PROOPH BOARD WIKI

Welcome to the prooph board wiki!

Cross-functional teams, collaboration and experiments are key drivers to innovate and keep software and business aligned.

Development teams are asked to be multidisciplinary.

How can they tackle domain and tech complexity in an efficient and sustainable way?

How can they set goals, plan work, run experiments and serve user needs in the best possible way and all this within a lean workflow?

prooph board is the tool to support it all.

Storm-Design-Validate-Build Cycle

Our approach is based on an iterative software development process consisting of four distinct phases. 1 Brainstorming, 2. Design, 3. Validate, 4 Build

Getting Started

Brainstorm first events with our help:

1. Start with your first Event by answering the question: **How does your company make money?**
Double-click on the Event card below and write down the answer as an Event (Fact) using **past tense.** Examples:
SaaS product: "**Subscription made**"
Online-Shop: "**Product bought"**
Hotel Mgmt: "**Hotel room booked**"
2. Pull more Event cards from the sidebar (first card in top left corner)
3. What Event needs to happen before the one to the right?

Rinse and repeat the question and answer it up to the very first Event of your value chain.

Once completed, continue in the other direction and answer the question:

What Event will happen after the Event to the right?

That's all you need to know for your first Big Picture Event Storming.

If you're curious what else prooph board has to offer, head over to the Design & Prototyping board.

What’s your role?

We’ve put together a getting started guide for each possible role of a cross-functional team, because every role has a slightly different focus and will enjoy different features:

Free Version

The free version of prooph board offers a very easy way to get started. **No credit card and even no registration required.** It is a browser-only version of prooph board. All Cody features are available ✔️.

Since the free client has no server connection, cloud storage is disabled and your work is only stored in the browser’s local storage. Use import/export functions to backup your work!

Example Boards

Here is a list of example boards that open directly in the free client. Check them out for some inspiration!

First Events Guide
Design & Prototyping
Holistic, event-centric workflow
Event Storming - The Picture
Event Modeling Example

Cody Tutorial

🤖 Want to get started with Cody? These two coding bot introduction tutorials explain the basics.

- Node.js Code Generation Tutorial
- PHP Code Generation Tutorial

Powered by prooph software

We run **Event Storming & Modeling workshops** for product development teams — onsite or remote. **If you need support with Event Sourcing & Modeling, Event Driven Architecture, Domain Driven Design or Frontend applications**, we are available as a well-coordinated external team or join-force with your internal teams.

WHAT IS EVENT STORMING

💡 Event Storming is a workshop-based method to collaboratively discover business processes and user flows. You can apply it practically on any technical or business domain, especially those that are large and complex. The fact that Event Storming puts events at the heart of the method makes it a perfect tool for event-based system engineering.

Purpose

Event Storming comes in different flavours which differ mainly in the level of detail. A typical starting point is a **Big Picture Event Storming**. It is recommended to invite all stakeholders such as developers, domain experts, decision makers, UX designers etc. to collect viewpoints from each participant.

The group uses Domain Events written on sticky notes to explore the domain. By visualizing the entire business flow as a series of events, critical insights and discoveries are triggered and the foundation for an event-driven system is made.

11 Orange Stickys:

1. Parcel scanned at airport by agent
2. Parcel added to transport unit
3. Transport Unit completed
4. Transport unit booked on flight
5. Export customs information handed over to external broker
6. Export customs approved
7. Airplane departed
8. Import customs information handed over to external broker
9. Import customs approved
10. Airplane arrived at destination airport
11. Last mile provider picked up parcel at airport

This exploration and discovery process fosters knowledge sharing and a common understanding of the company's key activities, users and their needs, information & cash flows as well as bottlenecks in the value chain.

Onsite vs Remote

There is no doubt that Event Storming works best with everybody being in the same room.

However, we're a remote-first company and this motivated us to experiment with remote Event Storming in 2019. Early sessions on realtimeboard (now called Miro) were promising. But we quickly realized that a general purpose online whiteboard has limitations. As long as you only run Event Storming sessions occasionally, online tools like Miro and Mural work just fine. But they become limiting the more you want to use Event Storming for detailed day-to-day team interactions and event-based system engineering. We call this advanced usage Continuous Event Storming

This is where **prooph board** comes into play. Our goal is to provide the best possible remote Event Storming experience and give distributed teams a powerful tool to understand business problems and design high-quality software solutions.

BASIC CONCEPTS

The goal of Event Storming is to first gain an overview of the entire domain and then zoom into details to discuss opportunities and find bottlenecks. Colored sticky notes are used to describe business and information flows, user system interactions, and automated processes.

Event Storming defines a limited set of elements. This simplicity is a key reason for its success. The entry barrier is low so that all people can use the method without long explanations.

System:

There are two models the Read Model, which transfers information to an actor and the Write Model. This Model start with a decision from an actor based on Process “Whenever” and external Systems, this information is needed in order to make system decision. A State Transition happened somewhere, that maybe influences the process “Whenever”, if so it changes the decision from the actor, when it doesn’t effect the process “Whenever”, the Actor gets the new information from the Read Model, which has the information of the Write Model.

prooph board’s primary modelling elements (called Cards) are based on Event Storming concepts. Instead of simple colored sticky notes, Cards have a type assigned. This enables rich model analysis and conversion to source code. We’ll cover that later in detail.

💡 *Moderator Tip: First Event Storming Session? Introduce elements step by step. Start with Domain Events and explain further colors and concepts as needed. Try to avoid too much details in a Big Picture Event Storming. Getting an overview quickly is more important at the beginning.*

Domain Event

Probably the most important element in Event Storming is the Domain Event. It represents a fact — **something that has happened**. The corresponding color is **orange**.

You can describe every activity, data flow or process as a series of Domain Events e.g. *“User Browsed Online Shop” -> “Item Put Into Cart” -> “Order Placed” -> “Order Paid” -> “Order Shipped”*.

It’s this simple concept that makes Event Storming successful. A quick explanation how to formulate an Event and the Big Picture Event Storming session can start.

Use past tense in Domain Events. Put them in chronological order from left to write to tell a story. Ask questiosn like: What other events need to happen before this event?, Who or what causes this event? When does the event happen? …

Command

A command represents a decision that **something should happen**. The color code is **blue** It can be a decision of a user e.g. who wants to *Place an Order* or *Pay an Invoice* as well as a decision based on a Policy or made in an External System.

Usually, commands are named with a verb in imperative mood, present tense - followed by subject e.g. *“Start Engine”, “Buy House”, “Share Picture” …*

Commands cause events. Often, you have a 1:1 relationship between them: *Pay Order -> Order Paid, Book Room -> Room Booked*. However, one command can cause one, multiple or no events. It depends on the business rules.

Actor

Actor cards are used to highlight that users with a certain **role** are involved in a business process or use a specific UI to make decisions. The color code is **lemon yellow** and the card is in portrait.

Write the user role on the card and put it next to a UI or Command card.

Aggregate

The term aggregate is borrowed from Domain-Driven Design. In Event Storming it’s basically a business rule that makes sure a Command can be executed e.g. *An order can only be shipped if it is paid*. Write down such rules on **sunny yellow** cards and put them between a command and one ore more events.

Blue Sticky: “Ship Order”, points to a lemon yellow sticky: “Order can ony be shipped if it is paid!”, that stick point to 2 orange Stickys. 1: “Order Shipped”, 2: Order Payment Deadline expired”.

Information

Green cards represent information needed to make a decision. Information are usually displayed in a UI, but can also be in an excel sheet, PDF document or printed out on paper. External Systems fetch information via API or from a database to make decisions. All these cases can be visualized with green Information Cards in combination with others.

Every information change should be described as a Domain Event. **If there was no event, the information did not change!**

Developers familiar with the architecture pattern CQRS should also know the term “Read Model”. Both terms can be used interchangeably.

Policy

A Policy is **a reaction to an event**. It represents automated decision and coordination logic. Simply said: *If this, then that!*. **Lilac** Cards are used for policies. They trigger new commands.

A good example of a process with policies involved is a reservation. Imagine a ticketing system of an airline. While booking a ticket, one can usually select a free seat in the airplane. A resulting event could be *Seat Reserved*. As soon as a Policy receives the final event *Flight Ticket Bought*, it can trigger a command to change the *reserved seat* into a *booked seat*. This should happen within a fixed timeframe, let’s say 15 minutes. If the potential passenger does not finish the booking process, the Policy will trigger a command to change the seat back to *available*.

Hot Spot

No Event Storming without Hot Spots, Hots Spots can include Questions that can’t be answered now or Action items or Problematic topics. If you don’t have a couple of **red** Cards – Hot Spots - on the board after a Big Picture Event Storming, you did not dig deep enough. Just kidding, but a Hot Spot is the swiss army knife for an Event Storming facilitator.

Use Hot Spots when:

A question came up that nobody can answer right now -> Write it on a Hot Spot and put it close to the elements on the board that triggered the question.

A discussion between two experts becomes heated or drifts into details so that no one else can follow anymore -> Stop the discussion, write the topic on a hot spot with a note that it should be revisited in a follow up.

Someone points out a pain point in a process or describes something as a bottleneck -> Write it down and mark the area on the board with the Hot Spot.

External System

A third-party system involved in a business process can be visualized with a **dusky pink** Card. Digitizing workflows often means connecting and orchestrating different digital services. Event Storming is a lightweight method to plan and document interactions between your system and external services.

UI / API

The original color code for UI stickies is white, but on prooph board all cards are white with a colored top bar. So we decided to invert the color and use **black** for UI Cards. We also extended its meaning to include APIs. APIs as well as UIs are public interfaces. One for humans the other one for machines. To keep the color code simple, black Cards combine both - human and machine interfaces - to a unified **Interface Card Type**.

Use icons on UI Cards to roughly sketch what is visible on the screen. Just enough to support the idea. Don’t waste your time with too much details. prooph board is not a wireframe tool. UX / UI designers have their specialised tools for this task. However, screenshots of wireframes / existing UIs are a quick and easy way to include more detailed designs in Event Storming sessions.

PREPARE BIG PICTURE SESSION

In this chapter you'll learn how to organize and run your first Big Picture Event Storming on prooph board.

Recorded Session

This recorded Big Picture Event Storming session gives you some insights about the format:

https://youtu.be/vwuSDCRghb8

Plan Session(s)

Ok, you decided to organize your first Event Storming. That's great 👏. Welcome to the club of Event Stormers. You won't regret it! Fortunately, there is not much upfront planning needed. However, it's important to set expectations correctly and start small. A few pitfalls should be avoided to not risk success.

Goal of the first session should be to a) get familiar with Event Storming itself and b) create a shared understanding of the domain or the area of the business that is of interest for you.

Avoid goals like: After the session we should be able to derive stories that we can work on next. Such goals often require detail sessions. If you set them too early - maybe to convince a manager to spent time and money - you could end up with unmet expectations and unhappy attendees or supporters.

👥 Make sure to have a facilitator available. You need someone with Event Storming experience and moderator skills. It's possible to run out of time or end up in heated discussions, if you don't have someone who takes care of that. We can help you find a facilitator. Just get in touch. Once you're familiar with the method, a team member can take over the facilitator role.

⏱️ Split a remote Big Picture workshop into at least two days. Each day not longer than 3 hours. Also plan enough breaks e.g. 10 min after every 50 minutes. Remote collaboration is fun but also draining! Keep that in mind. Nice side effect is, that you can give homework after the first day. At the end of day one ask questions like: *Are the events correct?, Did we forget something?, Should we change the current structure?*. Start day two with the same questions and see what new insights people got overnight.

Invite The Right People

Event Storming works best if attendees represent different areas of expertise: developers, product manager, business experts, UX, decision makers, you name it. The list highly depends on context. But it's important to stress the point that Event Storming can only unfold its full potential when not only developers do it. Otherwise you'll end up with the same problems that you likely want to solve.

6-10 people is a good number for a remote workshop. If more people want to join (or have to), you definitely need more than one facilitator. In this case plan grouping with breakout rooms in advance.

We're constantly working on improving the remote experience for larger groups on prooph board. An integrated, interactive video avatar feature is currently in beta test phase and will be rolled out soon. It enables seamless grouping and parallel discussions on the same modeling workspace. If you would like to try it and can't wait until official release: **Contact Us**

Prepare Board Workspace

First of all you need to create a new board in prooph board. After sign in you can directly do this on the dashboard. Click "Add Board" to create a new one and give it a name. You're redirected to the new, empty board. It's ready to be prepared for the workshop session.

💡 People can be invited using the "share" button in the top bar. Enter their email addresses and they'll receive an invitation.

It's not yet possible to use the free client for remote collaboration workshops. You need a paid subscription to be able to invite other people. They also have to create an account but don't need to subscribe to paid plan, too. Learn more in **Managing an Organization**

For the session itself you don't need to prepare much. Include the link to the basic concepts: https://wiki.prooph-board.com/event_storming/basic-concepts.html in the meeting invitation, so that people can study it upfront. Also communicate the goals and expectations of the session.

If you want to prepare some supportive areas (recommended) like an overview of the agenda or a Q&A area, you can make use of Frames. Here is an example:

https://wiki.prooph-board.com/assets/images/big_picture_prepare_board.gif

Frames help structuring the modelling space and allow for quick navigation using the **Tree View** in the left sidebar.

Ok everything is prepared. The session can start!

Explain Basic Concepts

At the beginning of each workshop it is useful to give an introduction. *What are we going to do today? What are our goals and expectations? Who is in the room?* Then the facilitator should quickly explain what Event Storming is and how to work with Domain Events.

It is not recommended to explain all Event Storming concepts in one go, but rather explain them as needed. It saves time and reduces cognitive load. You usually don’t need all Card types in a Big Picture Event Storming anyway. Explain the concept of Domain Events and just start with that.

First Event

The first event is the most difficult one. Everybody is still unsure what to do and how it works. But don’t worry, this changes quickly. A good starting point is to ask about the goals of the company/business unit. *What user needs do we serve? How do we make money?*

https://wiki.prooph-board.com/assets/images/Big_Picture/First_Event.png

Chaotic Mode

After the first event is on the board (the icebreaker event), you can ask every attendee to write down more events. This happens in parallel. Everybody should think about questions like: *What happens before the event? What comes after? What events are important for the company and our users?* Ordering is not important at this stage. Just eventstorm whatever comes to mind and put it somewhere. 10 - 15 minutes is a good timeframe for chaotic mode.

Use prooph boards integrated timer to keep an eye on the clock. One can start a timer by clicking on the **hour glass** icon in the top bar. Set a time, press enter and the timer will start to tick. It is synchronized with all attendees. So everybody can see how much time is left.

https://wiki.prooph-board.com/assets/images/Big_Picture/Chaotic_Mode.png

Tell a Story

Now that we have a couple of high level Domain Events on the board, it’s time to order them chronologically. One person takes over the driver seat (does the ordering) and constructs a story. Switch driver seat every 5 - 10 minutes to keep people engaged. Talking about the process (the story) will trigger discussions. Better events are suggested, wrong or duplicate events get removed and more detailed events added.

https://wiki.prooph-board.com/assets/images/Big_Picture/timeline.png

Group By Context

❗ Keep an eye on the planned breaks. Remote collaboration is fun, but also draining. People need breaks to get some fresh air or a new coffee/tea.

The following steps can be mixed as needed. It’s the facilitator’s task to coordinate work. Meaning, it’s likely that you’ll jump between context grouping and domain exploration and identify Hot Spots along the way.

Group events that belong together and try to find a name for the group. Again, someone should be in the driver seat, but change them every few minutes.

https://wiki.prooph-board.com/assets/images/Big_Picture/Grouping.png

If you’re a large group of participants, now is also a good moment to split into working groups. Each working group can explore one or more event groups.

Explore The Domain

It’s time to introduce more Event Storming concepts like Information, Actors, External Systems and Commands.

Zoom in by asking questions like *What needs to happen here exactly?* and use examples with personas and concrete data written on the stickies. This deep dive will unfold a lot of new information and knowledge.

https://wiki.prooph-board.com/assets/images/Big_Picture/Explore_Domain.png

This type of exploration needs a little bit practice. How much detail is too much for a Big Picture? When should we stop with the help of a Hot Spot and continue exploration in a follow up with a smaller group? Don’t worry, you’ll get better finding the right level of detail quickly.

Identify Hot Spots

Hot Spots are useful during the entire workshop. But still, if exploration get stuck a little you can push it forward again with explicit questions about bottlenecks or opportunities for improvements: *Is there an area where we could do better? What do our users feel about this user flow here? Should we look at this from a legal perspective? …*

https://wiki.prooph-board.com/assets/images/Big_Picture/Hot_Spots.png

Retro

At the end of each day you should do a short retro. You can prepare 2-3 Frames with questions: *What do you like about Event Storming? What don't you like about Event Storming? Do you have open questions? Could the workshop be improved next time?* Ask participants to write down their answers on Cards and put them into the respective Frames. 5 minutes should be enough. Remember using prooph board's timer! Then go through the answers and let everybody explain their thoughts.

Feedback is always very important. It helps to learn and grow and give people a good feeling about the next steps.

What's next?

We hope you enjoyed your first Event Storming, and it was a success! Big Picture sessions are already very valuable, and you can repeat them or continue with specific areas of the value chain, zoom into sub processes and do detailed sessions. Every Event Storming will sharpen your knowledge of the domain and create a common understanding of the things going on in the company.

But you can even take it one step further and integrate Event Storming and Modeling techniques into daily work to improve requirements engineering, collaboration, story writing and task breakdowns, documentation and even code assistance. How it works is covered in the next chapter: Event Modeling.

WHY EVENT MODELING

Event-centric design and software modeling. Product teams can use Event Modeling on a daily basis – seamlessly integrated into the development workflow.

Event Modeling provides a list of great benefits:

- One tool for the entire team to discuss and design software
- Event-driven Architecture design made easy
- System documentation
- Cross-Team collaboration
- API contract negotiation and documentation between teams
- Reliable work estimation
- Flat cost curve
- Modular systems

Originally invented by Adam Dymitruk, Event Modeling is used by teams around the globe. Check out the official website eventmodeling.org.

On prooph board, we use our own interpretation of Event Modeling that is optimized for continuous integration.

Before we dive in, let's have a quick review of how software development evolved and why there is still room for **improophment™**.

Waterfall

The classical approach to software development is throwing requirements specifications over the wall to a group of developers. Developers then take those requirements, interpret them and write code that they think does exactly what was asked for. Sounds simple, but we all know it isn't.

https://wiki.prooph-board.com/assets/images/Waterfall.png

This is also known as the "Stille-Post Effect". Here is a LinkedIn post describing the effect:

Agile Methods

Agile methods like Scrum aim to solve the problem by focusing more on the product and allowing only a rough roadmap. Precise requirements (e.g. problem briefs) are defined for one or two sprints only. That makes sense, because requirements are never perfect. There are just too much unknown unknowns along the way. So agile methods have improved the situation a lot. Unfortunately, in many cases results are still not as expected. Development takes too long, users are not satisfied, the software is buggy.

Why this? Well, agile methods alone are only half of the story. A common mistake is that the different roles of an agile team still work on their own. So the PO is doing requirements engineering, UX is designing and developers are developing.

https://wiki.prooph-board.com/assets/images/Scrum.png

Each role uses specific tools, a different mindset and a different language. They try to work together as a team, but they often don't really talk about the same thing. We've seen many story writing and task preparation sessions that caused more confusion than actually clarified things.

Enter Cross-Functional Teams

The goal of prooph board is to enable a product-driven and outcome oriented way of working that includes the entire team into the software design process.

https://wiki.prooph-board.com/assets/images/Cross-functional.png

Only if business experts, UX designers (even end users), product managers and tech people work closely together and have in depth discussions the problem can be solved.

prooph board provides a way for cross-functional teams to design software together, even if not all people know how to code. An abstract model helps to communicate on a level that is easy to understand on the one hand but still powerful enough to be a blueprint for software solutions.

Visualizing problems and solutions, sketching different ideas and playing with examples is much more fun and way more effective than reading through problem briefs and writing stories.

prooph board uses ideas from **Event Storming** and **Event Modeling** to let a cross-functional team design an abstract model. The cherry on top is our coding bot **Cody** that becomes a hidden team member and provides assistance for translation of an abstract model into working software.

In the next chapter, we explain how to integrate prooph board into the development workflow.

If you first want to know why Event Modeling is such a great technique, you can take a look at our in-depth philosophy behind the approach.

PHILOSOPHY

You might ask yourself why you should learn Event Modeling? And is it really worth the effort to integrate prooph board into your development workflow? Maybe even change the system architecture?

This page provides you with some fundamental thoughts of the technique. It should help you understand where we are coming from and where we are heading to. If you want to join us on the journey is up to you :).

Change is the root of time

"Nothing happens until something moves."

– Albert Einstein

Since Einstein we know that time is relative. It moves relative to the observer. This also means, that time is not a given. The clock is not ticking out of nowhere. Time is basically a measurement between two **events**.

Without events time would not exist, because if nothing changes you have nothing to measure.

Event equals Information changed over time

We perceive our environment through events. If we see something, it already happened. The picture in our head is a picture of the past. As an example, if you look at a star in the sky at night you cannot really know if the star still exists. The light of the star needs a lot of light years to reach the earth and finally your eyes.

When we see, hear, smell, touch or taste something we basically process information. The difference between one set of information and another set of information makes our reality. Our senses process information changes that already happened aka **events**.

💡 **Events are at the core of our reality.**

Why does it matter?

When it comes to traditional software design, people tend to put a lot of focus on things and how they relate to each other. In other words, they focus on specific sets of information and how those sets are connected. That itself is not a bad idea. However, the order is wrong.

💡 **Information is always the result of one or more events.**

If you design an information system without putting events at the core, you're doomed. You're ignoring the fundamental building blocks of our reality or at least give them not enough attention. This results in so-called accidental complexity. At the beginning of a new system everything looks clear and simple. But you continue to add more and more information changes and at some point you loose control over it. Without proper events, you cannot tell **why something happened**. Yesterday the system was all good then someone did something, and now it's broken, but you cannot perceive what happened, because you missed the events.

Observability is very important as soon as the system reaches a certain level of complexity.

Now one could argue: we have logs and logging tables for observability. We also have Mixpanel or Google Analytics in the frontend to track user behavior. So we know what happened.

Well, that's maybe true. But again, the order is wrong. Usually logs in whatever way are an afterthought. They are not part of the design process, and they are not at the heart of the system. It's an additional layer instead of being the foundation. Which basically reduces efficiency, clarity and alignment.

Patterns embrace efficiency

A pattern is a regularity in the world, in human-made design, or in abstract ideas. As such, the elements of a pattern repeat in a predictable manner.

Let's look at the pattern of what we've discussed so far.

Basic Pattern

https://wiki.prooph-board.com/assets/images/philosophy/basic-pattern.png

You have Information. Changing the Information is an Event at a specific point in time. That gives you new Information, which can be changed again and so on. No matter how complex a system is as a whole, a single unit of work should always be designed around that simple pattern.

And since events are at the core you can even simplify the pattern to this:

https://wiki.prooph-board.com/assets/images/philosophy/simplified-basic-pattern.png

Without an Event you cannot tell why Information is what it is at a specific point in time. It's just a snapshot like one frame of a movie or a thumbnail of YouTube video.

A desired Change is also incomplete without an Event, because you cannot tell if the Change really happened. If you have an older snapshot of the Information, you can try it by comparing the old snapshot with the new one, but still you cannot be 100% sure that the desired Change was actually the one that caused the Information to change.

Contrary to that, by looking at an Event you can say:

1. There was a desired Change, and it really happened because you can see the Event.
2. The Event produced a new set of Information.

So you can purely focus on events and design everything else around them. And that's the reason why a Big Picture Event Storming should always start with collecting events. It is one fundamental concept, and it is very easy to grasp. It provides you with everything you need to discuss behavior of a system, and it will be the same at any other scale or detail level.

Now lets look at a few different scales.

With User Interface

https://wiki.prooph-board.com/assets/images/philosophy/pattern-with-ui.png

Information is viewed by a user in a user interface. The interface provides the user with an option to tell the system: "Please change the Information in a particular way!". If and only if the system processes the desired Change a new Event has been occurred and the user can view the changed Information in the User Interface again.

With Digitized Rules

https://wiki.prooph-board.com/assets/images/philosophy/pattern-with-rules.png

Good software systems don't blindly trust the user. Usually they have business knowledge incorporated to protect invariants. What the user wants to change is one thing, what they are allowed to change is a different story. The best place for such rules is between a desired Change and an Event. If the Change is not accepted by the system, it is either rejected (so no Event happened at all) or a different Event occurs.

With Digitized Reaction

https://wiki.prooph-board.com/assets/images/philosophy/pattern-with-digitized-reaction.png

To increase efficiency of a process or be able to scale it, manual steps made by humans are often replaced by some kind of automation. Having events in place, that’s a straight forward task. Process automation can be designed and implemented as a reaction to events.

