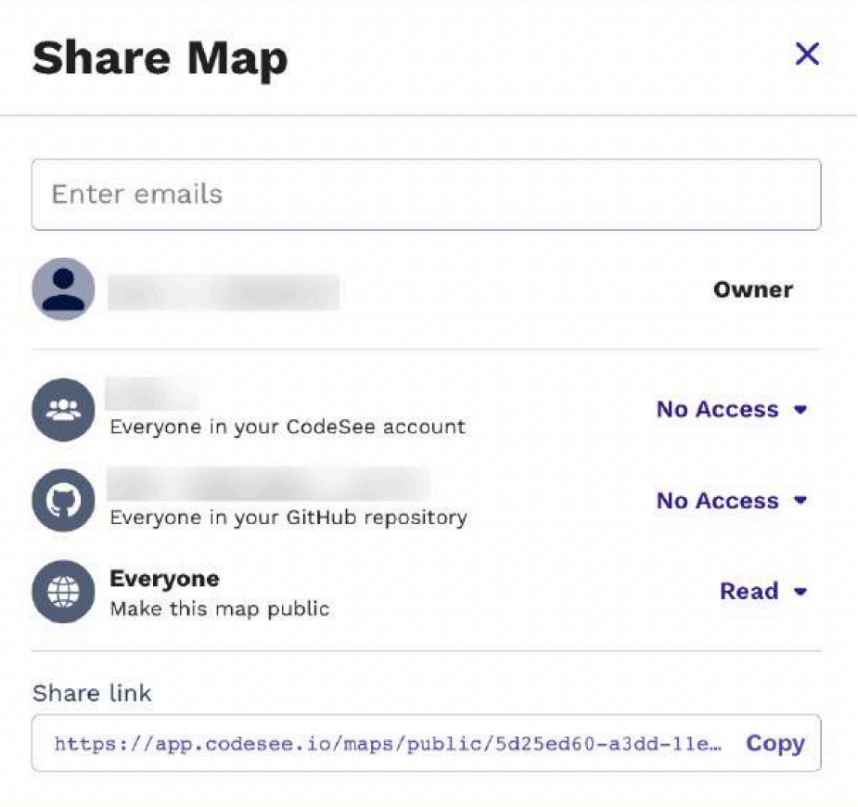
- **By team ownership.** Each team can set its own tours and describe the journey through the code that best fits the team's needs. How a frontend UX team sees a codebase is going to be different from how a backend API team sees a database. Tech leads on each team should be creating the right tours for their team members.

- **By developer level.** Junior developers need different information than senior developers. Throughout this guide we've been talking about onboarding. To use the Remix example from earlier, an experienced Remix developer already knows about routes and index.tsx. They don't need that within their tour. But a junior developer that might know JS but not Remix specifically will welcome an introduction to Remix as an initial tour.

- **By feature.** Tours can be as granular as needed, right down to individual tours for individual features. You can have a tour showing the entry point for a feature, the components it uses, all their dependencies, and how it uses data and API routes. This helps new engineers understand features fast, but can also be used by support engineers and PMs to help them understand internals, as well as helping senior developers reason about the code.

Share Your Map and Tour

Finally, once you've created your map and tour it's easy to share it. Head to 'Share' in the top-right corner of you map and then choose the best option for you:



You can choose to share it with:

- Select individuals through email
- Your team on your GitKraken account
- Everyone who has access to the particular GitHub repository
- Everyone by making the map public

Then you can share the link in Slack or wherever with your team.

Use Review Maps When You Make Code Changes

Visuals aren't just helpful as a single point in time. You'll want visuals throughout your development workflow, making onboarding seamless and more productive than ever. Download the GitKraken VSCode extension to get review maps right in your editor.

Encourage your dev team and contributors to use review maps whenever they make a complicated code change. Both of you can visualize how a code change impacts the rest of the codebase or more specifically if you made that change, what else is connected to it. Review Maps are like having an architect sitting next to you, telling you what's under the hood. This speeds up your onboarding.