If this Event happened, do that Change next.

Unit of Work

https://wiki.prooph-board.com/assets/images/philosophy/pattern-with-function.png

Processing a desired Change can be described as a function of the system – a unit of work.

Function Composition

A system composes a set of functions while acting as one “big function” to the outside world.

https://wiki.prooph-board.com/assets/images/philosophy/pattern-with-system-detailed.png

Or the Big Picture view of the same system:

https://wiki.prooph-board.com/assets/images/philosophy/pattern-with-system-simplified.png

At an even higher level a system of systems can be composed of multiple sub systems and so on.

You see the repeating pattern? **Composition is the only solid way to tackle complexity.**

The Pattern is everywhere

Developing software is a multidisciplinary activity best done by a team of experts in their respective fields. A typical list of disciplines includes (sometimes more, sometimes less):

- research
- design
- knowledge crunching
- implementation
- testing
- documentation

- monitoring
- reporting
- maintenance
- support
- scaling

All this activities have to be coordinated. People with different attitude and aptitude have to work together. Successful collaboration requires very good communication. Everybody needs to be on the same page. Everybody should talk about the same things using the same wording. Everybody should understand the work of every other team member to some extent. This does not mean, that everybody needs to be an expert in every field, but there should be some kind of overlapping understanding.

If you've read this page carefully and understand the implications, you might have an "Aha moment" now. When events are at the core of reality, and they are also at the core of the behavior of a system, then they can also be at the core of any activity related to working on a system. They cannot only be at the core, they actually are at the core! You just have to make the implicit explicit.

Collaboration based on Event Modeling and a system architecture based on events does exactly that. If you want to learn more head over to the next page: How To

HOW TO

Here you find suggestions on how to integrate Event Modeling into your existing development process. Keep in mind that every team is different so should be their processes. Iterate on the format until you feel comfortable.

Storm-Design-Validate-Build Cycle

Our approach is based on an iterative software development process consisting of four distinct phases.

https://wiki.prooph-board.com/assets/images/event-modeling/SDVB-Cycle.png

Phase I: Big Picture Event Storming

Brainstorm ideas, collect requirements and user needs, get the Big Picture.

How

This phase is best done as a workshop together with domain experts, key users, and basically everybody who has a stake in the software.

Work Mode

Moderated collaboration in workshop-like sessions. Depending on the topic, the number of sessions can vary from one to many.

Estimating

Plan the sessions ahead. Be aware that all involved people have a though schedule. Prepare yourself for making the sessions as efficient as possible.

Phase II: Event Modeling Design

Dive into the details, create a story plot, design the information flow.

How

Take the learnings from phase I and sketch a software design using Event Modeling on prooph board.

Work Mode

Highly creative phase. Put on the explorer hat. Take your time for research, deep discussions, and feedback. Iterate on the design. Sketch multiple models, and try them in Cody Play.

Estimating

Creative work is hard to estimate. Define timeboxes and investigation tasks. Negotiate with stakeholders how much time/money is worth to spend on the topic at hand.

Phase III: Prototype Validation

Perform a completeness check.

How

Validate the design in the Cody Play prototyping sandbox. Go through different scenarios with high quality test data. Check that information flows smoothly through the system, and the user journey matches with the sketched story plot.

Estimating

Set a fixed timebox depending on the size of the topic. Use the number of Event Modeling Slices as an indicator for how long the timebox should be.

Phase IV: Software Build

Develop, test, and deploy the software (or a part of it).

How

Build the software using the Event Modeling Design as a blueprint. If you’re using Cody Engine, you can let Cody generate source code that behaves like the Cody Play prototype. Then customize the software to satisfy quality and usability standards of your org.

Estimating

Event Modeling emphasis a design made of little bricks (called Slices), that can be plugged together to form a software. On average, each Slice will take the same time to implement. Over time, you’ll get a very good feeling for how long you need for a single slice. So you can just count the Slices and multiply them with your average time.

Development Process Integration

The SDVB Cycle can be integrated in any software development process. If you work with Scrum sprints, you have small iterations, and maybe do Phase I only for new epics. Shape Up also works great in combination with SDVB. You can set up a dual track, where part of the team is doing Phase II and Phase III, while the other part is doing Phase IV. Mix and match how you see fit. Just make sure that all four phases are represented in your specific workflow.

> Each team should have its own board within the organization. prooph board ships with advanced features to link and synchronize information across boards, which makes it easy to create high level system and process overviews.

Team Collaboration

We suggest having regular collaboration sessions going from one phase to the next.

💡 *Pro Tip:* Divide the team board into two areas. The first one focuses on the “Problem Space”. So in this area, anyone can bring up a problem or idea that the team should tackle in one of the next iterations. Often this is done by the product owner, but in a highly motivated, fast-acting team anyone will contribute. Ideas, problems, and user needs should first be discussed on a high level to get everybody on the same page and decide if the team wants to invest more time. Quickly sketch flows with Domain Events and other card types as you see fit. Mix and match it with formats like Example Mapping or Domain Storytelling (by using icons + arrows on prooph board).

To integrate UI Mocks in the collaboration you can either upload screenshots or combine UI Cards with icons and text to simulate **Fat marker sketches**.

https://wiki.prooph-board.com/assets/images/CES/fat_marker_sketch.gif

Managing Hot Spots

When collaborating on prooph board you'll soon need to find a good way to manage Hot Spots.

We love Hot Spots. They are the best tool in a collaboration session to keep focus. It's easy to zoom into process details or spontaneously discuss ideas with Event Storming. But at some point you'll need to stop to not lose focus on the main session topic. Hot Spots are made for such situations:

- Discovered a new bottleneck -> write it on a Hot Spot
- Found a promising new idea -> write it on a Hot Spot
- Need to address a bug in the system -> write it on a Hot Spot

A couple of sessions later your prooph board will be full of Hot Spots. That's normal. It always happens. People get the idea of Hot Spots really quick.

But a Hot Spot is only as good as the action item derived from it! So how to distinguish between urgent Hot Spots that need to be resolved as soon as possible and for example ideas for later?

To categorize Hot Spots you can use Card Tags. We assign the tag **#later** to all Hot Spots containing ideas. Tagging Hot Spots allows you to filter them in the tree view. You can search for tags in combination with card types and also use a “NOT” operator to look up all Hot Spots that don't have a specific tag assigned.

https://wiki.prooph-board.com/assets/images/CES/hot_spot_tagging.gif

Design-first Approach

Once the team decided to invest time into an idea or user need, they should design a solution together. If needed, give UX/UI a couple of days to prepare wireframes.

Usually, some questions (Hot Spots) also need to be clarified before diving into design details.

Everything prepared? Awesome! Then it's time to meet on prooph board for some Event Modeling. You are going to describe the solution in all its details from data structures to UI flows and business rules.

We've prepared a Step-by-Step Guide for you.

EVENT MODELING ON PROOPH BOARD

This page provides a step-by-step guide of how to design an event-driven system using Event Modeling on prooph board. The resulting artifact is called **Event Map**.

Throughout the guide we'll use an example from a car rental business.

Start from an Idea

Let's assume we are a car rental startup team that finished its first Big Picture Event Storming.

The first module should be some kind of **Fleet Management**. We need to be able to **Add a Car** to the fleet and **Create a Rental Offer** for it so that it can be **booked by customers**.

https://wiki.prooph-board.com/assets/images/CES/EventMap/FM_First_Features.png

An Event Modeling Design session will help us with the details. To get the ball rolling, we look at each event from above and think of it as a Slice of the new system.

Where - Identify the Entry Point

Adding a car to the fleet will be a manual task done by the fleet manager. In later iterations we can think about batch imports and derive car details from a third-party service, but for now we keep things simple. This ensures a fast feedback loop.

So we start with a UI Card to sketch a rough idea of a UI screen showing a list of cars in the fleet and a button to add a new car.

https://wiki.prooph-board.com/assets/images/CES/EventMap/Fleet_Overview_Add_Car.png

Instead of using UI cards, some teams prefer a more visual approach by uploading screenshots from wireframes (especially when UX is part of the team). That’s totally fine. The only advice is that those wireframes should not be too detailed as this can become a bottleneck in the flow.

Who - Identify the Actor

A Slice either represents an action that a user can perform in the system or an automated process step.

As a Fleet Manager
I want to add a car to the fleet
so that it can be booked later.

Pull a new Slice from the sidebar. Each Slice has one or more user lanes with an Actor Card as a label. Put the UI card into a fleet manager lane.

https://wiki.prooph-board.com/assets/images/event-modeling/fleet-overview-actor-lane.png

What - Identify the Intention

The Fleet Manager wants to **add a car** to the fleet. This intention is represented as a Command Card placed in the API lane of the Slice.

When the system accepts and processes the command an event occurs, in our case the **Car Added** Event. Events are placed in the module lane of the system module where they occur. Here, this is the **Fleet Management** module.

https://wiki.prooph-board.com/assets/images/event-modeling/fleet-overview-add-car-command.png

Input - Which Information

As a next step, we should define which information is needed for the command to be accepted by the system. In a collaboration session the best way to do that is by using an Information Card per field or property. This allows all participants to contribute by adding cards in parallel.

Input - Which Information

As a next step, we should define which information is needed for the command to be accepted by the system. In a collaboration session the best way to do that is by using an Information Card per field or property. This allows all participants to contribute by adding cards in parallel.

https://wiki.prooph-board.com/assets/images/event-modeling/fleet-overview-add-car-input.png

After a collaboration session you might want to clean up the Event Map and structure such information in the **Card Metadata**

How - Command Handling

Some service or function in the system needs to handle the command. And for each one you should ask: **Do we need to check some rules when handling the command?** If the question is answered with yes, write down the rules on a Business Rules Card and place it between command and event in the same module lane as the event.

https://wiki.prooph-board.com/assets/images/event-modeling/fleet-overview-add-car-rules.png

Output - How does State change

Handling a command usually results in some changed state in the system. The state change is represented by an Event Card (like **Car Added** here) and the new/changed state (or the new/changed information) can be visualized by using an Information Card again.

https://wiki.prooph-board.com/assets/images/event-modeling/fleet-overview-state.png

One Slice at a Time

As a last step, we should mark the Slice as planned.

https://wiki.prooph-board.com/assets/images/event-modeling/slice-planned.gif

Slices are put along an imaginary timeline from left to right just like we do with events in a Big Picture Event Storming. You might have noticed that the Cards within a Slice are ordered from top to bottom: **UI > Command > Business Rules > Event**. This illustrates a specific point in time.

Each Slice should only contain one action (Command > Business Rule > Event) not more. This ensures, that the system as a whole is composed of small, simple building blocks. Connect the Slices through events and/or steps in the UI.

If you strictly follow this rule and learn how to design a system in such a way, you'll notice after some time that your code is very clean and easy to understand. Refactorings become a no-brainer. You can add new slices as reactions to existing events. You can replace or remove slices. And you can do this for years without increasing complexity. Sure, your system becomes larger, but zoomed-in to a single Slice it will still be as simple as the one we just designed. The Event Map will help you navigate through the system and keep an overview. So make sure that it stays in sync with the implementation.

Story Writing

Story Writing is optional. In fact, you already have all the information on the board, writing it down in a ticket might be useful, but it is not strictly necessary.

Now that we have designed a Slice on prooph board, we can easily create a ticket for it. We use a Github issue here as an example, but this could also be a task in Jira or similar.

https://wiki.prooph-board.com/assets/images/event-modeling/story-writing.png

Here is our recommended issue template:

Section	Description
Title	Slice Name
User Story	Derive from Actor + Command + Event(s)
Image	Make a screenshot of the Slice on prooph board -> paste in issue
Link to Event Map	Right click on prooph board Slice -> choose "Direct Link" -> paste in issue
Sub Tasks	Split design and development work into small chunks

With some routine you can create issues using this template in less than a minute. Compare this to your normal story writing and task breakdown sessions

You can also link the prooph board Slice with the issue by choosing “**Link to Task**” from the Slice context menu:

https://wiki.prooph-board.com/assets/images/event-modeling/link-to-task.gif

Feature Slicing

The key for any system to be maintainable over a long period of time is **composition**. A complex system should be composed of simple parts. When designing Slices around events, you can use a heuristic to keep them simple:

Too much cards in a single Slice are an alarming signal that the Slice is probably too big.

Let’s have a look at the next event on our Event Map: **Car Updated**. Sounds like a simple operation, right? But wait, what exactly do we want to update? A design session should give us some insights.

https://wiki.prooph-board.com/assets/images/CES/EventMap/Update_Car_information.png

We quickly realize that a car is going to have a lot of information assigned to it. When adding a car, we only focused on a small portion. The update command should support the full set, even thought it’s a lot.

We know that the Fleet Manager will update a car step by step. Not all information is available right from the beginning for example the licence plate or the equipment list. So does it make sense to have this huge update command? Wouldn’t it be better to split the command into smaller chunks and give the task more structure?

https://wiki.prooph-board.com/assets/images/event-modeling/update-car-slices.png

So instead of one big update command, we now have 5 distinct commands for updating different parts of a car. Those commands can be represented as 5 tabs in the UI or a 5-step editing process. In the system we gain some benefits from this design:

1. Fine-grained events

If we want to add more automation later, the events can act as triggers. Let’s say we want to run an image optimization process whenever an image is assigned to a car.

To add this feature, you don't need to touch existing functionality. Just react on `Image Assigned` events and call it a day!

2. Simple Slices

What if we later discover that car equipment can always be changed no matter if the car is booked or not, but basic and technical information are not allowed to change? You can adjust the business rule for the `Set Equipment` command without any risk of introducing a bug in the other commands!

3. Possibility to analyse behavior

We do have the feeling that fleet managers spent a significant amount of time maintaining car information. To validate our assumption we can look at the events and see how often a car is updated until it is ready to be published. We can also see what information is updated last or more often than other information. This can give us an idea how to improve the system and provide a metric to measure if our changes improve the situation or make it worse.

Prototyping

https://wiki.prooph-board.com/assets/images/cody-play/cody-play-in-action.gif

The Event Map is basically a blueprint of the system. You can use Cody Play to translate it into a working prototype.

TIPS AND TRICKS

You're viewing an old version of the wiki. Head over to the new Event Modeling section.

Sketch fast, complete later

Don't try to make the Event Map look perfectly arranged in a collaboration session. Design/sketch fast and assign a responsible person to clean up or add missing details after the session. This ensures a fast flow and does not kill creativity during collaboration.

Make use of Tags and Metadata

The Event Map should give all team members a good overview of the system, but not everybody is interested in all information. Card Metadata can be used to separate technical information like the schema of a command or event from more general information like its name and position in the business flow. Try to find a balance between always visible and secondary information. Card Tags on the other hand can highlight and group specific information.

1-2-4-all Event Modeling

1-2-4-all is a collaboration technique that fosters active contribution of all participants. It can be perfectly combined with Online Event Modeling if you have the right video tooling at hand.

Fortunately, prooph board has exclusive beta access to a video conferencing tool called “Swarply”, which allows us to offer Swarply integration (beta version) within the regular prooph board subscription plan. You can start a Swarply video session on any board by choosing “Join Session” in the top menu and enjoy distance-sensitive video avatars that enable parallel discussions, a precondition for 1-2-4-all.

So let’s say your team is in an Event Modeling session to design a new feature collaboratively. Instead of meeting in a video call and screen share a Jira board, you meet on prooph board with Swarply video avatars turned on.

1. The PO should provide information about the story or epic as usual.
2. Then each team member moves their video avatar to a free space on the board and does a litte Event Storming for them self. This should take 10 - 15 min. It activates the brain to focus on requirements and possible solutions.
3. Next, you pair with another teammate to merge your Event Stormings and create an Event Model. This works well with our distance sensitive video avatars. You don’t need breakout rooms to have parallel discussions in the same meeting. Set the timer for another 10 min.
4. Depending on group/team size either repeat step 3 by forming groups of 4 people and merge results or skip the 4-people phase and directly merge results with the whole team.
5. As good Event Modelers, you should have collected Hot Spots with open questions along the way and also have a good visual model at the end that will help you slice the story/epic and define acceptance criteria.
6. Assign open Hot Spots to people for clarification and finish the meeting with a feeling of real productive progress.

INTRODUCTION

Cody is a bot or agent that is able to translate an Event Map Design into working software. It's available in different variations to match the needs of different use cases.

Variant 1: Core SDK

The Core SDK is a Cody server that runs on your local machine. prooph board can connect to that server and sends information from a board to it. This happens when you open the Cody Console on prooph board, connect to the server in the console and then "trigger Cody" from the context menu of a selected card on prooph board. See the Intro Slides for more information.

The Core SDK is currently available for NodeJS and PHP.

If you would like to use Cody with another language, don't worry, the basic server implementation is quickly developed. Get in touch, if you want to contribute! The needed endpoints are documented on the Cody Server page.

Variant 2: Cody Engine

Cody Engine is a "batteries-included" prototyping and lowcode system. It is also available as Open Source and very flexible. While the Core SDK gives you the freedom to set up code generation exactly as you need it e.g. integrate it into your existing development stack, Cody Engine provides you with a ready-to-use solution. You can experience the power of **Event-Map-To-Code** without hassle.

Cody Engine is based on NodeJS, TypeScript and React.

Even if you use a different development stack, you might want to take a look at Cody Engine. The **prototyping mode** alone can save you a lot of time. You can quickly validate your design, ask stakeholders and users for feedback and only spend expensive implementation time, when you have the **feedback** that the design will work as expected.

SERVER IMPLEMENTATION

Implementing your own Cody server is as easy as implementing any other web application that handles HTTP requests. In this guide we're looking at the necessary steps and the API spec.

Intro Video

https://youtu.be/0FAgsPNqUV4

Intro Slides

https://wiki.prooph-board.com/cody/Cody-Server.html

Specification

So Cody is a HTTP server that receives requests from prooph board and returns **Cody Responses**. Those responses can be of different types to separate between informational messages (operation succeeded), questions (should this file really be overridden?) and errors (something went wrong).

A Cody Server should be able to handle a fixed set of HTTP requests:

View Spec in Swagger Editor on editor.swagger.io

NodeJS Example Server

The NodeJS Express server should give you an idea how to implement your own Cody Server. If you want to see more code, check out the complete implementation on Github.

This part of the documentation needs some more ❤️. We'll provide a more detailed explanation soon!

```
import compression from 'compression';
import cors from 'cors';
import {Request, Response} from "express";
import express from 'express';
import { Server } from 'http';
import http from 'http';
import {CodyConfig, ElementEdited, handleElementEdited, Sync} from './board/code';
import {makeNodeRecord, Node} from './board/graph';
import { greeting, IioSaidHello } from './general/greeting';
import { checkQuestion, handleReply, Reply, test } from './general/question';
import {CodyResponse, CodyResponseType} from './general/response';
import {Map} from "immutable";
// tslint:disable-next-line:no-var-requires
const bodyParser = require('body-parser');
// Simple Express server with some basic configuration like CORS handling
const codyServer = (codyConfig: CodyConfig): Server => {
    const app = express();
    const options: cors.CorsOptions = {
        allowedHeaders: [
            'Origin',
            'X-Requested-With',
            'Content-Type',
            'Accept',
            'X-Access-Token',
            'Authorization'
        ],
        credentials: true,
      methods: 'GET,HEAD,OPTIONS,PUT,PATCH,POST,DELETE',
      origin: '*',
      preflightContinue: false,
   };
   // GZIP compress resources served
   app.use(compression());
   app.use(cors(options));
   app.use(bodyParser.json());
   const server = http.createServer(app);
   enum Events {
      IioSaidHello= 'IioSaidHello',
      UserReplied = 'UserReplied',
      ElementEdited = 'ElementEdited',
      ConfirmTest = 'ConfirmTest',
   }
   enum Commands {
      Sync= 'Sync',
      SyncDeleted= 'SyncDeleted'
   }
   // Connect request initiated by prooph board
   app.post(`/messages/${Events.IioSaidHello}`, (req: Request<any, CodyResponse,
IioSaidHello>, res: Response<CodyResponse>) => {
      console.log(Events.IioSaidHello);
      // This server supports board synchronization. It's optional, but very useful for
advanced code generation
      // Here we just memoize that a sync is required before next code generation run
      codyConfig.context.syncRequired = true;
```

```
        // Send back a greeting response (CodyResponse of type: Info with a message saying hello to the user)
        res.send(greeting(req.body.user))
    });
    // Invoked when user replied to a Cody Question.
    app.post(`/messages/${Events.UserReplied}`, (req: Request<any, CodyResponse, Reply>, res: Response<CodyResponse>) => {
        console.log(Events.UserReplied, req.body);
        // handleReply has access to last CodyResponse, which was of type Question and contained a callback
        // that is now invoked with the user response
        handleReply(req.body.reply).then(codyRes => {
            res.send(checkQuestion(codyRes));
        }, reason => {
            res.send({
                cody: "Look's like something went wrong!",
                details: reason.toString(),
                type: CodyResponseType.Error
            });
        });
    });
    // Invoked for each element on prooph board when user triggered Cody.
    // Invocation is always single threaded. You can return a Question to the user and await a reply.
    app.post(`/messages/${Events.ElementEdited}`, (req: Request<any, CodyResponse, ElementEdited>, res: Response<CodyResponse>) => {
        console.log(Events.ElementEdited, req.body);
        // Do not trigger code generation, but instead send back a SyncRequired response
    // prooph board will invoke POST /messages/Sync and when finished trigger
ElementEdited again
    if(codyConfig.context.syncRequired) {
      // Reset in-memory node map as preparation for a full sync
      codyConfig.context.syncedNodes = Map<string, Node>();
      res.send({
        cody: 'I need to sync all elements first.',
        details: "Lean back for a moment. I'll let you know when I'm done.",
        type: CodyResponseType.SyncRequired
      })
      return;
    }

    // Map Node.Type to a hook (like onEvent) configured in codyConfig
    handleElementEdited(makeNodeRecord(req.body.node),
codyConfig).then(codyRes => {
      // checkQuestion memoize the callback, in case CodyResponse is of type
Question
      res.send(checkQuestion(codyRes));
    }, reason => {
      console.log(reason);
      res.send({
        cody: `Uh, sorry. Cannot handle element
${makeNodeRecord(req.body.node).getName()}!`,
        details: reason.toString(),
        type: CodyResponseType.Error
      });
    });
});

// If prooph board receives a SyncRequired CodyResponse

// it switches to synchronization mode and sends all elements in batches to this Sync
endpoint.

// The server should store a list of elements in memory.

app.post(`/messages/${Commands.Sync}`, (req: Request<any, CodyResponse,
Sync>, res: Response<CodyResponse>) => {

    console.log(Commands.Sync, "full sync");

    // Full sync is happening so we can turn off the flag

    codyConfig.context.syncRequired = false;


    let nodes: Node[] = [];

    if(req.body.nodes && Array.isArray(req.body.nodes)) {

      nodes = req.body.nodes.map(makeNodeRecord);

    } else {

      res.send({

        cody: 'No nodes given in sync request!',

        type: CodyResponseType.Error

      })

      return;

    }

    nodes.forEach(node => {

      console.log("synced node: ", node.getName());

      codyConfig.context.syncedNodes =
codyConfig.context.syncedNodes.set(node.getId(), node);

    })

    // Synced without errors, so we can return an empty response.
    // This avoids spam in the cody console. Empty responses are not shown to the
user.
    res.send({
      cody: '',
      type: CodyResponseType.Empty
    });
});


// After full sync, prooph board continues synchronizing elements each time a change
was made on the board (within the current session).
// Updated elements are sent in batches to this Sync endpoint. The server should
update its in-memory list of elements.
app.put(`/messages/${Commands.Sync}`, (req: Request<any, CodyResponse,
Sync>, res: Response<CodyResponse>) => {
    console.log(Commands.Sync, "edit sync");
    if(codyConfig.context.syncRequired) {
      // Seems like server lost in-memory sync due to restart but prooph board
continues to send sync requests
      // Ignore sync until user triggers next code generation and therefore next full
sync again
      console.log("sync ignored");
      res.send({
        cody: '',
        type: CodyResponseType.Empty
      });
      return;
    }
    let nodes: Node[] = [];
    if(req.body.nodes && Array.isArray(req.body.nodes)) {
      nodes = req.body.nodes.map(makeNodeRecord);
    } else {
      res.send({
        cody: 'No nodes given in sync request!',
        type: CodyResponseType.Error
      })
      return;
    }
    nodes.forEach(node => {
      console.log("synced node: ", node.getName(), `(${node.getId()} -
${node.getType()})`, "parent: ", node.getParent()? node.getParent()!.getId() : '-');
      codyConfig.context.syncedNodes =
codyConfig.context.syncedNodes.set(node.getId(), node);
    })
    res.send({
      cody: '',
      type: CodyResponseType.Empty
    });
});
// Deleted elements are synced through this endpoint. HTTP DELETE requests do
not have a body,
// hence we use another POST request to let the Cody server remove elements from
its in-memory element list.
// Deleted elements are sent in batches.
app.post(`/messages/${Commands.SyncDeleted}`, (req: Request<any,
CodyResponse, Sync>, res: Response<CodyResponse>) => {
    console.log(Commands.SyncDeleted);

    if(codyConfig.context.syncRequired) {
      // Seems like server lost in-memory sync due to restart but prooph board
continues to sent sync requests
      // Ignore sync until user triggers next code generation and therefore next full
sync
      console.log("sync ignored");
      res.send({
        cody: '',
        type: CodyResponseType.Empty
      });
      return;
    }
    let nodes: Node[] = [];
    if(req.body.nodes && Array.isArray(req.body.nodes)) {
      nodes = req.body.nodes.map(makeNodeRecord);
    } else {
      res.send({
        cody: 'No nodes given in sync request!',
        type: CodyResponseType.Error
      })
      return;
    }
    nodes.forEach(node => {
      console.log("synced node: ", node.getName(), `(${node.getId()} -
${node.getType()})`, "parent: ", node.getParent()? node.getParent()!.getId() : '-');
      codyConfig.context.syncedNodes =
codyConfig.context.syncedNodes.delete(node.getId());
    })
    res.send({
      cody: '',
      type: CodyResponseType.Empty
    });
});
// Invoked when user types /talk into console. CodyResponse should be a test
question.
// The endpoint is meant to be used as a health check to verify that question-
answer/confirmation mechanism works
app.post(`/messages/${Events.ConfirmTest}`, (req: Request<any, CodyResponse,
any>, res: Response<CodyResponse>) => {
    console.log(Events.ConfirmTest);
    res.send(checkQuestion(test()));
});
return server;
}
export default codyServer;
```

NODEJS CODY TUTORIAL

prooph board can connect to a coding bot called **Cody**. With its help you can generate working code from an event model. This tutorial guides you through the first steps. You'll learn the basics as well as customizing code generation to suit your needs.

Preparation

We've prepared a repository containing some exercises.

1. Clone Repo and execute setup

```
git clone git@github.com:proophboard/ts-cody-tutorial.git
cd ts-cody-tutorial
```

Install dependencies

```
npm install
```

Initialize Cody

```
npm run bootstrap
```

Start Cody Server

```
npm run cody
```

1. Create cody tutorial board on prooph board and test connection

You can use prooph board free version for the tutorial (no login required).

> ***prooph board*** *is a modeling tool specifically designed for remote Event Storming. It ships with realtime collaboration features for teams (only available in paid version). The free version is a standalone variant without any backend connection. Your work is stored in local storage and can be exported. It is hosted on Github Pages and has the same code generation capabilities as the SaaS version.*

Create a new board called "Cody Tutorial". You'll be redirected to the fresh board. Choose "Cody" from top menu to open the **Cody Console**. Just hit ENTER in the console to connect to the default Cody server that we've setup and started in the previous step.

Finally type "**/help**" in the console to let Cody explain the basic functionality.

https://wiki.prooph-board.com/assets/images/cody_tutorial_connect.gif

Cody Explained

Cody is an express http server running on your local machine. prooph board can connect to it using `/connect [server address]` in the **Cody Console**. `codyconfig.ts` is the central place to configure code generation. Generated code needs to be written to a target repository (e.g. src/). The tutorial contains a Jest test case for each exercise to validate that generated code looks like the expected one.

Exercise I

The Cody Server should be running and listen on `http://localhost:3311`:
💡 *Note: nodemon restarts the server on changes. If something does not work as expected try to restart the server manually and take a look at the logs in the console.`*

Test-Driven Exercises

Navigate to the tutorial project in a **second** console `cd ts-cody-tutorial` and run the first exercise using:

```
npm run exercise1
```

https://wiki.prooph-board.com/assets/images/node_cody_tutorial_failing_exercise_1.png

Of course the test case is failing. it is looking for a **Command** called **AddBuilding** and that's the first file we want to generate from a prooph board event map. You might have noticed the commented hooks in `codyconfig.ts`. What we need is a `onCommandHook`, so let's create one!

Create Command Hook

```
# current dir: ts-cody-tutorial

# Create interface for Context object
echo -e "export interface Context {\n  srcFolder: string;\n}\n" >> .codyhooks/Context.ts

# Create onCommandHook.ts
touch .codyhooks/onCommandHook.ts
```

Open `.codyhooks/onCommandHook.ts` in your favorite editor and copy this content into it:

```
import {Context} from "./Context";
import {CodyHook, CodyResponse, Node} from "@proophboard/cody-types";
import {isCodyError, nodeNameToPascalCase, writeFileSync} from "@proophboard/cody-utils";

/**
 * onCommandHook
 *
 * @param {Node}    command  Information about command card received from prooph board
 * @param {Context} ctx      Context object populated in codyconfig.ts
 * @returns Promise<CodyResponse>
 */
export const onCommandHook: CodyHook<Context> = async (command: Node, ctx: Context):
Promise<CodyResponse> => {
    // Cody ships with some util functions for common tasks
    const cmdName = nodeNameToPascalCase(command);
    const cmdFilename = cmdName+'.ts';
    // ctx.srcFolder is set in codyconfig.ts
    const cmdFile = ctx.srcFolder + `/Command/${cmdFilename}`;
    let successDetails = 'Checklist\n\n';

    const content = `
export interface ${cmdName} {}
`;

    // Util functions return Cody-Error-Responses in case something went wrong
    const writeFileErr = writeFileSync(cmdFile, content);

    if(isCodyError(writeFileErr)) {
        return writeFileErr;
    }

    successDetails = successDetails + `✓  Command file ${cmdFile} written\n`;

    // Cody responses can be formatted similar to browser console formatting
    // @see https://developers.google.com/web/tools/chrome-devtools/console/console-
write#styling_console_output_with_css
    return {
        cody: `Wasn't easy, but command ${cmdName} should work now!`,
        details: ['%c'+successDetails, 'color: #73dd8e;font-weight: bold'],
    }
}
```

Register Hook in codyconfig

To activate the hook, we need to register it in `codyconfig.ts`. You can find the file in the project root. We also set a `srcFolder` property in the context part of the configuration.

The `context` object is passed as a second argument to each hook. You can use it to pass your own configuration options into the hooks.

```
import {Map} from "immutable";
import {onCommandHook} from "./.codyhooks/onCommandHook";

module.exports = {
    context: {
        /*
         * The context object is passed to each hook as second argument
         * use it to pass configuration to your hooks like a src directory, credentials, ...
         */
        srcFolder: 'src',

        // This Cody server implements the optional Sync flow and stores all synced nodes in this
context property
        syncedNodes: Map({})
    },
    hooks: {
        /**
         * Uncomment and implement a hook to activate it
         */
        // onAggregate: onAggregateHook,
        // onBoundedContext: onBoundedContextHook,
        onCommand: onCommandHook,
        // onDocument: onDocumentHook,
        // onEvent: onEventHook,
        // onFeature: onFeatureHook,
        // onFreeText: onFreeTextHook,
        // onExternalSystem: onExternalSystemHook,
        // onIcon: onIconHook,
        // onImage: onImageHook,
        // onHotSpot: onHotSpotHook,
        // onLayer: onLayerHook,
        // onPolicy: onPolicyHook,
        // onRole: onRoleHook,
        // onUi: onUiHook,
    }
}
```

Modeling On Event Map

Cody is now able to turn information from command stickies (on a prooph board event map) into Typescript interfaces. Switch to the Cody Tutorial board in prooph board and add a command sticky (blue one) with label `AddBuilding`. Right click on the newly created sticky and choose **Trigger Cody** from context menu. In the **Cody Console** you can check the response. Cody should tell you: `Wasn't easy, but command AddBuilding should work now!`.
https://wiki.prooph-board.com/assets/images/node_cody_tutorial_add_building_command.gif

Awesome, it works! Rerun:

```
npm run exercise1
```

It should turn green now. The test verifies that a *cody-tutorial/exercises/src/Command/AddBuilding.ts* has been generated by **Cody**.

https://wiki.prooph-board.com/assets/images/node_cody_tutorial_succeeding_exercise_1.png

In exercise I we implemented our first **Cody Hook** and registered it in `codyconfig.ts`. The hook is called with information received from a prooph board event map and with a user defined context object, which can be used to pass configuration options (like a source path) to each hook. We can trigger the hook from prooph board by selecting an appropriate sticky on an event map and **Trigger Cody** from context menu.

Exercise II

To see what we have to do next, execute:

```
npm run exercise2
```

https://wiki.prooph-board.com/assets/images/node_cody_tutorial_failing_exercise_2.png

We're asked to add a `buildingId` and a `name` property (both of type `string`) to the `AddBuilding` command. Now it would be easy to just open the file and add those two properties. But we should expand our code generation logic instead. This has some significant advantages:

- a prooph board event map acts as documentation
- one can generate contracts like an OpenAPI schema from Card Metadata
- it is easier to discuss and design new features or refactorings when such details are included on the event map

Expand Command Hook Logic

Ok let’s use a simple code generator implementation first to understand the basics. In a later tutorial part we’ll look at some useful libraries that provide abstractions for advanced use cases.

Change the command hook in `.codyhooks/onCommandHook.ts` like this:

```
import {Context} from "./Context";
import {CodyHook, CodyResponse, CodyResponseType, Node} from "@proophboard/cody-types";
import {isCodyError, nodeNameToPascalCase, parseJsonMetadata, writeFileSync} from
"@proophboard/cody-utils";

/**
 * onCommandHook
 *
 * @param {Node}    command  Information about command card received from prooph board
 * @param {Context} ctx      Context object populated in codyconfig.ts
 * @returns Promise<CodyResponse>
 */
export const onCommandHook: CodyHook<Context> = async (command: Node, ctx: Context):
Promise<CodyResponse> => {
    // Cody ships with some util functions for common tasks
    const cmdName = nodeNameToPascalCase(command);
    const cmdFilename = cmdName+'.ts';
    // ctx.srcFolder is set in codyconfig.ts
    const cmdFile = ctx.srcFolder + `/Command/${cmdFilename}`;
    let successDetails = 'Checklist\n\n';

    const commandMetadata = parseJsonMetadata<{[prop: string]: string}>(command);

    if(isCodyError(commandMetadata)) {
        return commandMetadata;
    }

    if(typeof commandMetadata !== 'object') {
        // You can return your own error responses, too
        return {
            cody: `I expected metadata of command "${command.getName()}" to be an object, but it
is of type: `
                + typeof commandMetadata,
            type: CodyResponseType.Error
        };
    }

    let properties = "";

    for(const prop in commandMetadata) {
        if(commandMetadata.hasOwnProperty(prop)) {
            const type = commandMetadata[prop];
            // Append property to properties string which is inserted in interface below
            properties = properties + `  ${prop}: ${type};\n`;
        }
    }

    const content = `
export interface ${cmdName} {
${properties}
}
`;

    // Util functions return Cody-Error-Responses in case something went wrong
    const writeFileErr = writeFileSync(cmdFile, content);

    if(isCodyError(writeFileErr)) {
        return writeFileErr;
    }

    successDetails = successDetails + `✓  Command file ${cmdFile} written\n`;

    // Cody responses can be formatted similar to browser console formatting
    // @see https://developers.google.com/web/tools/chrome-devtools/console/console-
write#styling_console_output_with_css
    return {
        cody: `Wasn't easy, but command ${cmdName} should work now!`,
        details: ['%c'+successDetails, 'color: #73dd8e;font-weight: bold'],
    }
}
```

Set Command Metadata in prooph board

Now we can switch to prooph board and set the following metadata in JSON format to the `AddBuilding` command:

```
{
  "buildingId": "string",
  "name": "string"
}
```

💡 *Hint: Metadata can be set by opening the Metadata Sidebar (choose Metadata from top menu) and selecting the appropriate card or sticky note. Metadata changes are saved automatically.*

Once metadata is set we can trigger Cody again …

https://wiki.prooph-board.com/assets/images/node_cody_tutorial_add_building_properties.gif

… and validate the result by executing the test run again:

```
npm run exercise2
```

https://wiki.prooph-board.com/assets/images/node_cody_tutorial_succeeding_exercise_2.png

Cards on a prooph board event map can have additional information set as metadata. By default metadata is stored in JSON format. Metadata itself is schemaless, meaning users are free to define any structure. The structure can be defined and even be type checked using **Metadata Templates**. In a Cody hook you have access to raw metadata and use it for advanced code generation.

Exercise III

One last basic concept is missing to complete the picture. On a prooph board event map you model message flows, behavior, business processes. Different objects interact with each other by exchanging messages like commands and events. A Cody Hook can make use of this information to assist developers by generating all required glue code if not a fully working implementation (depends on complexity).

We start again by looking at the failing test case for exercise III:

```
npm run exercise3
```

https://wiki.prooph-board.com/assets/images/node_cody_tutorial_failing_exercise_3.png

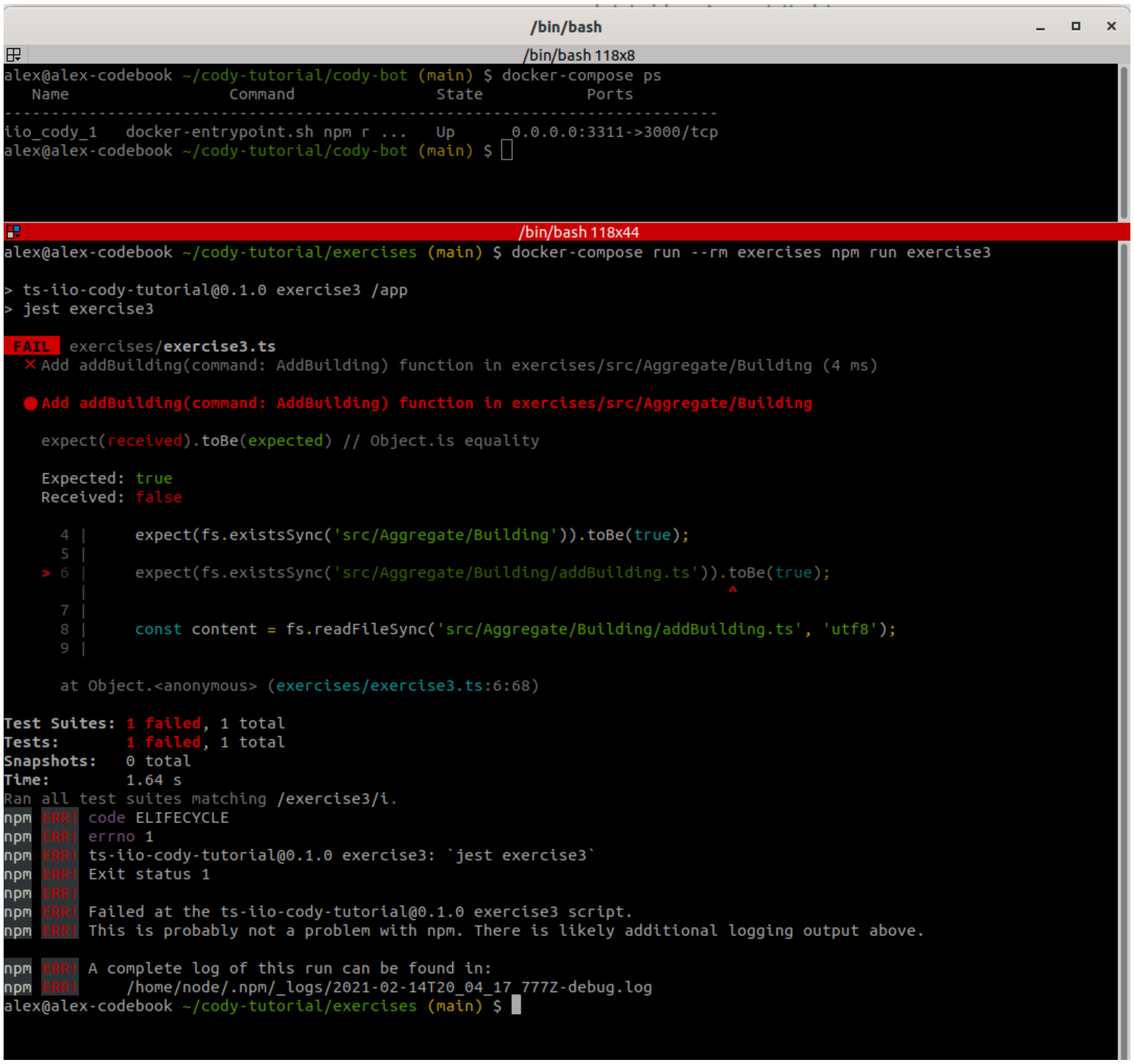
Ok, this time we're asked to generate a command handling function named `addBuilding` in an aggregate directory `Aggregate/Building`. The function should handle `AddBuilding` command from previous exercises.

Aggregate Hook

In Exercise I you've learned how to create a Command Hook and register it in `codyconfig.ts`. Do the same for an `onAggregateHook` and let the hook generate a directory **src/Aggregate/${aggregateName}**.
💡 *Hint: You can use the util function `mkdirIfNotExistsSync` to generate the directory. Make sure to check if the util function returns a Cody-Error-Response!*
If you think the hook is ready, switch to prooph board, add an aggregate card with label `Building` on the event map and trigger Cody.
https://wiki.prooph-board.com/assets/images/node_cody_tutorial_building_aggregate.gif

Did it work? You can verify the result by executing:

```
npm run exercise3
```



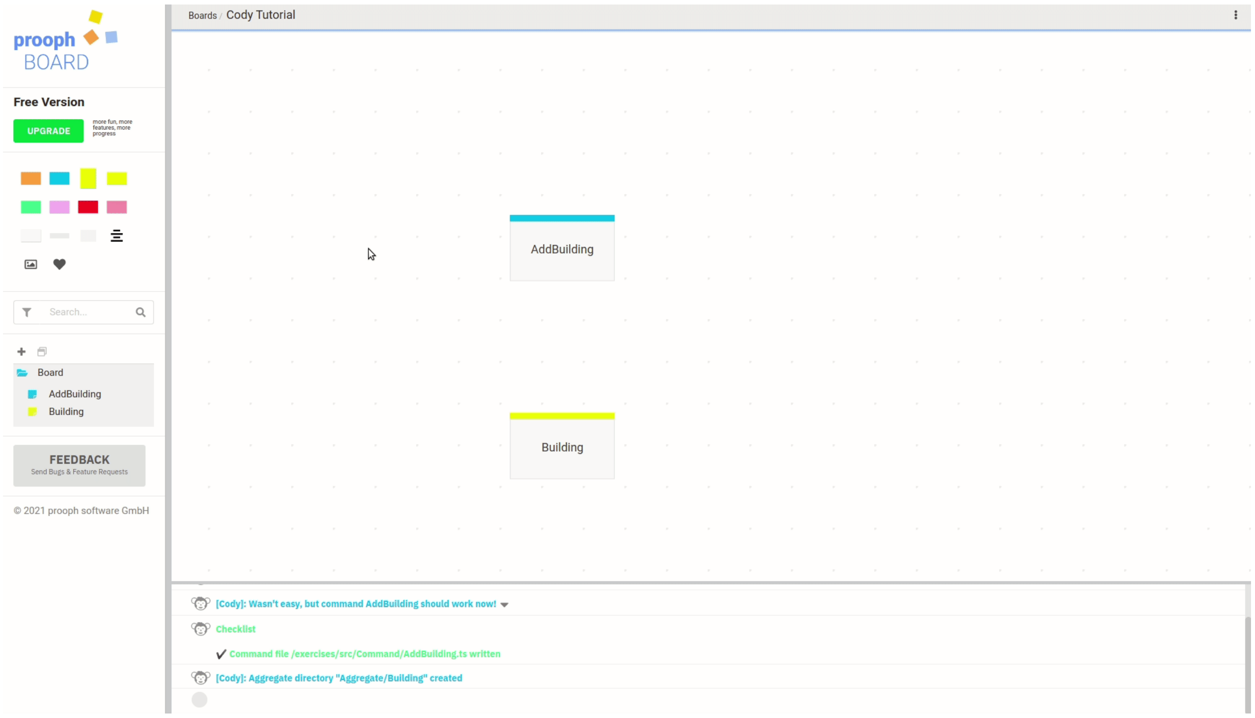
```
alex@alex-codebook ~/cody-tutorial/cody-bot (main) $ docker-compose ps
   Name                  Command               State           Ports
---------------------------------------------------------------------------
iio_cody_1   docker-entrypoint.sh npm r ...   Up      0.0.0.0:3311->3000/tcp
alex@alex-codebook ~/cody-tutorial/cody-bot (main) $
```

```
alex@alex-codebook ~/cody-tutorial/exercises (main) $ docker-compose run --rm exercises npm run exercise3

> ts-iio-cody-tutorial@0.1.0 exercise3 /app
> jest exercise3

 FAIL  exercises/exercise3.ts
  ✕ Add addBuilding(command: AddBuilding) function in exercises/src/Aggregate/Building (4 ms)

  ● Add addBuilding(command: AddBuilding) function in exercises/src/Aggregate/Building

    expect(received).toBe(expected) // Object.is equality

    Expected: true
    Received: false

      4 |     expect(fs.existsSync('src/Aggregate/Building')).toBe(true);
      5 |
    > 6 |     expect(fs.existsSync('src/Aggregate/Building/addBuilding.ts')).toBe(true);
        |                                                                    ^
      7 |
      8 |     const content = fs.readFileSync('src/Aggregate/Building/addBuilding.ts', 'utf8');
      9 |

      at Object.<anonymous> (exercises/exercise3.ts:6:68)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        1.64 s
Ran all test suites matching /exercise3/i.
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! ts-iio-cody-tutorial@0.1.0 exercise3: `jest exercise3`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the ts-iio-cody-tutorial@0.1.0 exercise3 script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /home/node/.npm/_logs/2021-02-14T20_04_17_777Z-debug.log
alex@alex-codebook ~/cody-tutorial/exercises (main) $
```

Test case is still failing but the error has changed. `src/Aggregate/Building/addBuilding.ts` file is missing.

Event Map Connections

Go back to the event map and draw an arrow from `AddBuilding` command to `Building` aggregate.



```
[Cody]: Wasn't easy, but command AddBuilding should work now!
Checklist
✓ Command file /exercises/src/Command/AddBuilding.ts written
[Cody]: Aggregate directory "Aggregate/Building" created
```

The connection we've just drawn can be read in a hook. Here is the `Node` interface passed to a hook:

```
interface Node {
  getId: () => NodeId;
  getName: () => NodeName;
  getDescription: () => NodeDescription;
  getType: () => NodeType;
  getLink: () => NodeLink;
  getTags: () => List<NodeTag>;
  isLayer: () => boolean;
  isDefaultLayer: () => boolean;
  getParent: () => Node | null;
  getChildren: () => List<Node>;
  getGeometry: () => GraphPoint;
  getSources: () => List<Node>;
  getTargets: () => List<Node>;
  getMetadata: () => string | null;
}
```

We already worked with `getName()` and `getMetadata()`. From a `Node` you can follow connections using `getSources()` and `getTargets()`. In our case the `AddBuilding` command node has the `Building` aggregate referenced as a target and from `Building` point of view `AddBuilding` command is a source.

One important thing to note here is that a hook gets only access to directly connected nodes, even if those nodes have further connections! This avoids circular reference problems and keeps payload exchanged between prooph board and Cody small. However, it's possible to request a full board sync and build up a node map in Cody. But this is an advanced topic, that should be covered later.