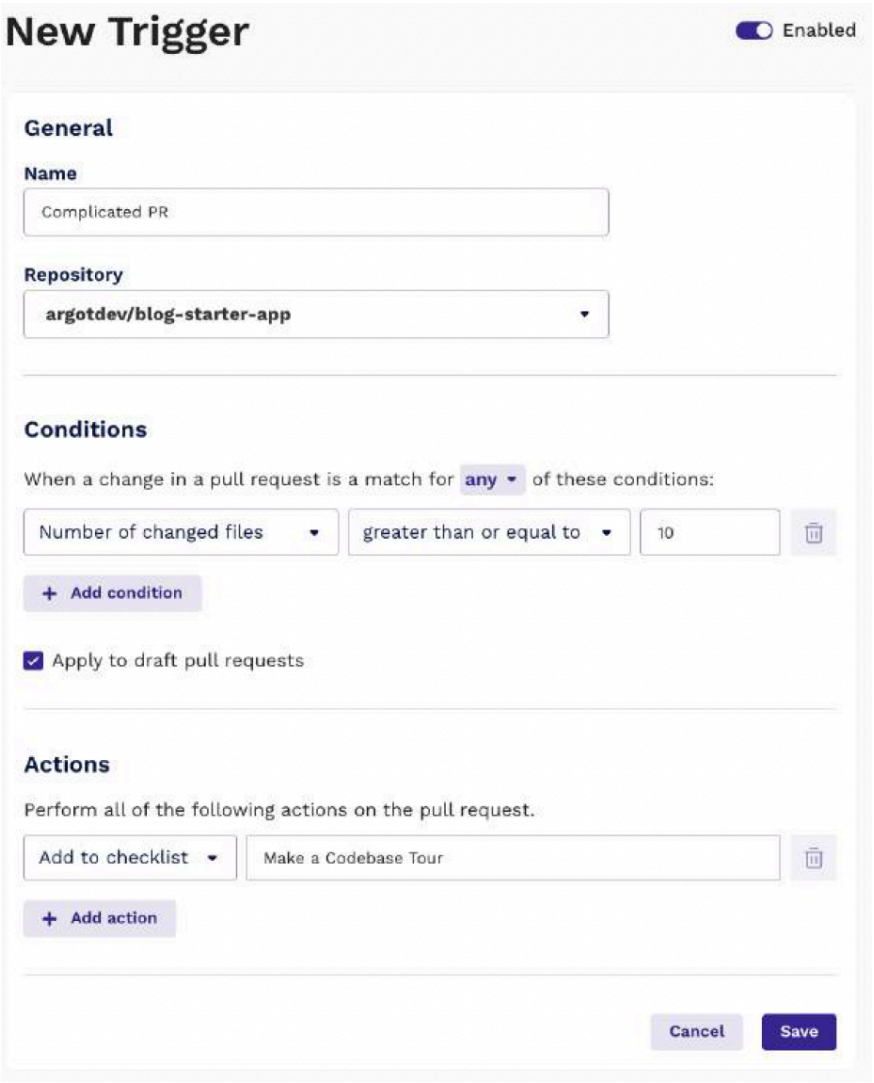
Write Code Automations to Share Organizational Knowledge

Another option for sharing organizational knowledge is code automations. These are set routines that fire anytime certain changes are made to the codebase. These triggers might be when a specific file is changed, when new code is added, or when a PR is opened.

You can then add actions:

- Add a comment to a pull request
- Add a checklist item to a pull request
- Assign a pull request to someone
- Assign someone as a reviewer

For new developers, this might take the form of an automation that would alert new developers that if their PR was complicated, they should make a tour:



So in this case, we:

- Set the Conditions as “Number of changed files” to “greater than or equal to” 10. So if the new developer is changing more than 10 files within their PR, this automation will be used.
- Set the Actions as “Add to checklist” an entry to Make a Codebase Tour.

When they are drafting their PR, they can see this checklist item and add a codebase tour for the reviewers. In this case, this isn’t just good onboarding, but accustoms the new developer to using tours to describe complicated processes throughout their time on the team. You can read more about code automations here.

Next Steps

Getting started with codebase maps and tours for onboarding developers to your codebases is a five-minute task

- Set up GitKraken and integrate it with your GitHub repository.
- Map out the different elements of your codebase so it's easy for newcomers to quickly see what's what.
- Add tour elements to your codebase to take a developer through the codebase step-by-step.

You don't have to be producing huge, multi-step tours initially, just something that anyone new to your team can follow so they can get the code running on their local machine and in a position to be effective. From there you can expand out to more mapping techniques and more tours to help new and old developers alike.