💡 *`Node` interface also provides information about `getParent()` and `getChildren()` which are useful when grouping cards in a frame*

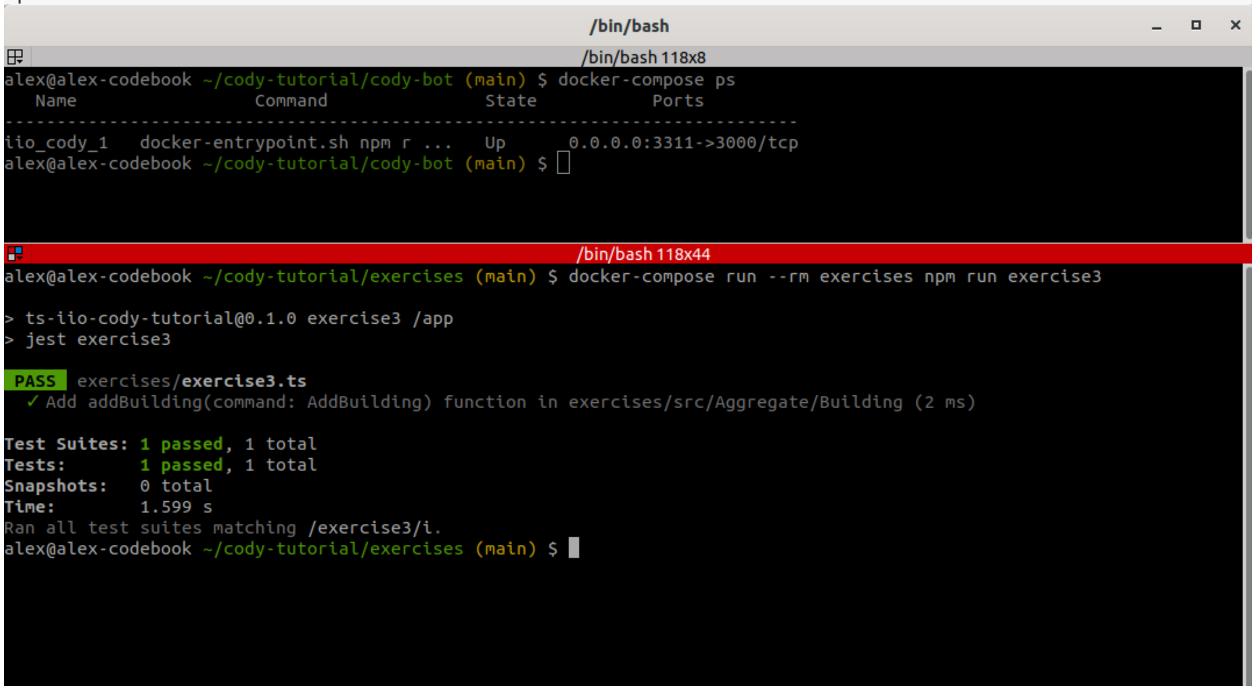
Enough theory! Let’s see it in action. Update your `onAggregateHook` with this version:

```
import {CodyHook, CodyResponse, Node, NodeType} from "@proophboard/cody-types";
import {Context} from "./Context";
import {
    getSingleSource,
    isCodyError,
    mkdirIfNotExistsSync,
    nodeNameToCamelCase,
    nodeNameToPascalCase, writeFileSync
} from "@proophboard/cody-utils";

export const onAggregateHook: CodyHook<Context> = async (aggregate: Node, ctx: Context):
Promise<CodyResponse> => {
    const aggregateName = nodeNameToPascalCase(aggregate);
    const aggregateDir = ctx.srcFolder + `/Aggregate/${aggregateName}`;
    let successDetails = 'Checklist\n\n';

    const dirErrResponse = mkdirIfNotExistsSync(aggregateDir);

    if(isCodyError(dirErrResponse)) {
        return dirErrResponse;
    }

    successDetails = successDetails + `✓  Aggregate directory ${aggregateDir} exists\n`;

    const command = getSingleSource(aggregate, NodeType.command);

    if(isCodyError(command)) {
        return command;
    }

    const commandName = nodeNameToPascalCase(command);
    const commandFunction = nodeNameToCamelCase(command);
    const functionFile = `${commandFunction}.ts`;
    const content = `import {${commandName}} from "../../Command/${commandName}";

export function ${commandFunction}(command: ${commandName}): void {}
`;

    const writeFileErr = writeFileSync(aggregateDir + `/${functionFile}`, content);

    if(isCodyError(writeFileErr)) {
        return writeFileErr;
    }

    successDetails = successDetails + `✓  Command handler ${aggregateDir}/${functionFile}
created\n`;

    return {
        cody: `${aggregateName} Aggregate can now handle ${commandName} commands`,
        details: ['%c'+successDetails, 'color: #73dd8e;font-weight: bold'],
    }
}
```

The implementation should be self explanatory. The util function `getSingleSource` is used to ensure that an aggregate card should only have one command as a source. If more than one is found a Cody error is returned. This way you can harden your hooks and validate event maps according to your own rules.

Trigger Cody with the `Building` aggregate card in prooph board and run exercise III once again:

```
npm run exercise3
```



```
alex@alex-codebook ~/cody-tutorial/cody-bot (main) $ docker-compose ps
   Name                  Command                 State              Ports
---------------------------------------------------------------------------
iio_cody_1   docker-entrypoint.sh npm r ...    Up      0.0.0.0:3311->3000/tcp
alex@alex-codebook ~/cody-tutorial/cody-bot (main) $
```

```
alex@alex-codebook ~/cody-tutorial/exercises (main) $ docker-compose run --rm exercises npm run exercise3

> ts-iio-cody-tutorial@0.1.0 exercise3 /app
> jest exercise3

 PASS  exercises/exercise3.ts
  ✓ Add addBuilding(command: AddBuilding) function in exercises/src/Aggregate/Building (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.599 s
Ran all test suites matching /exercise3/i.
alex@alex-codebook ~/cody-tutorial/exercises (main) $
```

This exercise introduced the last basic building block: **Connections between cards**. Depending on arrow direction on prooph board a connected card appears either in the `sources` or `targets` collection of the `Node` passed to a hook. It's important to keep in mind that a hook only has access to directly connected cards.

Exercise IV

Exercise IV coming soon!

Meanwhile you can checkout how to implement a cody server in your preferred language or check the following example boards to learn more about working on prooph board:

- Annotated prooph board Example
- Event Storming - The Picture
- Event Modeling Example

PHP CODY TUTORIAL

prooph board can connect to a coding bot called **Cody**. With its help you can generate working code from an event map. This tutorial guides you through the first steps. You'll learn the basics as well as customizing code generation to suit your needs.

Preparation

We've prepared a repository containing some exercises.

Please make sure you have installed Docker and Docker Compose to execute the exercises. Jest is used to validate the results.

1. Clone Repo and execute setup

```
git clone git@github.com:proophboard/php-cody-tutorial.git
cd php-cody-tutorial
./setup.sh
```

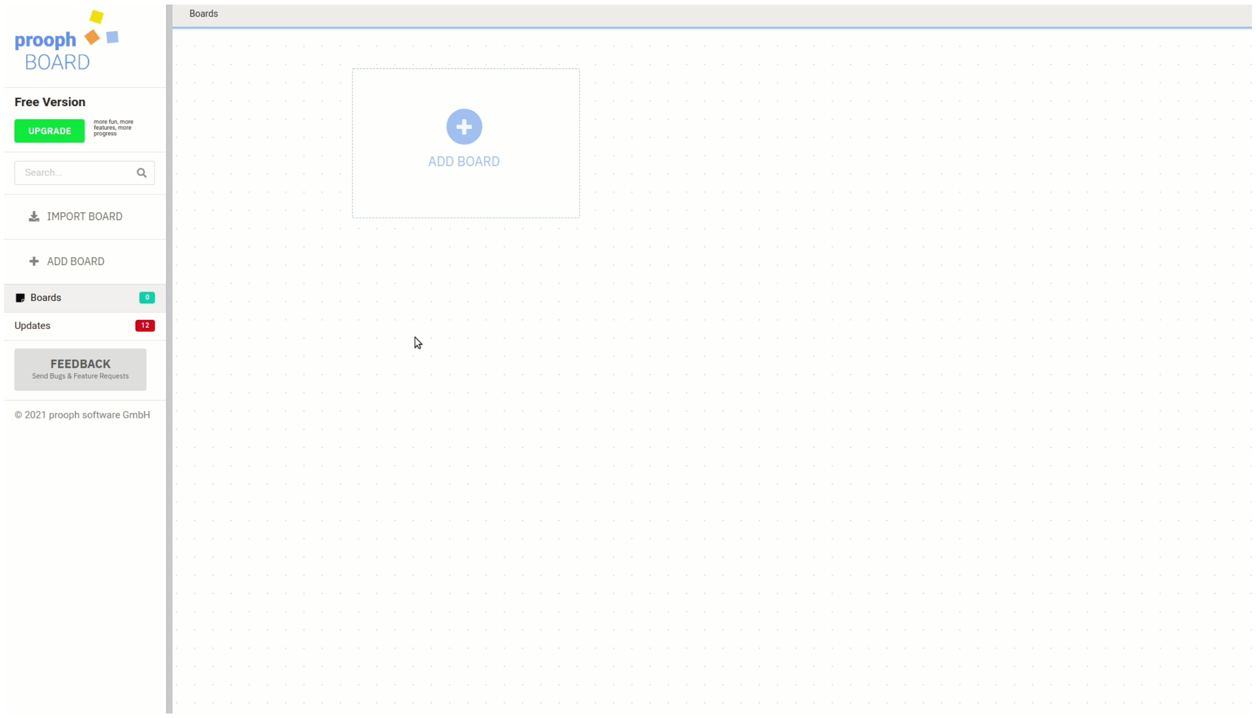
1. Create cody tutorial board on prooph board and test connection

You can use prooph board free version for the tutorial (no login required).

> ***prooph board*** *is a modeling tool specifically designed for remote Event Storming. It ships with realtime collaboration features for teams (only available in paid version). The free version is a standalone variant without any backend connection. Your work is stored in local storage and can be exported. It is hosted on Github Pages and has the same code generation capabilities as the SaaS version.*

Create a new board called "Cody Tutorial". You'll be redirected to the fresh board. Choose "Cody" from top menu to open the **Cody Console**. Just hit ENTER in the console to connect to the default Cody server that we've setup and started in the previous step.

Finally type "**/help**" in the console to let Cody explain the basic functionality.



Cody Explained

Cody is a http server running in a docker container on your local machine. prooph board can connect to it using `/connect [server address]` in the **Cody Console**. `codyconfig.php` is the central place to configure code generation. Generated code needs to be written to a target repository. We can use a docker volume mount to give **Cody** access to our project repository. In the tutorial our project is represented by `exercises`, which contains a PHPUnit test case for each exercise to validate that generated code looks like the expected one.

Exercise I

Your folder structure should look like this:

```
cody-tutorial
|__ cody-bot
|__ exercises
```

The Cody Server should be running and listen on `http://localhost:3311`:

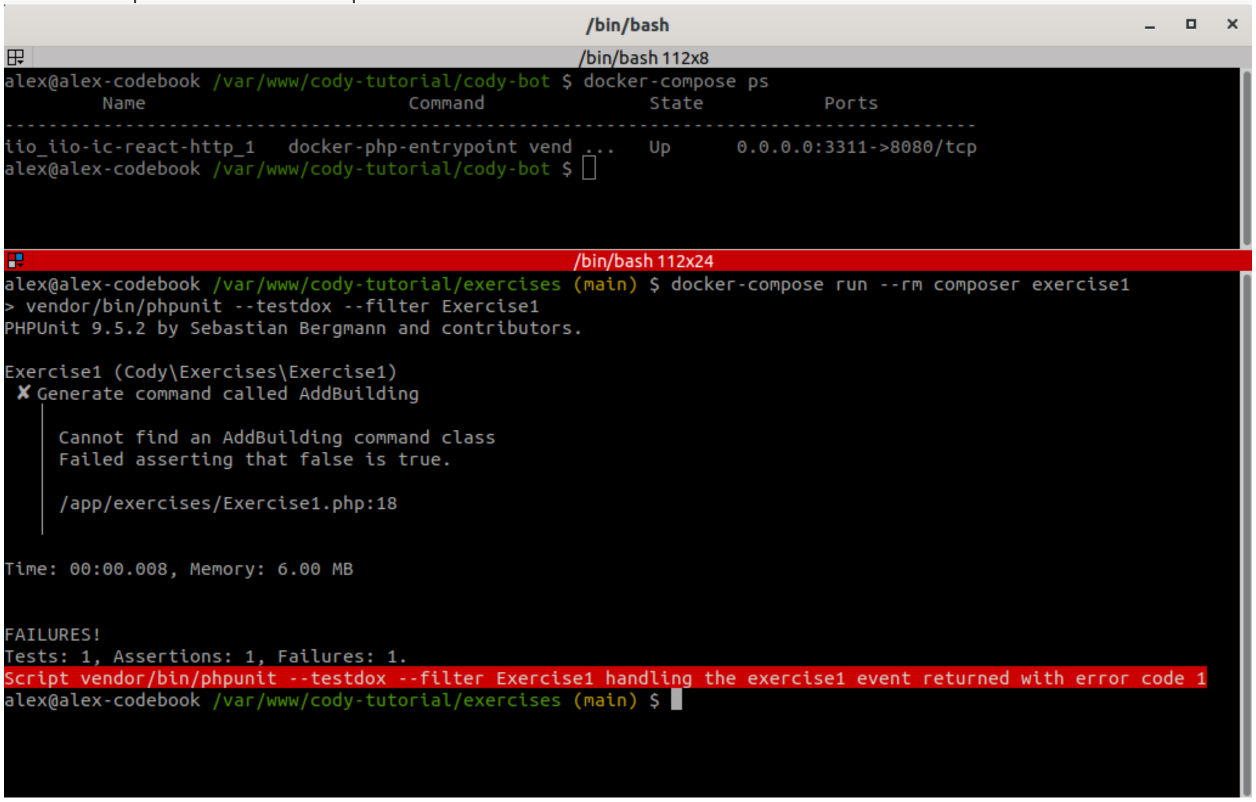
```
cd cody-tutorial/cody-bot
docker-compose ps

         Name                        Command               State           Ports
----------------------------------------------------------------------------------------
iio_iio-ic-react-http_1   docker-php-entrypoint vend ...   Up      0.0.0.0:3311->8080/tcp
```

💡 *Note: Use `dev.sh` to start **Cody**. After that you can use docker-compose commands to stop, start again, inspect logs, etc. A file watcher restarts the server on changes. If something does not work as expected try to restart the server manually and take a look at the logs: `docker-compose stop && docker-compose up -d && docker-compose logs -f`*

Test-Driven Exercises

Navigate to the tutorial project in a **second** console and run the first exercise using:

```
cd cody-tutorial/exercises
docker-compose run --rm composer exercise1
```



```
/bin/bash
/bin/bash 112x8
alex@alex-codebook /var/www/cody-tutorial/cody-bot $ docker-compose ps
         Name                          Command               State           Ports
------------------------------------------------------------------------------------------
iio_iio-ic-react-http_1   docker-php-entrypoint vend ...   Up      0.0.0.0:3311->8080/tcp
alex@alex-codebook /var/www/cody-tutorial/cody-bot $
```

```
/bin/bash 112x24
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $ docker-compose run --rm composer exercise1
> vendor/bin/phpunit --testdox --filter Exercise1
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

Exercise1 (Cody\Exercises\Exercise1)
 ✘ Generate command called AddBuilding
   │
   │ Cannot find an AddBuilding command class
   │ Failed asserting that false is true.
   │
   │ /app/exercises/Exercise1.php:18
   │

Time: 00:00.008, Memory: 6.00 MB


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
Script vendor/bin/phpunit --testdox --filter Exercise1 handling the exercise1 event returned with error code 1
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $
```

Of course the test case is failing. it is looking for a **Command** called **AddBuilding** and that's the first class we want to generate from a prooph board event map. You might have noticed the commented hooks in `codyconfig.php`. What we need is a `CommandHook`, so let's create one!

Create Command Hook

```
# current dir: cody-tutorial/cody-bot
# Create a Hook folder
mkdir src/Hook
# Create CommandHook.php
touch src/Hook/CommandHook.php
```

Open `src/Hook/CommandHook.php` in your favorite editor and copy this content into it:

```
<?php
declare(strict_types=1);

namespace EventEngine\InspectioCody\Hook;

use EventEngine\InspectioCody\Board\BaseHook;
use EventEngine\InspectioCody\Http\Message\CodyResponse;
use EventEngine\InspectioCody\Http\Message\Response;
use EventEngine\InspectioGraphCody\Node;

final class CommandHook extends BaseHook //BaseHook provides some helper methods like writeFile()
{
    /**
     * @param  Node     $command Information about Command sticky received from prooph board
event map
     * @param  object   $context Context object populated in codyconfig.php
     * @return CodyResponse      Response sent back to prooph board, shown in Cody Console
     */
    public function __invoke(Node $command, object $context): CodyResponse
    {
        $commandName = $command->name();
        $commandFile = $commandName . '.php';
        $commandPath = $context->path . '/Command/' . $commandFile;

        $code = <<<CODE
        <?php
        declare(strict_types=1);

        namespace Cody\Tutorial\Command;

        class $commandName
        {

        }
        CODE;

        $this->writeFile($code, $commandPath);

        return Response::fromCody(
            "Command \"{$commandName}\" generated",
            ["Command written to {$commandPath}"]
        );
    }
}
```

Register Hook in codyconfig

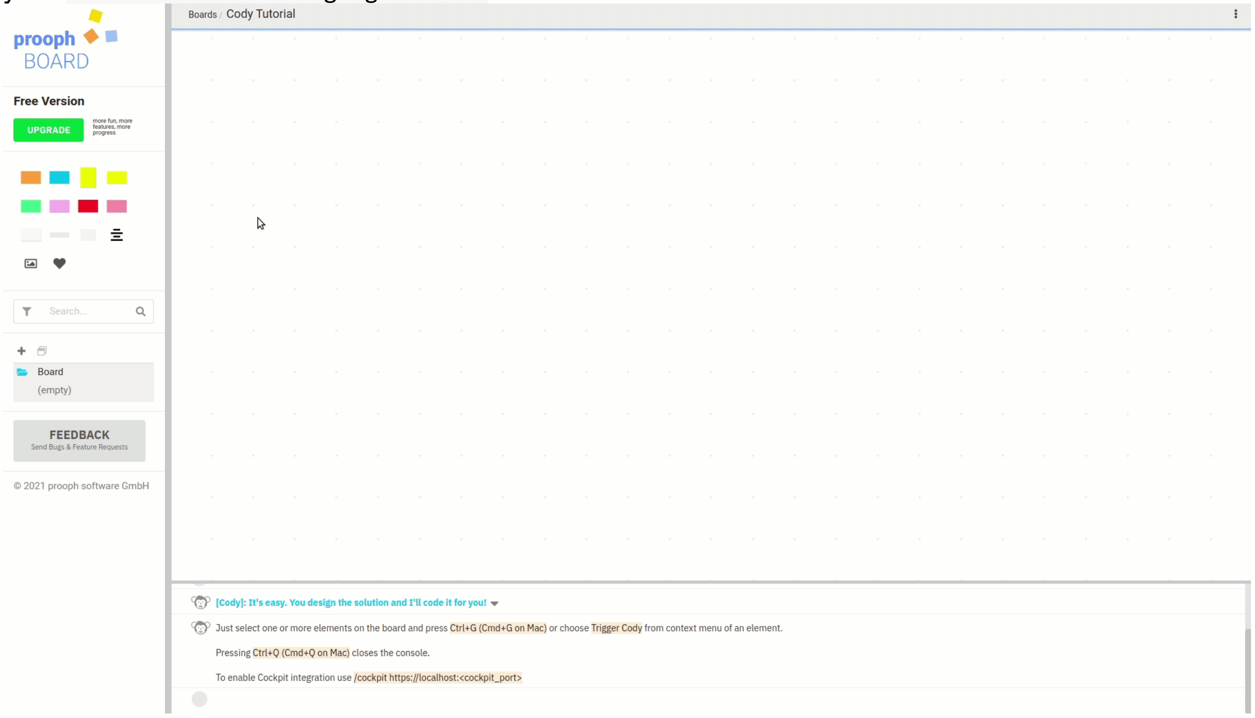
To activate the hook, we need to register it in `codyconfig.php`:

```
<?php

/**
 * @see       https://github.com/event-engine/php-inspectio-cody for the canonical source
repository
 * @copyright https://github.com/event-engine/php-inspectio-cody/blob/master/COPYRIGHT.md
 * @license   https://github.com/event-engine/php-inspectio-cody/blob/master/LICENSE.md MIT
License
 */

declare(strict_types=1);

use EventEngine\InspectioCody\CodyConfig;
// IMPORT COMMAND HOOK
use EventEngine\InspectioCody\Hook\CommandHook;

$context = new class() implements \EventEngine\InspectioCody\CodyContext {
    public function isFullSyncRequired(): bool
    {
        // TODO: Implement isFullSyncRequired() method.
        return false;
    }

    public function clearGraph(): void
    {
        // TODO: Implement clearGraph() method.
    }
}; // replace it with your own context class

$context->path = '/exercises/src';
```

```
return new CodyConfig(
    $context,
    [
//        CodyConfig::HOOK_ON_AGGREGATE => new AggregateHook(),
        // UNCOMMENT COMMAND HOOK
        CodyConfig::HOOK_ON_COMMAND => new CommandHook(),
//        CodyConfig::HOOK_ON_EVENT => new EventHook(),
//        CodyConfig::HOOK_ON_POLICY => new PolicyHook(),
//        CodyConfig::HOOK_ON_DOCUMENT => new DocumentHook(),
    ]
);
```

Modeling On Event Map

Cody is now able to turn information from command stickies (on a prooph board event map) into PHP classes. Switch to the Cody Tutorial board in prooph board and add a command sticky (blue one) with label `AddBuilding`. Right click on the newly created sticky and choose **Trigger Cody** from context menu. In the **Cody Console** you can check the response. Cody should tell you: `Command "AddBuilding" generated`.


prooph BOARD
Free Version
UPGRADE
more fun, more features, more progress
Search...
Board
(empty)
FEEDBACK
Send Bugs & Feature Requests
© 2021 prooph software GmbH
Boards / Cody Tutorial

[Cody]: It's easy. You design the solution and I'll code it for you!

Just select one or more elements on the board and press Ctrl+G (Cmd+G on Mac) or choose Trigger Cody from context menu of an element.

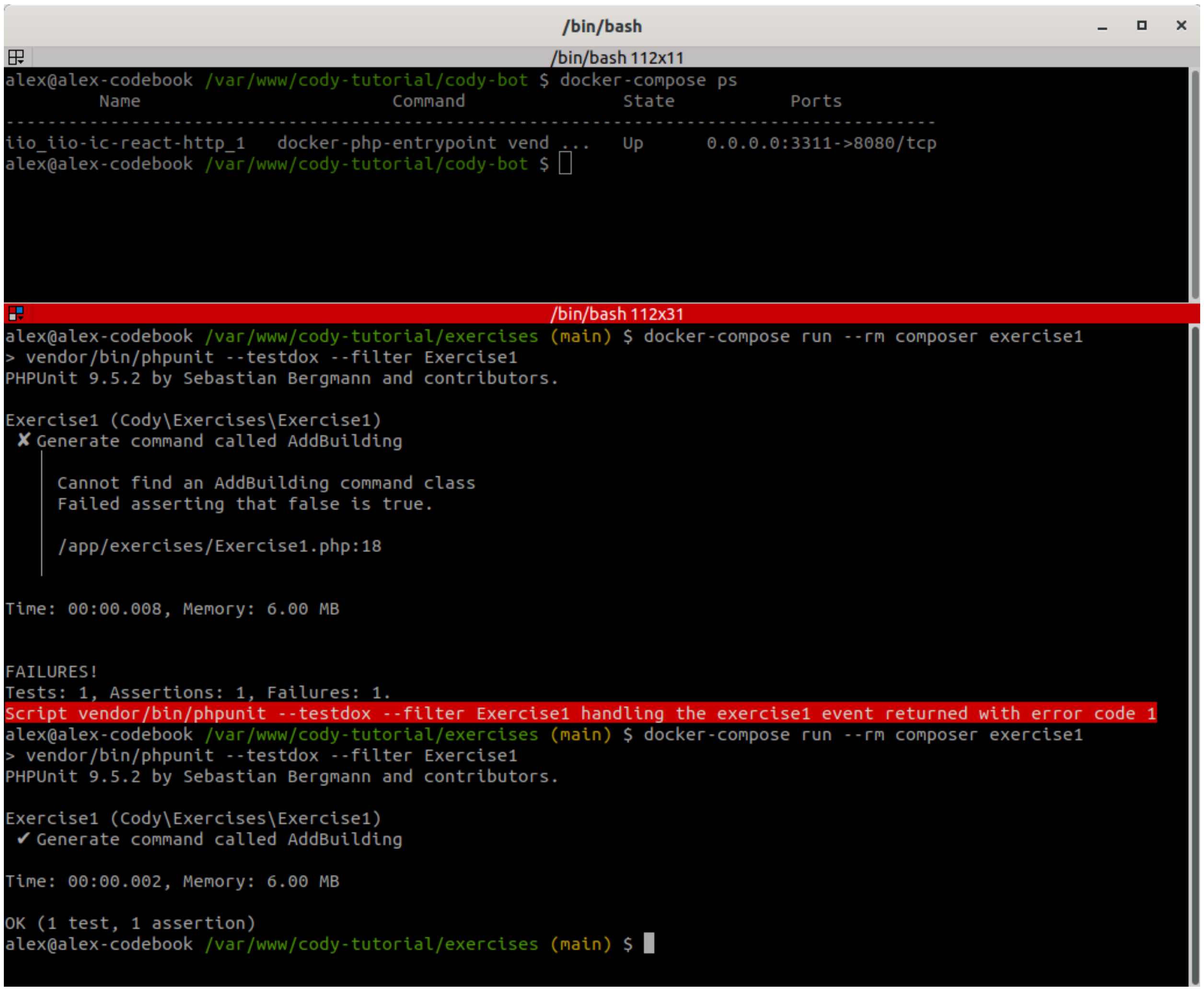
Pressing Ctrl+Q (Cmd+Q on Mac) closes the console.

To enable Cockpit integration use /cockpit https://localhost:<cockpit_port>

Awesome, it works! Rerun:

```
docker-compose run --rm composer exercise1
```

It should turn green now. The test verifies that a **Cody\Tutorial\Command\AddBuilding** has been generated by **Cody** and put into **cody-tutorial/exercises/src/Command/AddBuilding.php**.



```
alex@alex-codebook /var/www/cody-tutorial/cody-bot $ docker-compose ps
         Name                        Command               State           Ports
-----------------------------------------------------------------------------------------
iio_iio-ic-react-http_1   docker-php-entrypoint vend ...   Up      0.0.0.0:3311->8080/tcp
alex@alex-codebook /var/www/cody-tutorial/cody-bot $
```

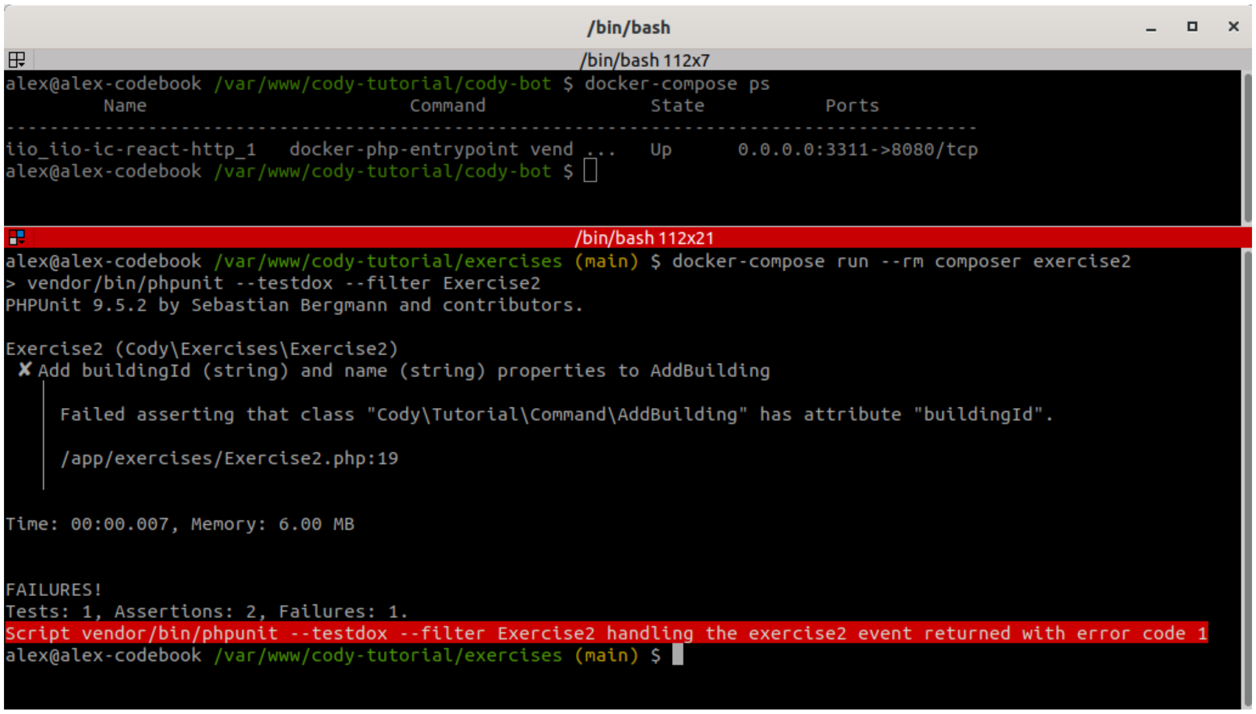
```
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $ docker-compose run --rm composer exercise1
> vendor/bin/phpunit --testdox --filter Exercise1
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

Exercise1 (Cody\Exercises\Exercise1)
 ✘ Generate command called AddBuilding
   │
   │ Cannot find an AddBuilding command class
   │ Failed asserting that false is true.
   │
   │ /app/exercises/Exercise1.php:18
   │

Time: 00:00.008, Memory: 6.00 MB


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
Script vendor/bin/phpunit --testdox --filter Exercise1 handling the exercise1 event returned with error code 1
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $ docker-compose run --rm composer exercise1
> vendor/bin/phpunit --testdox --filter Exercise1
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

Exercise1 (Cody\Exercises\Exercise1)
 ✔ Generate command called AddBuilding

Time: 00:00.002, Memory: 6.00 MB

OK (1 test, 1 assertion)
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $
```

In exercise I we implemented our first **Cody Hook** and registered it in `codyconfig.php`. The hook is called with information received from a prooph board event map and with a user defined context object, which can be used to pass configuration options (like a source path) to each hook. We can trigger the hook from prooph board by selecting an appropriate sticky on an event map and **Trigger Cody** from context menu.

Exercise II

To see what we have to do next, execute:

```
docker-compose run --rm composer exercise2
```



```
alex@alex-codebook /var/www/cody-tutorial/cody-bot $ docker-compose ps
        Name                        Command               State           Ports
-----------------------------------------------------------------------------------------
iio_iio-ic-react-http_1   docker-php-entrypoint vend ...   Up      0.0.0.0:3311->8080/tcp
alex@alex-codebook /var/www/cody-tutorial/cody-bot $
```

```
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $ docker-compose run --rm composer exercise2
> vendor/bin/phpunit --testdox --filter Exercise2
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

Exercise2 (Cody\Exercises\Exercise2)
 ✘ Add buildingId (string) and name (string) properties to AddBuilding
   │
   │ Failed asserting that class "Cody\Tutorial\Command\AddBuilding" has attribute "buildingId".
   │
   │ /app/exercises/Exercise2.php:19
   │

Time: 00:00.007, Memory: 6.00 MB


FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
Script vendor/bin/phpunit --testdox --filter Exercise2 handling the exercise2 event returned with error code 1
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $
```

We're asked to add a `buildingId` and a `name` property (both of type `string`) to the `AddBuilding` command. Now it would be easy to just open the class and add those two properties. But we should expand our code generation logic instead. This has some significant advantages:

- a prooph board event map acts as documentation
- one can generate contracts like an OpenAPI schema from Card Metadata
- it is easier to discuss and design new features or refactorings when such details are included on the event map

Expand Command Hook Logic

Ok let's use a simple code generator implementation first to understand the basics. In a later tutorial part we'll look at some useful libraries that provide abstractions for advanced use cases.

Change the command hook in `cody-tutorial/cody-bot/src/Hook/CommandHook.php` like this:

```
<?php
declare(strict_types=1);

namespace EventEngine\InspectioCody\Hook;

use EventEngine\InspectioCody\Board\BaseHook;
use EventEngine\InspectioCody\Http\Message\CodyResponse;
use EventEngine\InspectioCody\Http\Message\Response;
use EventEngine\InspectioGraphCody\Node;
use function is_array;
use function json_decode;

final class CommandHook extends BaseHook //BaseHook provides some helper methods like writeFile()
{
    /**
     * @param  Node     $command Information about Command sticky received from prooph board
event map
```

```
     * @param  object   $context Context object populated in codyconfig.php
     * @return CodyResponse      Response sent back to prooph board, shown in Cody Console
     */
    public function __invoke(Node $command, object $context): CodyResponse
    {
        $commandName = $command->name();
        $commandFile = $commandName . '.php';
        $commandPath = $context->path . '/Command/' . $commandFile;

        // Get raw metadata info about command and convert it to associative array
        $metadata = json_decode($command->metadata(), true);

        $properties = "";

        if(is_array($metadata)) {
            // @TODO: some structural validation might be a good idea here
            foreach ($metadata as $property => $type) {
                // Each property-type-pair is appended to properties string
                // and resulting string is inserted in the class template below
                $properties.= "    public $type $$property;\n";
            }
        }


        $code = <<<CODE
        <?php
        declare(strict_types=1);

        namespace Cody\Tutorial\Command;

        class $commandName
        {
        $properties
        }
        CODE;

        $this->writeFile($code, $commandPath);

        return Response::fromCody(
            "Command \"{$commandName}\" generated",
            ["Command written to {$commandPath}"]
        );
    }
}
```

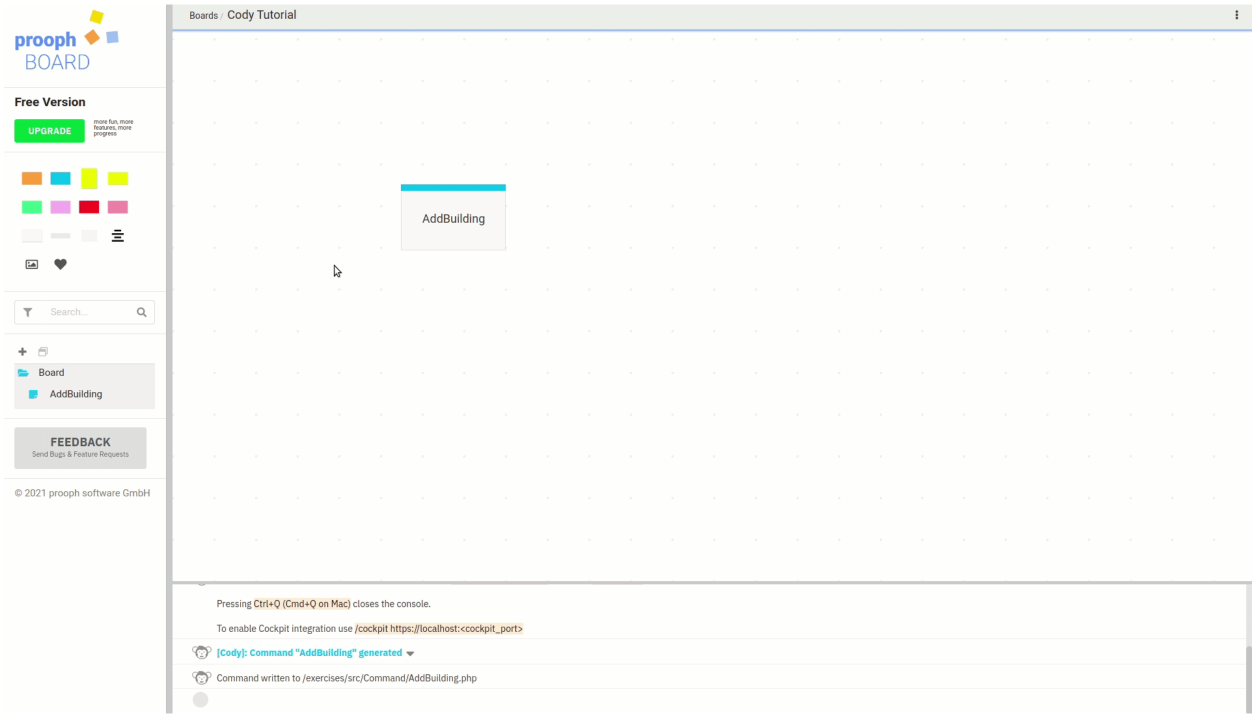
Set Command Metadata in prooph board

Now we can switch to prooph board and set the following metadata in JSON format to the `AddBuilding` command:

```
{
  "buildingId": "string",
  "name": "string"
}
```

💡 *Metadata can be set by opening the Metadata Sidebar (choose Metadata from top menu) and selecting the appropriate card or sticky note. Metadata changes are saved automatically.*

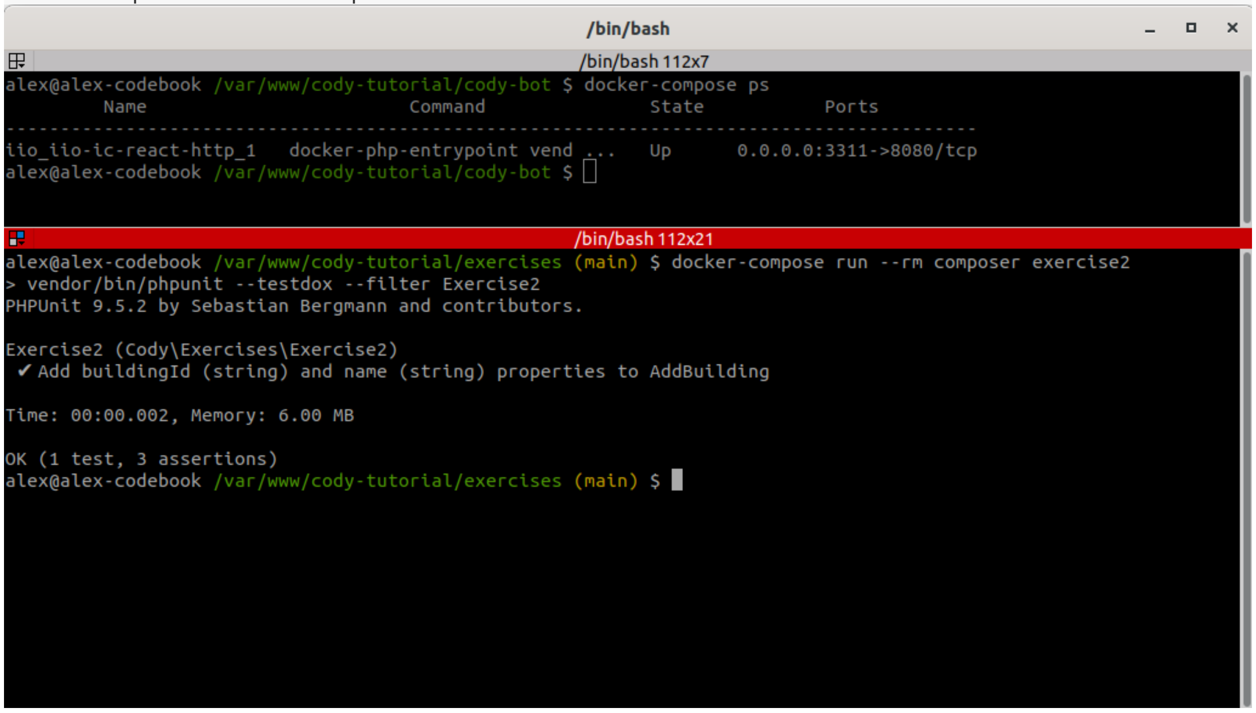
Once metadata is set we can trigger Cody again …



```
Pressing Ctrl+Q (Cmd+Q on Mac) closes the console.
To enable Cockpit integration use /cockpit https://localhost:<cockpit_port>
[Cody]: Command "AddBuilding" generated
Command written to /exercises/src/Command/AddBuilding.php
```

… and validate the result by executing:

```
docker-compose run --rm composer exercise2
```



```
alex@alex-codebook /var/www/cody-tutorial/cody-bot $ docker-compose ps
         Name                          Command               State            Ports
-------------------------------------------------------------------------------------------
iio_iio-ic-react-http_1   docker-php-entrypoint vend ...   Up      0.0.0.0:3311->8080/tcp
alex@alex-codebook /var/www/cody-tutorial/cody-bot $
```

```
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $ docker-compose run --rm composer exercise2
> vendor/bin/phpunit --testdox --filter Exercise2
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

Exercise2 (Cody\Exercises\Exercise2)
 ✔ Add buildingId (string) and name (string) properties to AddBuilding

Time: 00:00.002, Memory: 6.00 MB

OK (1 test, 3 assertions)
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $
```

Cards on a prooph board event map can have additional information set as metadata. By default metadata is stored in JSON format. Metadata itself is schemaless, meaning users are free to define any structure. The structure can be defined and even be type checked using [Metadata]

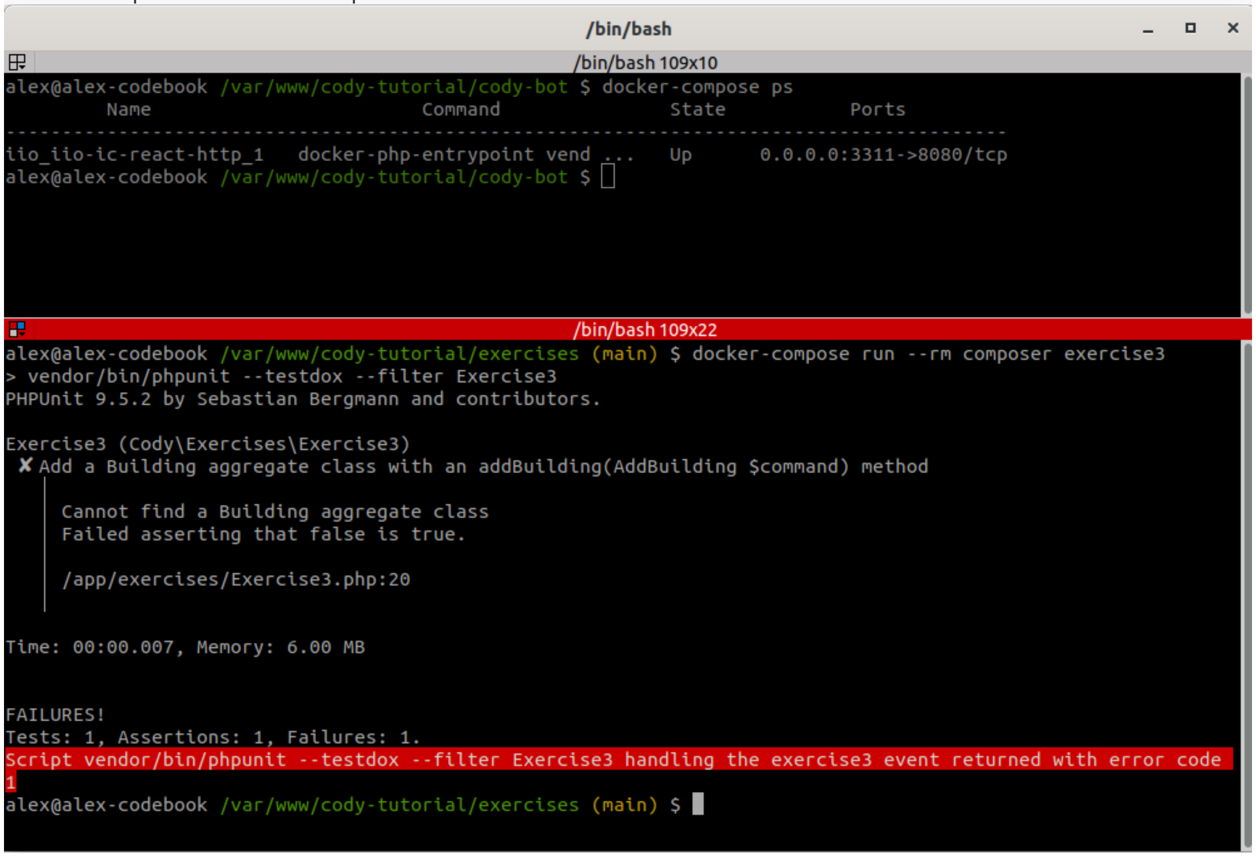
Templates. In a Cody hook you have access to raw metadata and use it for advanced code generation.

Exercise III

One last basic concept is missing to complete the picture. On a prooph board event map you model message flows, behavior, business processes. Different objects interact with each other by exchanging messages like commands and events. A Cody Hook can make use of this information to assist developers by generating all required glue code if not a fully working implementation (depends on complexity).

We start again by looking at the failing test case for exercise III:

```
docker-compose run --rm composer exercise3
```

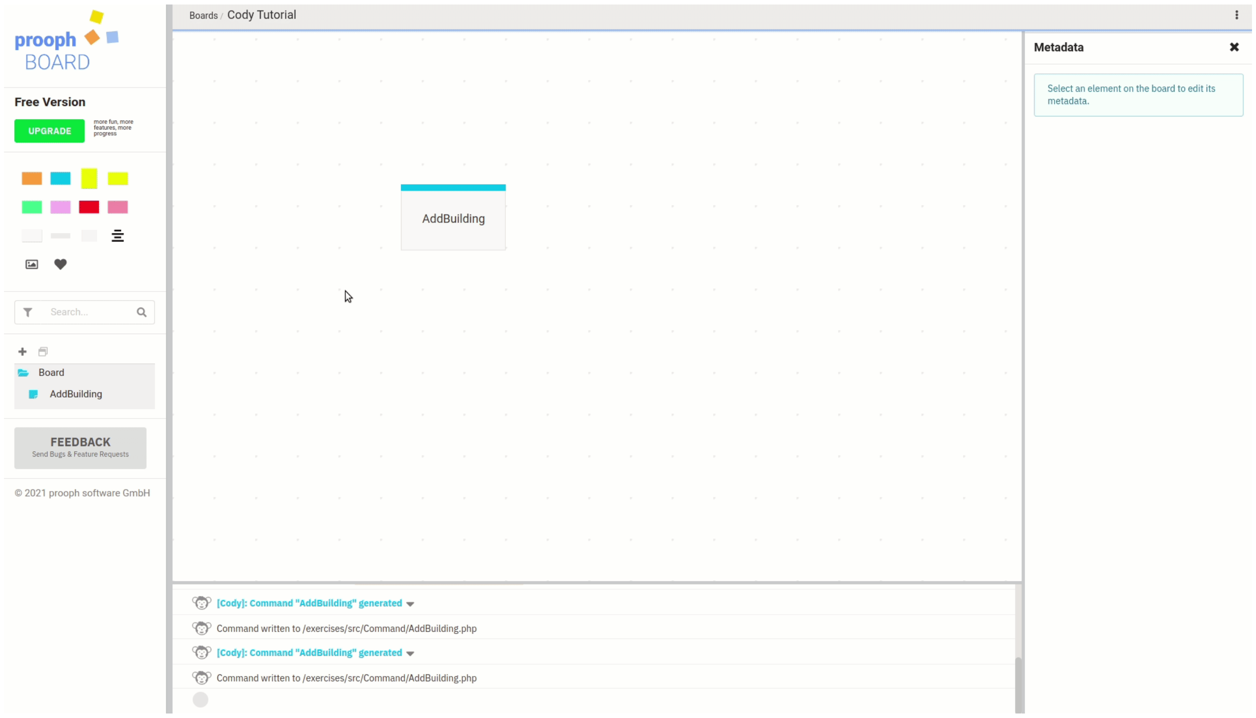


```
/bin/bash
/bin/bash 109x10
alex@alex-codebook /var/www/cody-tutorial/cody-bot $ docker-compose ps
        Name                        Command               State            Ports
-----------------------------------------------------------------------------------------
iio_iio-ic-react-http_1   docker-php-entrypoint vend ...   Up      0.0.0.0:3311->8080/tcp
alex@alex-codebook /var/www/cody-tutorial/cody-bot $

/bin/bash 109x22
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $ docker-compose run --rm composer exercise3
> vendor/bin/phpunit --testdox --filter Exercise3
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

Exercise3 (Cody\Exercises\Exercise3)
 ✘ Add a Building aggregate class with an addBuilding(AddBuilding $command) method
   │
   │ Cannot find a Building aggregate class
   │ Failed asserting that false is true.
   │
   │ /app/exercises/Exercise3.php:20
   │

Time: 00:00.007, Memory: 6.00 MB


FAILURES!
Tests: 1, Assertions: 1, Failures: 1.
Script vendor/bin/phpunit --testdox --filter Exercise3 handling the exercise3 event returned with error code
1
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $
```

Ok, this time we're asked to generate an aggregate class named `Building` with a method handling the `AddBuilding` command from previous exercises.

Aggregate Hook

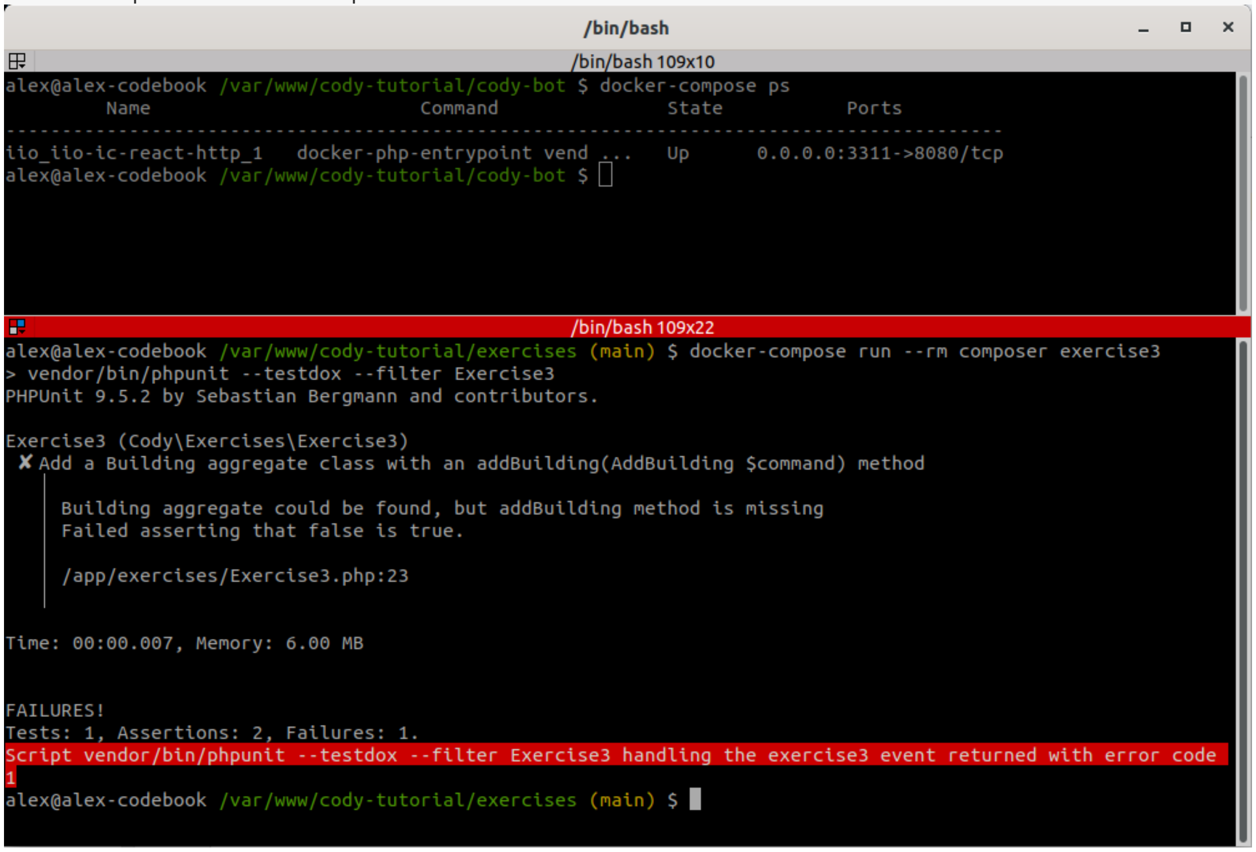
In Exercise I you've learned how to create a Command Hook and register it in `codyconfig.php`. Do the same for an `AggregateHook` and let the hook generate an empty aggregate class. The class should be written to the directory `exercises/src/Aggregate`.
If you think the hook is ready, switch to prooph board, add an aggregate card with label `Building` on the event map and trigger Cody.


prooph BOARD
Free Version
UPGRADE
more fun, more features, more progress
Search...
Board
AddBuilding
FEEDBACK
Send Bugs & Feature Requests
© 2021 prooph software GmbH
Boards / Cody Tutorial
AddBuilding
Metadata
Select an element on the board to edit its metadata.

```
[Cody]: Command "AddBuilding" generated
Command written to /exercises/src/Command/AddBuilding.php
[Cody]: Command "AddBuilding" generated
Command written to /exercises/src/Command/AddBuilding.php
```

Did it work? You can verify the result by executing:

```
docker-compose run --rm composer exercise3
```



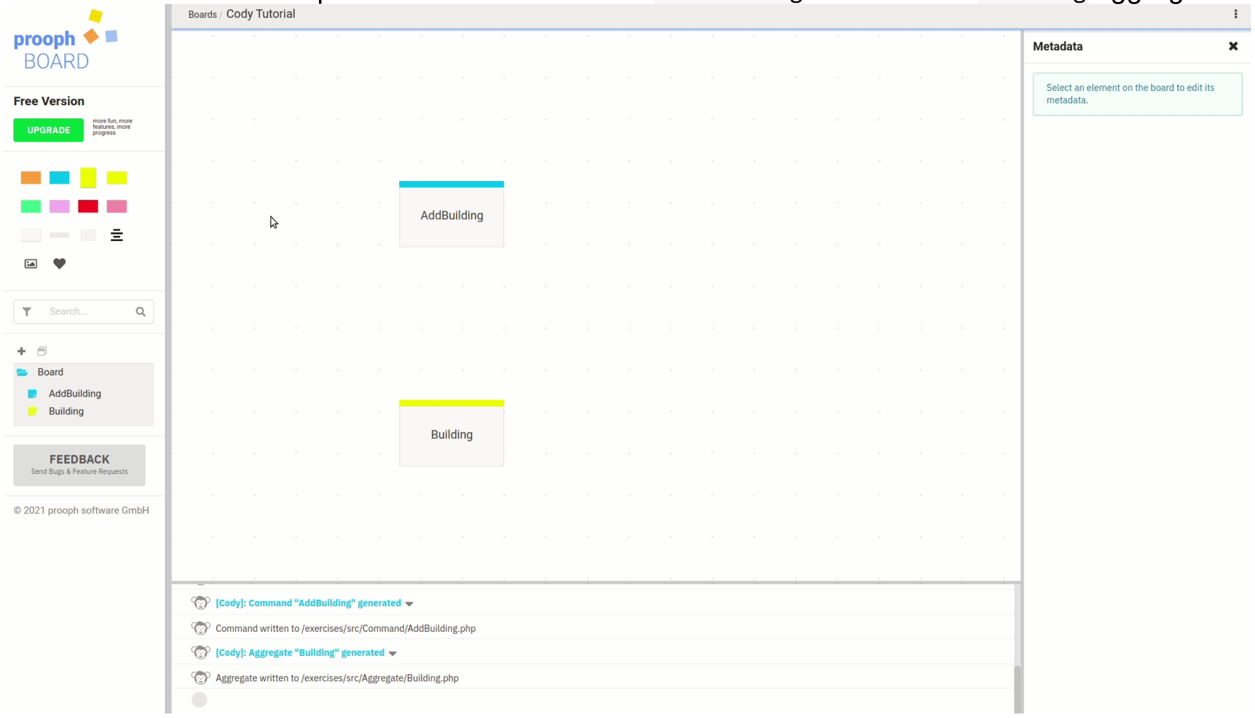
```
alex@alex-codebook /var/www/cody-tutorial/cody-bot $ docker-compose ps
         Name                        Command               State            Ports
-----------------------------------------------------------------------------------------
iio_iio-ic-react-http_1   docker-php-entrypoint vend ...   Up      0.0.0.0:3311->8080/tcp
alex@alex-codebook /var/www/cody-tutorial/cody-bot $
```

```
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $ docker-compose run --rm composer exercise3
> vendor/bin/phpunit --testdox --filter Exercise3
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

Exercise3 (Cody\Exercises\Exercise3)
 ✘ Add a Building aggregate class with an addBuilding(AddBuilding $command) method
   │
   │ Building aggregate could be found, but addBuilding method is missing
   │ Failed asserting that false is true.
   │
   │ /app/exercises/Exercise3.php:23
   │

Time: 00:00.007, Memory: 6.00 MB


FAILURES!
Tests: 1, Assertions: 2, Failures: 1.
Script vendor/bin/phpunit --testdox --filter Exercise3 handling the exercise3 event returned with error code
1
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $
```

Test case is still failing but the error message has changed. `Building::addBuilding(AddBuilding $command)` method is still missing.

Event Map Connections

Go back to the event map and draw an arrow from `AddBuilding` command to `Building` aggregate.



Select an element on the board to edit its metadata.

```
[Cody]: Command "AddBuilding" generated
Command written to /exercises/src/Command/AddBuilding.php
[Cody]: Aggregate "Building" generated
Aggregate written to /exercises/src/Aggregate/Building.php
```

The connection we've just drawn can be read in a hook. Here is the `Node` interface passed to a hook:

```php
<?php

/**
 * @see       https://github.com/event-engine/php-inspectio-graph-cody for the canonical source
repository
 * @copyright https://github.com/event-engine/php-inspectio-graph-cody/blob/master/COPYRIGHT.md
 * @license   https://github.com/event-engine/php-inspectio-graph-cody/blob/master/LICENSE.md MIT
License
 */

declare(strict_types=1);

namespace EventEngine\InspectioGraphCody;

interface Node
{
    public function id(): string;

    public function name(): string;

    public function type(): string;

    public function tags(): iterable;

    public function isLayer(): bool;

    public function isDefaultLayer(): bool;
    public function parent(): ?Node;

    /**
     * @return Node[]
     */
    public function children(): iterable;

    /**
     * @return Node[]
     */
    public function sources(): iterable;

    /**
     * @return Node[]
     */
    public function targets(): iterable;

    public function metadata(): ?string;
}
```

We already worked with `name()` and `metadata()`. From a `Node` you can follow connections using `sources()` and `targets()`. In our case the `AddBuilding` command node has the `Building` aggregate referenced as a target and from `Building` point of view `AddBuilding` command is a source.

One important thing to note here is that a hook gets only access to directly connected nodes, even if those nodes have further connections! This avoids circular reference problems and keeps payload exchanged between prooph board and Cody small. However, it's possible to request a full board sync and build up a node map in Cody. But this is an advanced topic, that should be covered later.

💡 *`Node` interface also provides information about `parent()` and `children()` which are useful when grouping cards in a frame*

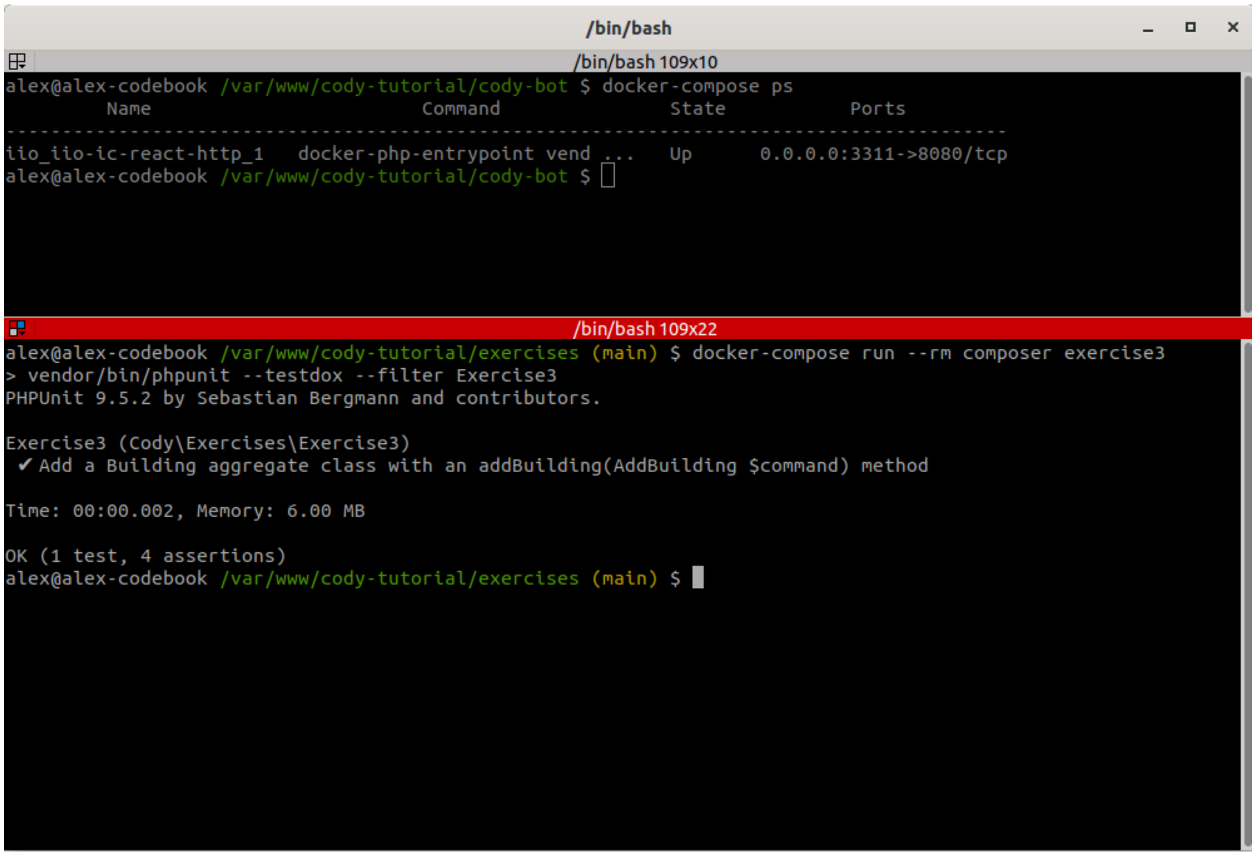
Enough theory! Let's see it in action. Update your `AggregateHook` with this version:

```php
<?php
declare(strict_types=1);

namespace EventEngine\InspectioCody\Hook;

use EventEngine\InspectioCody\Board\BaseHook;
use EventEngine\InspectioCody\Http\Message\CodyResponse;
use EventEngine\InspectioCody\Http\Message\Response;
use EventEngine\InspectioGraph\VertexType;
use EventEngine\InspectioGraphCody\Node;
use LogicException;
use function lcfirst;

final class AggregateHook extends BaseHook
{
    /**
     * @param  Node     $aggregate Information about aggregate sticky received from prooph board
event map
     * @param  object   $context Context object populated in codyconfig.php
     * @return CodyResponse      Response sent back to prooph board, shown in Cody Console
     */
    public function __invoke(Node $aggregate, object $context): CodyResponse
    {
        $aggregateName = $aggregate->name();
        $aggregateFile = $aggregateName . '.php';
        $aggregatePath = $context->path . '/Aggregate/' . $aggregateFile;
        $commandHandlingMethod = "";
        $includes = "";
        $command = null;

        foreach ($aggregate->sources() as $source) {
            if($source->type() === VertexType::TYPE_COMMAND) {
                if($command) {
                    throw new LogicException(
                        "Aggregate $aggregateName is connected to more than one command"
                    );
                }

                $command = $source;
            }
        }

        if($command) {
            $includes.= "use Cody\Tutorial\Command\\{$command->name()};\n";

            $methodName = lcfirst($command->name());

            $commandHandlingMethod =
                "public static function $methodName({$command->name()} \$command): void {}";
        }

        $code = <<<CODE
        <?php
        declare(strict_types=1);

        namespace Cody\Tutorial\Aggregate;

        $includes

        class $aggregateName
        {
            $commandHandlingMethod
        }
        CODE;

        $this->writeFile($code, $aggregatePath);

        return Response::fromCody(
            "Aggregate \"{$aggregateName}\" generated",
            ["Aggregate written to {$aggregatePath}"]
        );
    }
}
```

The implementation should be self explanatory. We included a logical validation that an aggregate card should only have one command as a source. If more than one is found an exception is thrown. Exceptions are caught by Cody and sent back to prooph board as an error response. This way you can harden your hooks and validate event maps according to your own rules.

Trigger Cody with the `Building` aggregate card in prooph board and run exercise III once again:

```
docker-compose run --rm composer exercise3
```



```
/bin/bash 109x10
alex@alex-codebook /var/www/cody-tutorial/cody-bot $ docker-compose ps
         Name                          Command                State            Ports
------------------------------------------------------------------------------------------
iio_iio-ic-react-http_1   docker-php-entrypoint vend ...   Up      0.0.0.0:3311->8080/tcp
alex@alex-codebook /var/www/cody-tutorial/cody-bot $
```

```
/bin/bash 109x22
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $ docker-compose run --rm composer exercise3
> vendor/bin/phpunit --testdox --filter Exercise3
PHPUnit 9.5.2 by Sebastian Bergmann and contributors.

Exercise3 (Cody\Exercises\Exercise3)
 ✔ Add a Building aggregate class with an addBuilding(AddBuilding $command) method

Time: 00:00.002, Memory: 6.00 MB

OK (1 test, 4 assertions)
alex@alex-codebook /var/www/cody-tutorial/exercises (main) $
```

This exercise introduced the last basic building block: **Connections between cards**. Depending on arrow direction on prooph board a connected card appears either in the `sources` or `targets` collection of the `Node` passed to a hook. It's important to keep in mind that a hook only has access to directly connected cards.

Exercise IV

Exercise IV coming soon!

Meanwhile checkout these libraries for advanced code generation:

- PHP prooph board Graph Cody: High-level abstraction for Cody nodes
- Event Engine Code Generator: High-level abstraction for generating Event Engine code

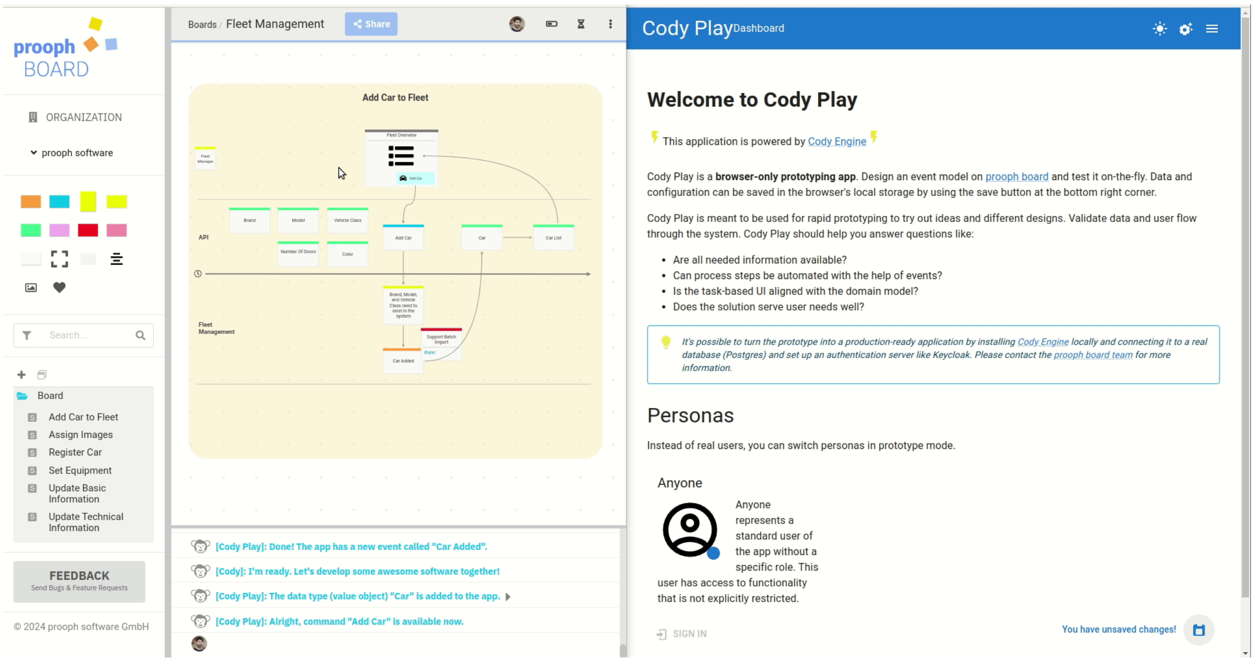
And have a look at the example boards to learn more about working on prooph board:

- Annotated prooph board Example
- Event Storming - The Picture
- Event Modeling Example

A PLAYGROUND FOR YOUR DESIGN

"We strive for a really fast **feedback-loop** to keep cross-functional teams in the creativity mode as long as possible."

– *Alexander Miertsch, Founder and CEO of prooph software*



[Cody Play]: Done! The app has a new event called "Car Added".
[Cody]: I'm ready. Let's develop some awesome software together!
[Cody Play]: The data type (value object) "Car" is added to the app.
[Cody Play]: Alright, command "Add Car" is available now.

Welcome to Cody Play

This application is powered by Cody Engine

Cody Play is a **browser-only prototyping app**. Design an event model on prooph board and test it on-the-fly. Data and configuration can be saved in the browser's local storage by using the save button at the bottom right corner.

Cody Play is meant to be used for rapid prototyping to try out ideas and different designs. Validate data and user flow through the system. Cody Play should help you answer questions like:

- Are all needed information available?
- Can process steps be automated with the help of events?
- Is the task-based UI aligned with the domain model?
- Does the solution serve user needs well?

It's possible to turn the prototype into a production-ready application by installing Cody Engine locally and connecting it to a real database (Postgres) and set up an authentication server like Keycloak. Please contact the prooph board team for more information.

Personas

Instead of real users, you can switch personas in prototype mode.

Anyone

Anyone represents a standard user of the app without a specific role. This user has access to functionality that is not explicitly restricted.

[Tutorial](#)	[Docs](#)	[Playshots](#)

[Tutorial](#)

CODY PLAY TUTORIAL

Let's bring our [Event Modeling Design](#) to life with the help of Cody Play.

In this first-steps-tutorial you'll learn the basics of working with the prototyping tool Cody Play. It is seamlessly integrated in prooph board. We will continue to work on the Fleet Management application, that we started to design in the [Event Modeling on prooph board](#) step-by-step guide.

Start Cody Play

Usually, you would continue with the existing design and enhance it with Cody instructions. However, for the tutorial we should start with a fresh board and focus on one concept at a time to keep cognitive load on a consistent level.

As a first step, we need a new prooph board as our work surface, and also a clean Cody Play application. You can do the entire tutorial in the free version of prooph board (no registration required), that ships with full Cody Play support.

1. Open prooph board
2. Add a new board and name it: **Fleet Management**
3. Start a new Cody Play session (Top Menu -> Cody Play -> Start a new Session)

Cody Play is a read-to-use prototyping application. It runs completely in your browser. You don't need to install anything, and all data is only stored in your browser's local storage. You can save Playshots to back up your work and share it with anybody who has access to the connected prooph board (only possible in the paid version of prooph board).

CODY ENGINE INTRO

Looking for a way to bridge the gap between Event Modeling and prototyping or even production-ready solutions?

Cody is a barbone system. Cody Engine builds on top of it to provide a ready-to-use prototyping and application framework.

Built on the shoulders of giants

Cody Engine is a **ready-to-use** Open Source solution to transform an Event Map into a prototype and if you want also into a production-ready system.

To make this possible, we decided to go all-in with TypeScript. TypeScript gives us a single platform to develop a web-based **Client-Server-Application**, well suited for an information management system.

Open Source

You can find the Cody Engine project on GitHub.

Questions, feedback, ideas and contributions are all welcome! You can use the **Cody Engine Issue Tracker** to get in touch.

Installation

To install and run Cody Engine you need Git and Node.js installed on your local computer.

Head over to the Cody Engine README for installation instructions.

Troubleshooting

Sometimes code generation causes a “Compiler Error” that is then displayed in the Cody Engine frontend. If that happens, try to restart Cody Engine first and then reload the app in the browser. If the error is still shown, something is broken, and you have to investigate further. If you need help, don’t hesitate to open an issue: Issue.

You can restart Cody Engine by stopping the process in the terminal with `Ctrl+C`, and starting it again with `npm run serve`.

INFORMATION SCHEMA

Cody Engine uses JSON Schema to describe the structure of Commands, Events, Queries and Information like state, state list and value objects.

BOARD API

You can programmatically access and modify a board.

API Key

For each board that you want to access via API, you need to generate an API Key. You can do this in the Board Settings.

You need to be a board admin to be able to access the Board Settings.

Retrieve the client secret from the server and include it as `X-Auth-Secret` header in every request to the API.

Board Agent

To modify elements on a board via API, you have to send tasks to a Board Agent. The tasks are queued and executed directly on the board as soon as someone opens it (if it is not already open).

View Spec in Swagger Editor on editor.swagger.io

Please note: you cannot use the swagger editor to test the API due to CORS restrictions, but you can copy the cUrl commands from the editor and run them from a console.

WORKING WITH CARDS

Modeling on prooph board is mainly done with colored Cards. The color codes are inspired by Event Storming sticky colors. This makes it easier to switch between digital and onsite Event Stormings.

Adding a Card

Cards can be dragged from the left sidebar. A legend helps with mapping colors to different concepts like events, commands and aggregates.

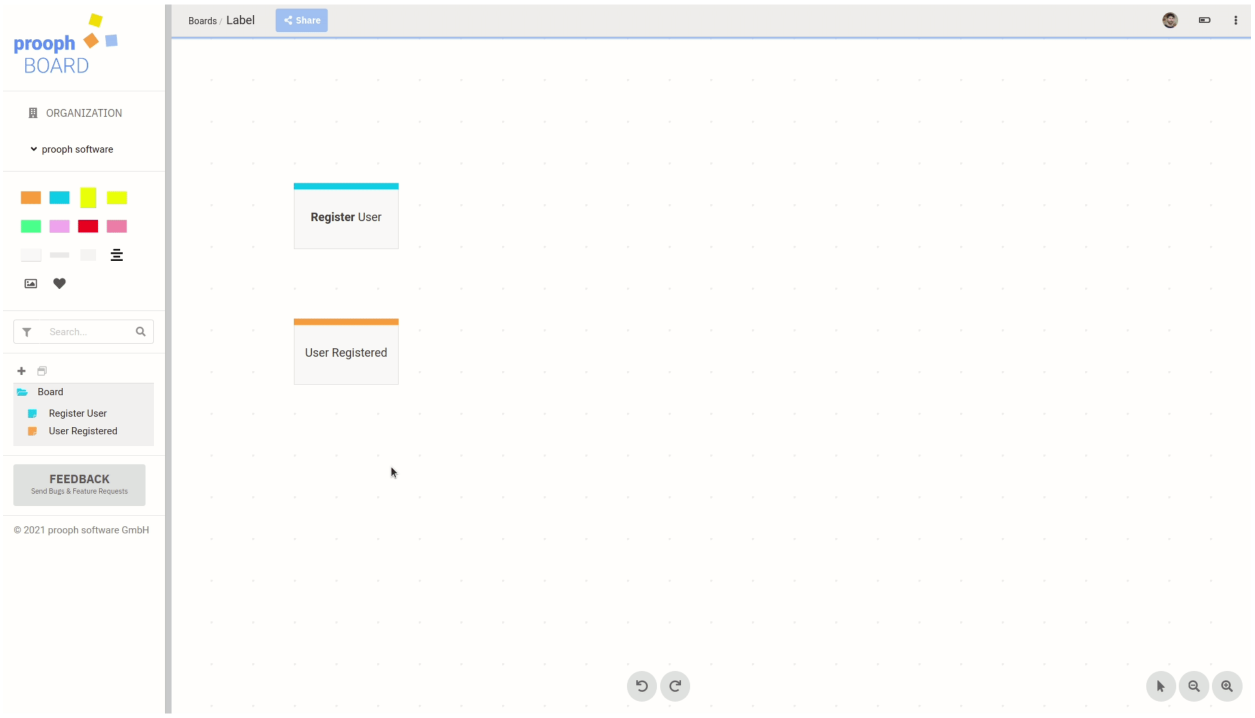
Writing on Cards

When writing on a card a toolbar shows up with text formatting options like changing text size and color, insert links and horizontal rulers.

Horizontal rulers

A horizontal ruler on a Card has special meaning. It divides the name of the Card from it's details. The name of the card is used to reference it in the tree view and for history entries. When you select a card and press **Ctrl+F** (Cmd+F on Mac), the Card is looked up in the tree view using it's name. This way you can quickly find Cards with the same name on a large board.

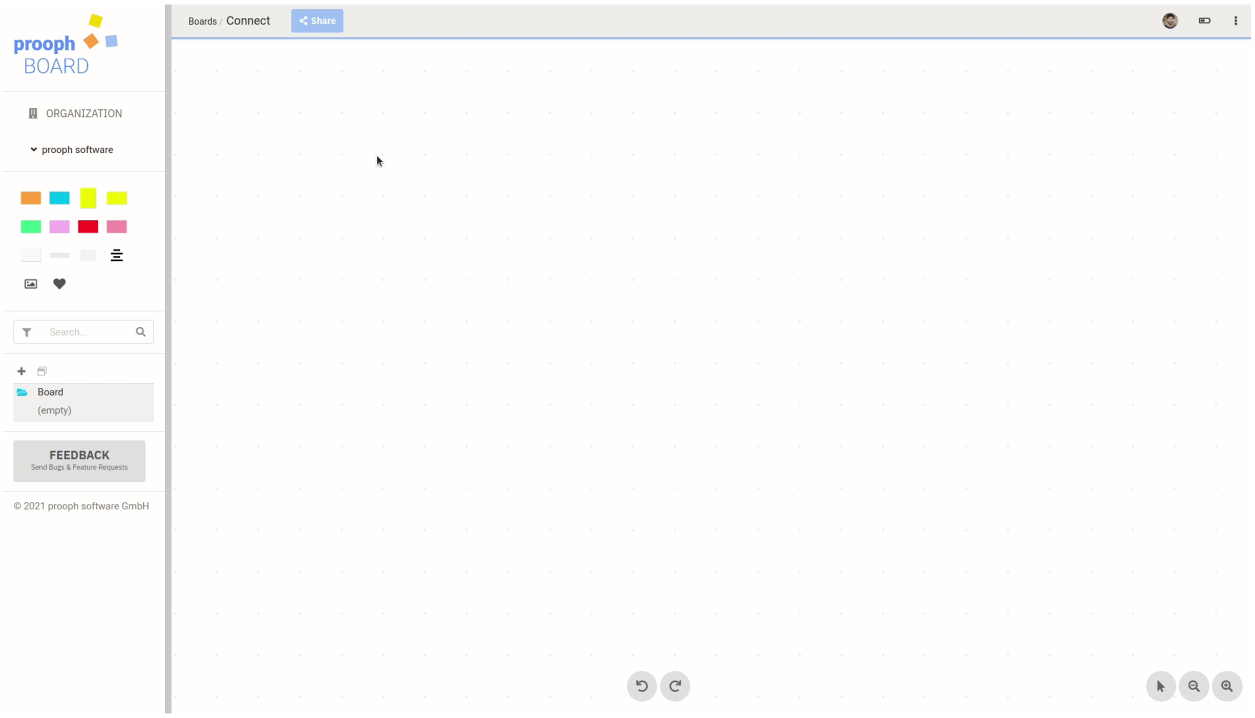
After a horizontal ruler text alignment switches from **centered** to **left aligned** like shown in the example:



Autosize

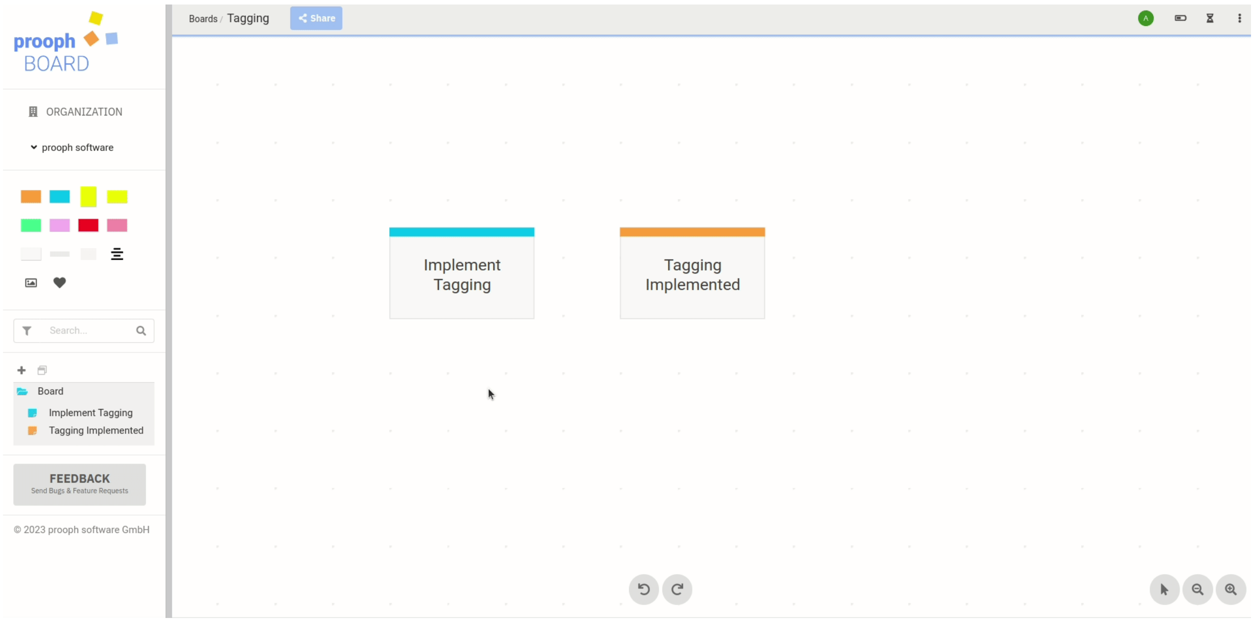
Card size changes automatically according to the text. Wordwrap is used for automatic line breaks. This means, that as long as words are not longer than the currently available width, only the height of a Card will change. You can manually resize a Card to adjust the width. If you press **Ctrl-Key** (Cmd on Mac) while resizing, the aspect ratio of the card is fixed.

Connecting Cards



Tagging

You can use tags (words prefixed with #) on cards. They turn into clickable links. A click sets the tag as filter in the Tree View.



Using **!#** in the search filters all elements that don’t have a specific tag assigned.

CODY SUGGESTIONS

https://wiki.prooph-board.com/board_workspace/Cody-Suggestions.html

Cody is deeply integrated on prooph board. It does not only translate designs into working code, but also helps you with the design itself.

When Event Modeling Mode is enabled, Cody appears within a Slice to suggest cards. If Event Modeling is turned off, Cody switches the suggestion style to Continuous Event Storming. The latter was the default format on prooph board, before we fully integrated Event Modeling. If you prefer this style, Cody will continue to help you with it.

Turn Off

You can disable Cody Suggestions in the board settings.

https://wiki.prooph-board.com/board_workspace/Cody-Suggestions.html

EVENT MODELING MODE

By default, every prooph board has Event Modeling support enabled.

The main feature of that mode is a special Slice element, that you can pull from the sidebar.

A Slice represents the smallest coherent unit of the software, e.g. one action performed by a user or one automated task step. Event Modeling organizes the software design in swim lanes. Slices on prooph board reflect this structure, so each Slice can have multiple swim lanes. Learn more about Slices on the **Event Modeling on prooph board** page.

Turn Off

You can turn off Event Modeling support in the board settings.
https://wiki.prooph-board.com/assets/video/board_workspace/event-modeling-mode-turn-off.mp4

USE FRAMES FOR GROUPING

Frames are used to group cards and other elements.

prooph board supports 3 frame types:

Context / Module Frame

Depending on the scope of the board, this frame refers to a team, a DDD Bounded Context, a sub-domain, a business capability, a software module or a (micro)service. It's meant to group information, concepts and design that belong together. And it's the next level in the tree view after layers.

Event Modeling Slice

By default, Event Modeling Mode is turned on, which gives you access to Slices instead of Feature frames (see below).

A Slice is similar to a Feature frame in the way that it groups elements of a specific feature. The main difference is the structure of a Slice. It comes with swim lanes for user roles and system modules.

You can add additional lanes by hovering the topmost or lowest lane label and click the plus icon. Doing this on the first Slice of the board also modifies the Slice blueprint. This means, that every new Slice pulled from the sidebar will have the additional lanes, too.

You can also mark a different Slice on the board as **Slice Blueprint** in case you add new lanes later in the design. And you can copy new lanes to all existing Slices by choosing the action in the Slice context menu (see video).

Slices are the second level below layers and contexts in the tree view.

Feature Frame

Feature Frames are the alternative to Slices (see above), when Event Modeling Mode is turned off. You can also use them to group elements. Features are the second level below layers and contexts in the tree view.

Bird View

When zooming out the content of a frame is collapsed and its label scaled. This provides you with a bird view to keep an overview on large boards.

The two frame types collapse at different zoom levels:

Context / Module: collapses later than **Feature / Slice** (later = more zoomed out).

The bird view can be disabled for a frame. Open the context menu (right mouse click / tap and hold on touch device) and toggle **Bird View -> Hide Details**. If you want turn off bird view for all frames, please see Lite Mode.

Disabling the collapse functionality can be useful if you want to use frames as swim lanes or sections of a canvas.

Move Frames

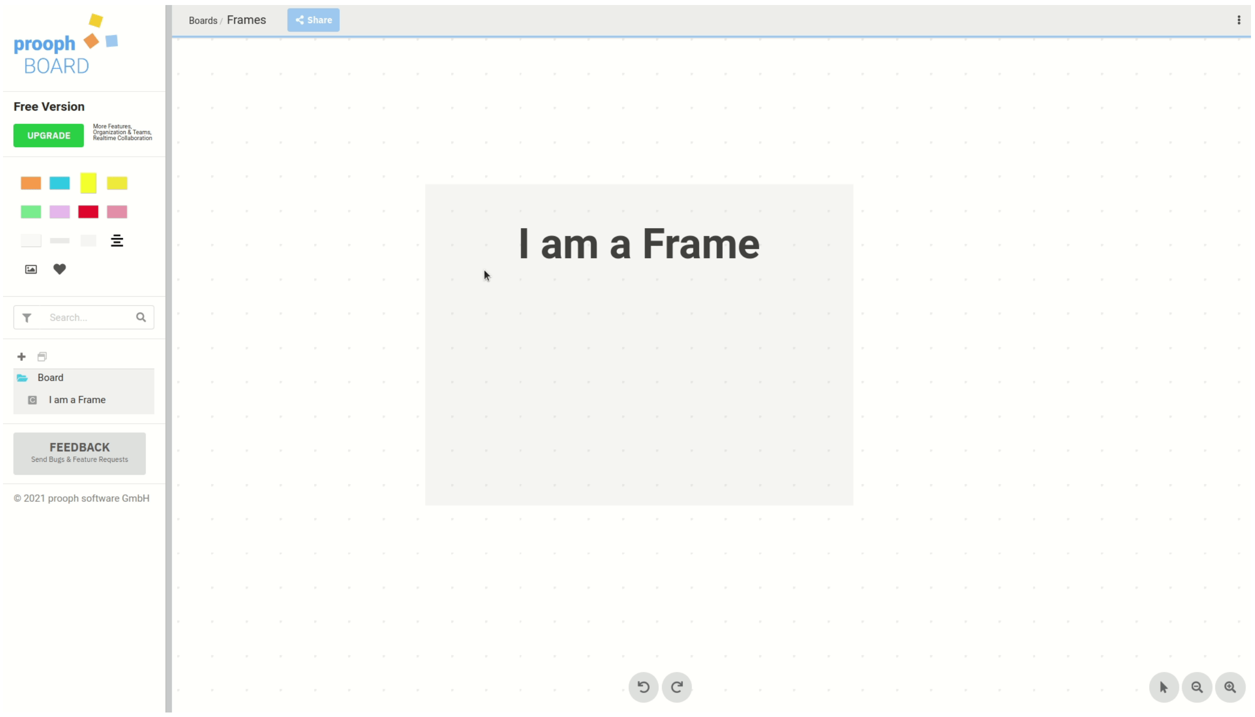
To be able to navigate on the board even if a frame fills the entire screen, they only have a small drag zone. Otherwise you would always move the frame instead of moving around on the board.

As you can see in the GIF, the border of a frame gets highlighted when the mouse pointer is close to it. You need to select the frame, move your mouse next to border, so that the pointer turns into a hand and only then you’re able to move the frame. Don’t worry if it is a bit tricky at the beginning. You should get used to it quickly and it really helps to navigate on the board.

Change Shape

Frames can have different shapes e.g. to represent different kinds of modules or contexts.

Chose a shape from **context menu -> Style**


prooph BOARD
Free Version
UPGRADE
More Features, Organization & Teams, Realtime Collaboration
Search...
Board
I am a Frame
FEEDBACK
Send Bugs & Feature Requests
© 2021 prooph software GmbH
Boards / Frames
Share
I am a Frame

KEYMAP

prooph boards comes with some handy keyboard shortcuts to quickly add, modify and arrange elements on a board.

Navigation

This keymap applies when no element is selected on the board workspace:

Key(s)	Description
Arrow keys	Move around on the workspace, keep pressed for faster

	move
Shift + arrow keys	Move fast on the workspace, keep pressed for even faster move
Ctrl/Cmd + +/- (Numpad)	Zoom in/out
Ctrl/Cmd + F	Focus search in sidebar
Ctrl/Cmd + V	Paste copied or cut elements
Tab	Select first element on the board that is currently in view port
Ctrl/Cmd + Shift + Backspace	Go back to previous position on the board

Single Selection

This keymap applies for a single selected element:

Key(s)	**Description**
Arrow keys	Move element, keep pressed for faster move
Shift + arrow keys	Move element fast, keep pressed for even faster move
Ctrl/Cmd + arrow keys	Resize element, keep pressed for bigger resize steps
Ctrl/Cmd + Shift + arrow keys	Resize element with bigger resize steps
Ctrl/Cmd + F	Search similar elements in the sidebar
Ctrl/Cmd + M	Show metadata of element

Ctrl/Cmd + G	Trigger Cody with element
Ctrl/Cmd + C	Copy selection
Tab	Select next element
Del/Backspace	Delete Element
Enter	Edit element label
Esc	Deselect element
Alt + E	Change Sticky Type to Event
Alt + C	Change Sticky Type to Command
Alt + A	Change Sticky Type to Aggregate
Alt + R	Change Sticky Type to Role
Alt + D	Change Sticky Type to Document
Alt + S	Change Sticky Type to External System
Alt + H	Change Sticky Type to Hot Spot
Alt + P	Change Sticky Type to Policy
Alt + U	Change Sticky Type to UI / API
Ctrl/Cmd + Shift + F	Move to Front
Ctrl/Cmd + Shift + B	Move to Back
Alt + Shift + Page Up	Select parent element

Alt + Shift + Page Down	Select first child element
Alt + Shift + Arrow key	Copy and connect selected element
Ctrl/Cmd + Enter	Duplicate element

Multi Selection

This keymap applies for a group of selected elements:

Key(s)	Description
Arrow keys	Move elements, keep pressed for faster move
Shift + arrow keys	Move element fasts, keep pressed for even faster move
Esc	Deselect elements
Ctrl/Cmd + G	Trigger Cody with elements
Ctrl/Cmd + Shift + L	Align elements left
Ctrl/Cmd + Shift + R	Align elements right
Ctrl/Cmd + Shift + C	Align elements center
Ctrl/Cmd + Shift + U	Align elements up
Ctrl/Cmd + Shift + D	Align elements down
Ctrl/Cmd + Shift + M	Align elements middle

Text Editing

This keymap applies during text editing of an element’s label:

Key(s)	Description
Esc	Finish Editing
Ctrl/Cmd + Shift + L	Insert Link
Ctrl/Cmd + Shift + H	Insert Horizontal Ruler

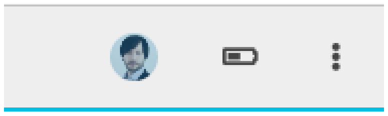
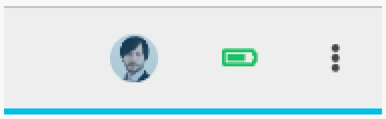
LITE MODE

If you encounter performance problems on large boards or during video calls, you can turn on the Lite Mode. This will disable zoom effects like hiding elements or scaling the font size of Bounded Context and Feature frames.

Effects turned off means your browser has less expensive tasks to do. This saves CPU and therefore also battery time.

Disadvantage of course is, that you no longer get the high level overview. Use the navigation tree and search in the left sidebar to still find the information you’re looking for.

Switching between Normal and Lite Mode can be done in the workspace top menu. It’s the battery icon next to your avatar. The following table compares both modes:

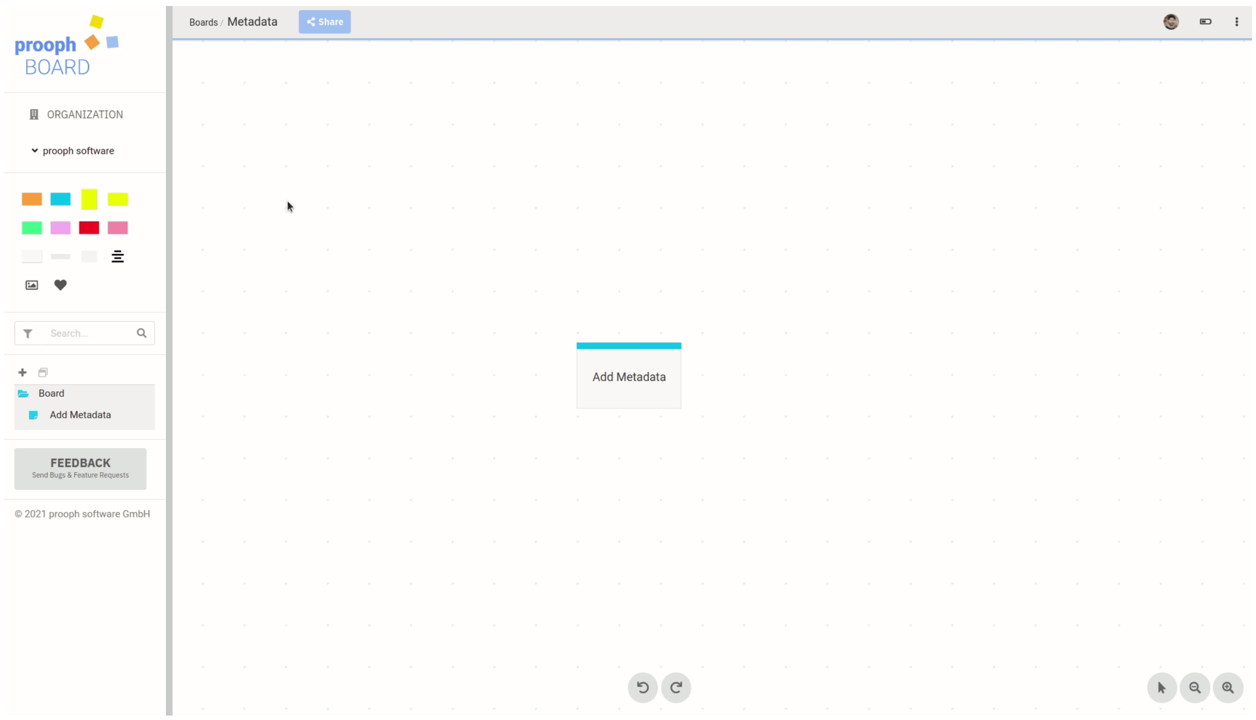
	Mode	High Level Overview	Font Scaling	Performance
	Normal	yes	yes	no
	Lite	no	no	yes

METADATA

prooph board bridges the gap between problem and solution space. Start from a High Level Event Storming and walk all the way down to a detailed solution. Therefore, prooph board offers advanced documentation features like **Metadata** support for most of the elements available on a board.

Metadata is useful to document technical details like message schema or information for **Cody** (the coding bot), without distracting non-technical team members, when working together on a board.

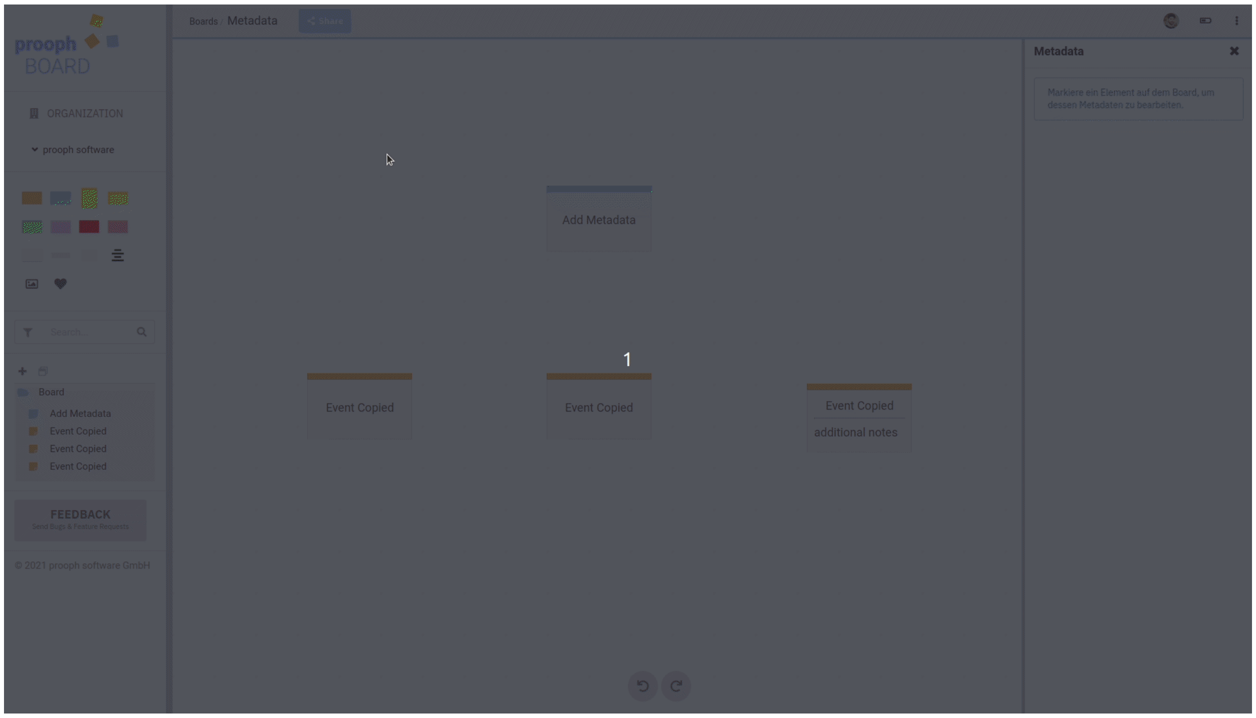
Add Metadata



You can open the metadata sidebar from the context menu of a card (or other elements), the top menu or by pressing `Ctrl+M` (Cmd+M on Mac). The sidebar shows some information and you can edit the card's metadata. If a template is set (see below) for the selected card's type and no metadata is set so far, the template is loaded into the editor.
Note: Changes are saved automatically and synchronized with your remote peers.

Write Once, Copy to All

It’s ok to duplicate cards on a board to improve readability. Too many arrows make it look chaotic. But when it comes to metadata, you want to edit it in one place. We got you covered! prooph board detects similar cards automatically and ask you to apply changes to all of them.

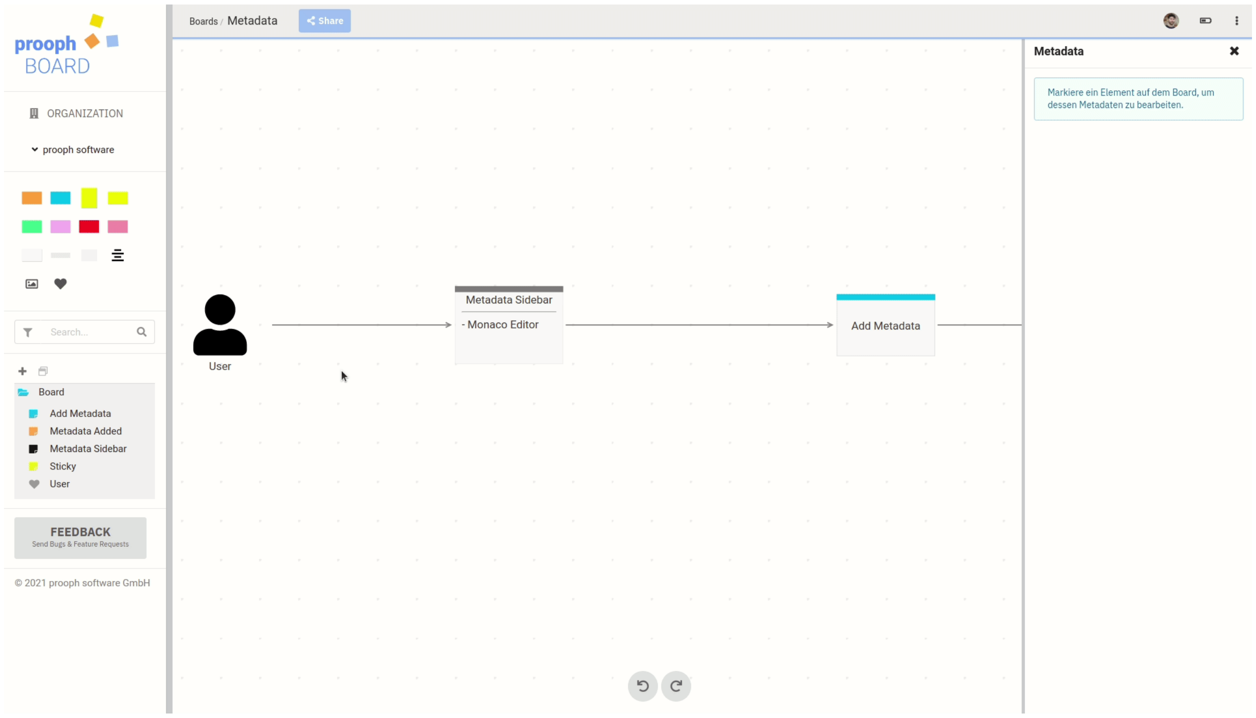


Markiere ein Element auf dem Board, um dessen Metadaten zu bearbeiten.

Lookup is based on card type and name. For the latter it is important to note that prooph board treats everything above the first horizontal ruler as [the name of a card](). This allows you to add notes, examples and other information that should be visible for all.

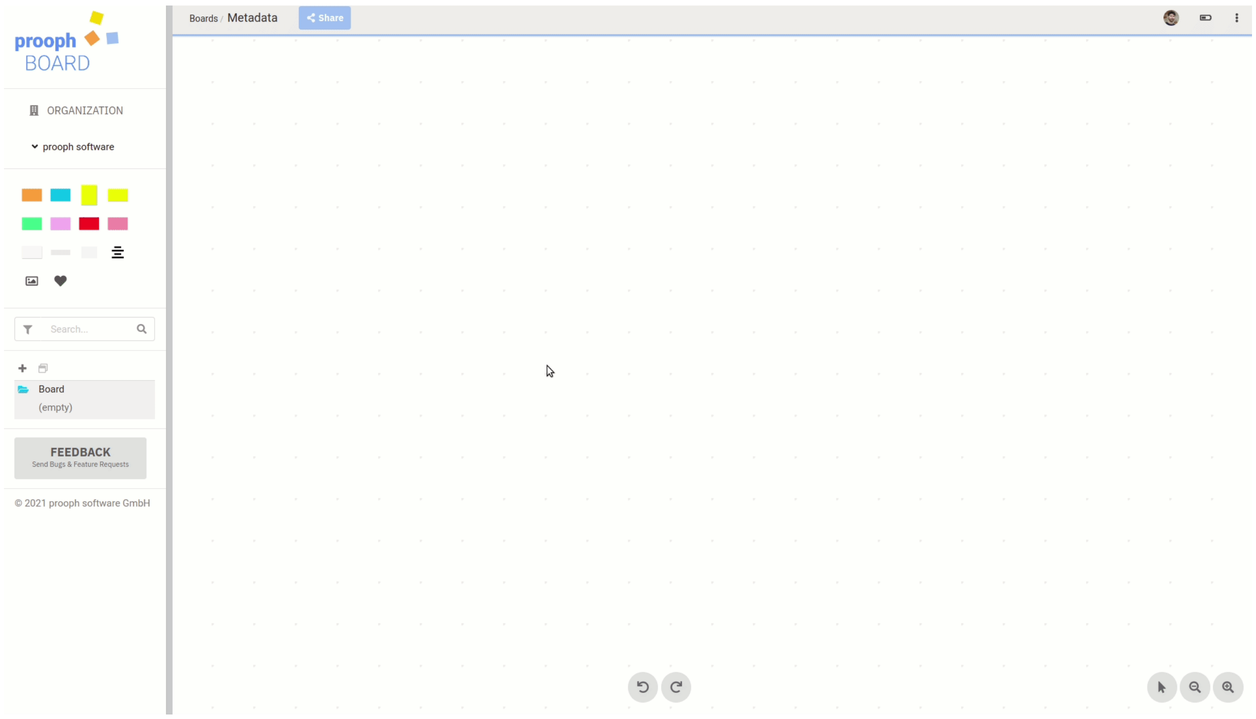
Follow Connections

Cards can be connected to document a message flow e.g. a command (blue card) produces an event (orange card). Connections are listed in the metadata sidebar and a click on one moves focus to it.



Card Metadata Templates

Metadata templates can be set per card type for each board. prooph board also ships with some default templates e.g. a JSON-Schema template for commands (blue), events (orange) and information (green). The default templates also contain information for the code generator like the `newAggregate: boolean` flag on commands which indicates that a new aggregate is created The code generator is fully customizable, so are the templates. It’s a powerful combination.



Upcoming Additions

Syntax Support

So far the template and metadata editor supports JSON syntax. We will add a possibility to switch syntax.

Global Templates

One of the next releases will include a way to quickly copy templates from one board to another.

TREE VIEW

The left sidebar contains a tree view. It’s purpose is similar to a table of contents. All elements on the board are also listed in the tree view.

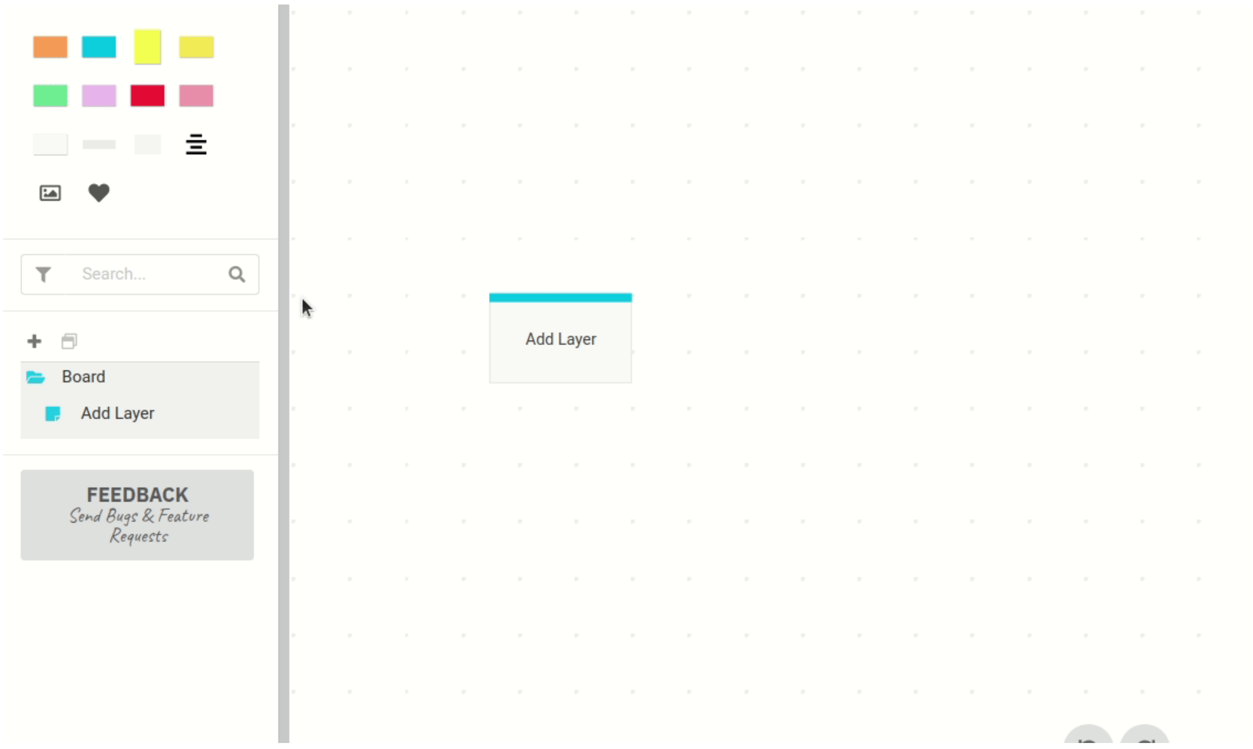
You can use the tree view to navigate to certain elements by clicking on them. If elements are nested within a parent (e.g. cards grouped by frame), a single click on the parent will show/hide children and a double click will navigate to the parent on the board.

Tree Structure

Layers define the top level of the workspace (sub layers are not supported yet). Beneath them the structure is automatically derived from your activities on the board. You have two frame types available that group elements in the tree.

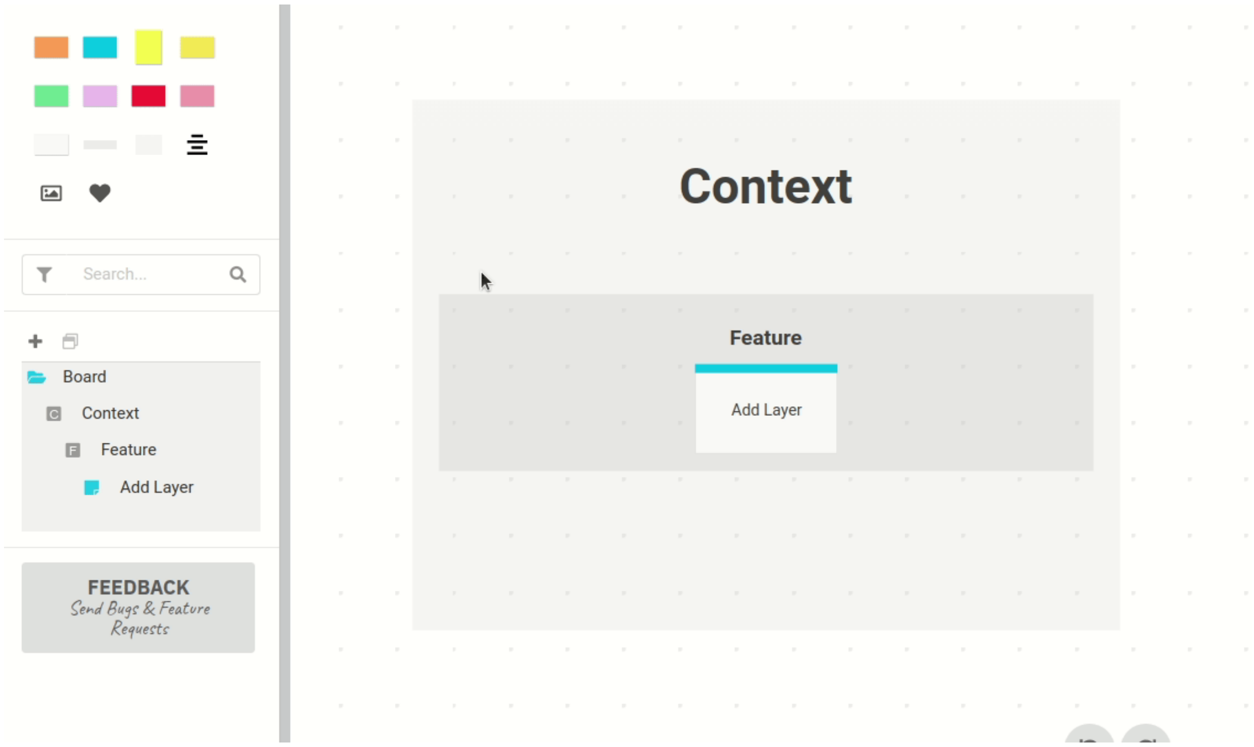
Nesting Frames

Nest Frames on the board and put Cards and other elements into them. The structure is then visible in the tree view and gives you an higher level overview when zooming out.



Show/Hide and (Un)lock Elements

A right click on an element in the tree opens a context menu with some options. You can lock or unlock elements, which means that no one can edit it or its nested elements.



Find an element on the board

Use the filter and search bar located above the tree view to find elements. The tree view is filtered immediately and shows all matching elements.

By clicking on the filter icon you can select the type of element you are looking for.

Advanced Search

Search Term	Description
some text	Matches text in element names
#tag	Matches all elements with “tag” assigned
!#tag	Matches all elements without “tag”
type:element	Matches all elements with given type. Same as using the type filter

term1; term2	Combine search terms, for example: #tag; !#other; type:event

If you have a card selected on the board and press *Ctrl+F (Cmd+F on Mac)* the type of the card will be set as filter and its name as search term. This shortcut is useful to quickly find similar cards on the board.

Drag Elements From The Tree

You can drag an element from the tree view and drop it on the board, which is a shortcut for copy & pasting the element.

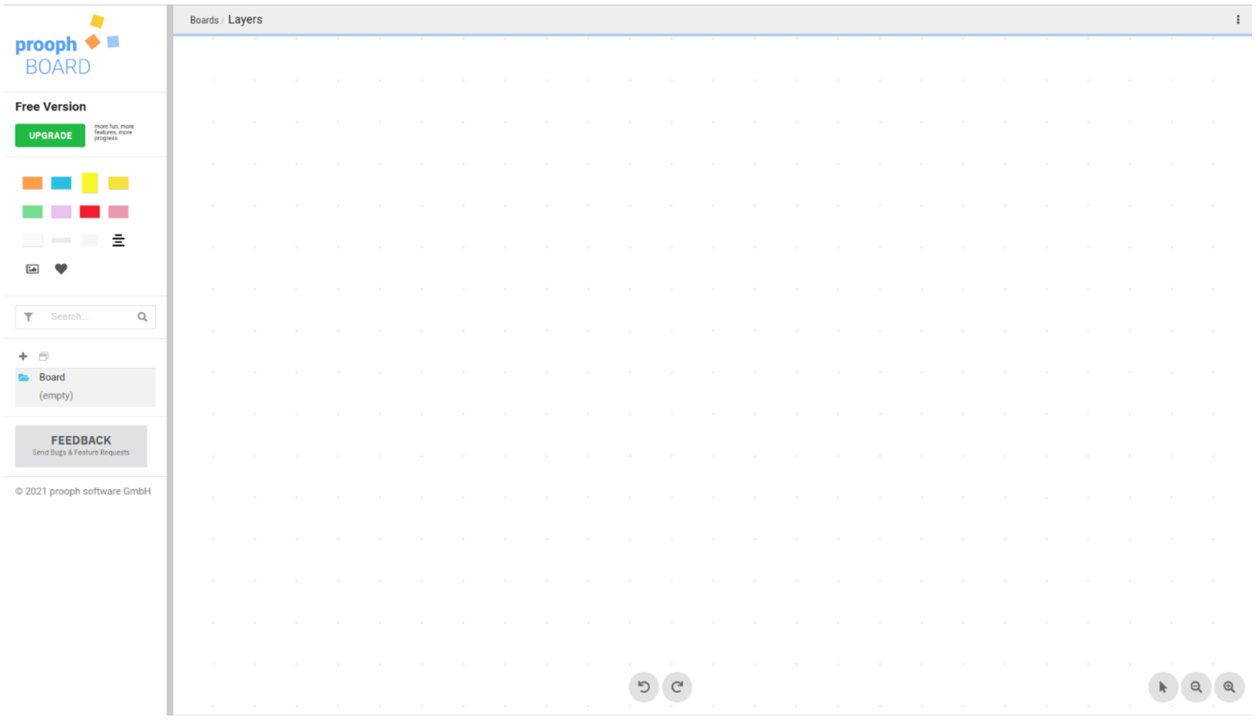
https://wiki.prooph-board.com/assets/video/board_workspace/draggable_tree_view.mp4

Working with Layers

You may know layers from drawing and image editing tools like Photoshop. In prooph board they work quite the same, but have a slightly different purpose. Let's imagine your team did a High Level Event Storming on prooph board and now you would like to sketch out some first model design ideas. You'd like to do that next to the High Level Storming to use it as a reference. Anyway, next time your team wants to continue with the High Level Storming and design-level ideas would distract other team members. With layers you can have both! Sketch ideas on dedicated layers, but hide them when the team works together on the event map.

That's just one example of using layers to better organize ongoing work on an event map.

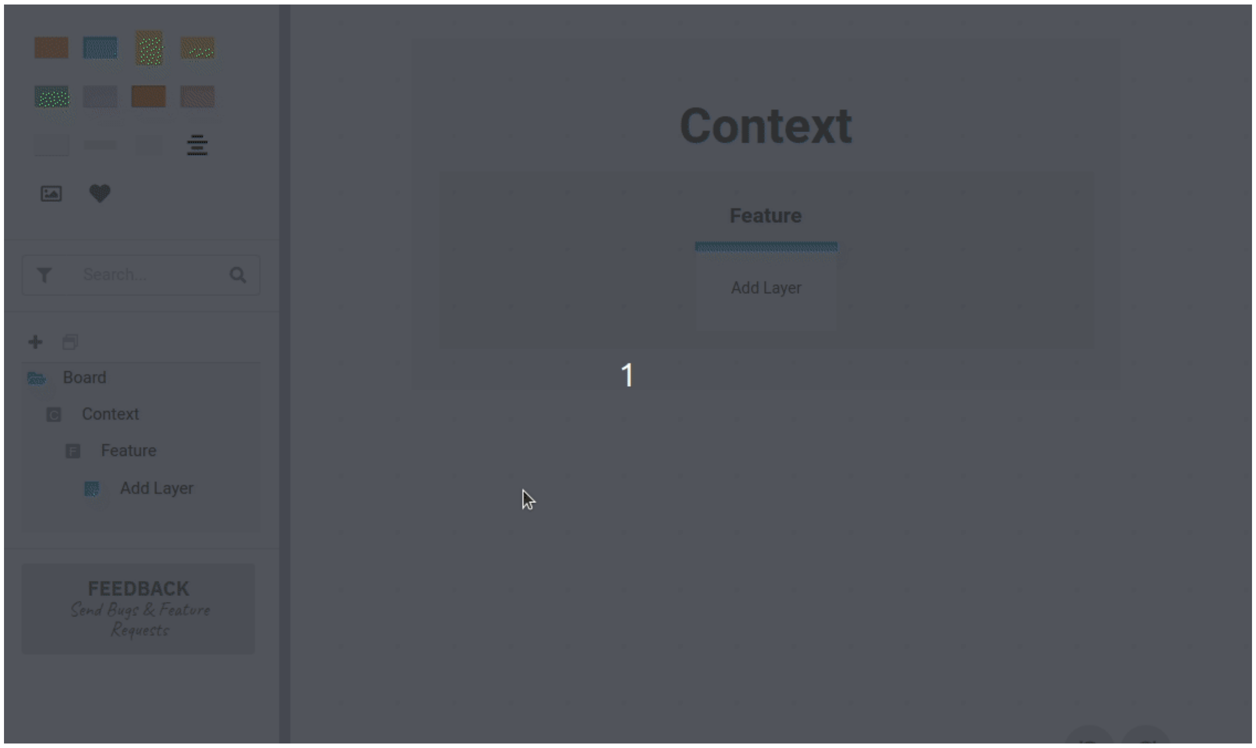
Default Layer


prooph BOARD
Free Version
UPGRADE
more fun, more features, more progress
Search...
Board
(empty)
FEEDBACK
Send Bugs & Feature Requests
© 2021 prooph software GmbH
Boards / Layers

Each board has a default layer which is called `Board` by default. You'll find it in the left sidebar on the board workspace. You can rename the layer, but it is not possible to delete it (other layers are deletable, though).

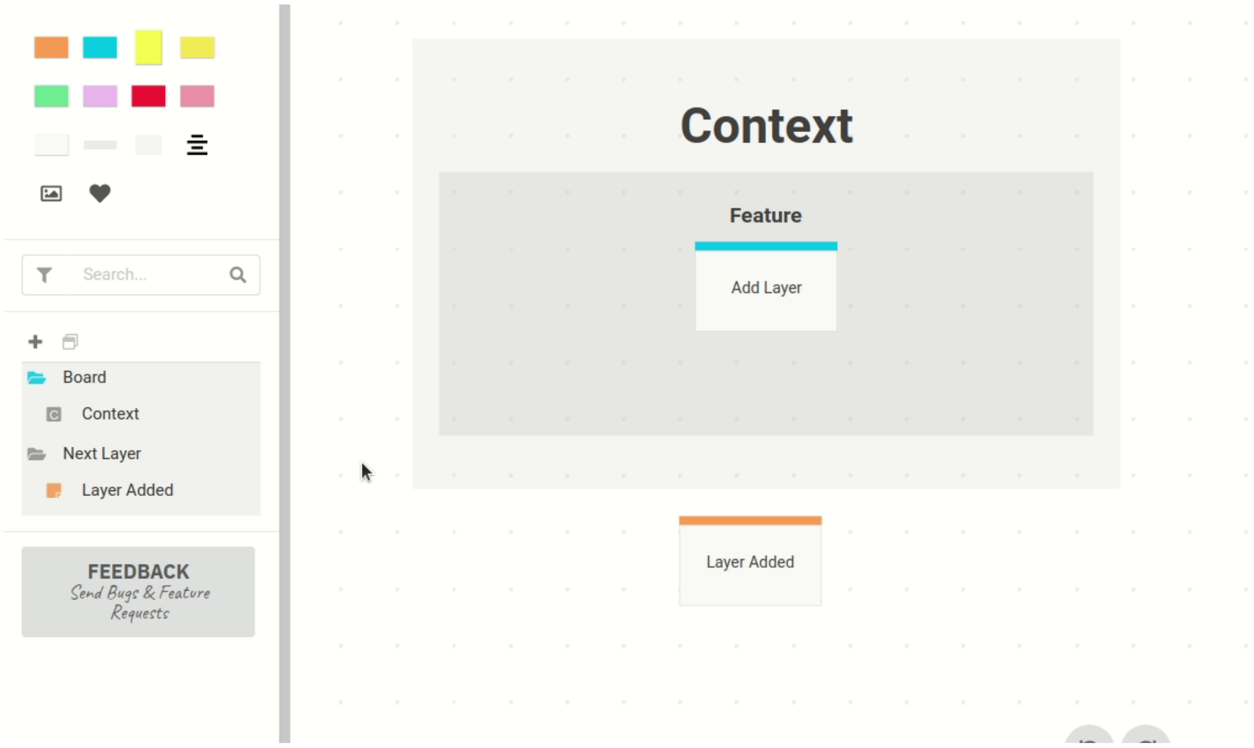
Adding a Layer

Adding a layer is quickly done. Click on the `plus icon` above the layer tree. A new layer gets added to the tree and you can set a name for it. Pressing `ESC` will abort the operation and pressing `Enter` completes it.
The layer tree is sorted alphabetically, so your new layer is moved to it's final position in the tree after pressing enter. It becomes the active layer (see below, for more information) and is therefor marked with a blue folder icon (instead of black).



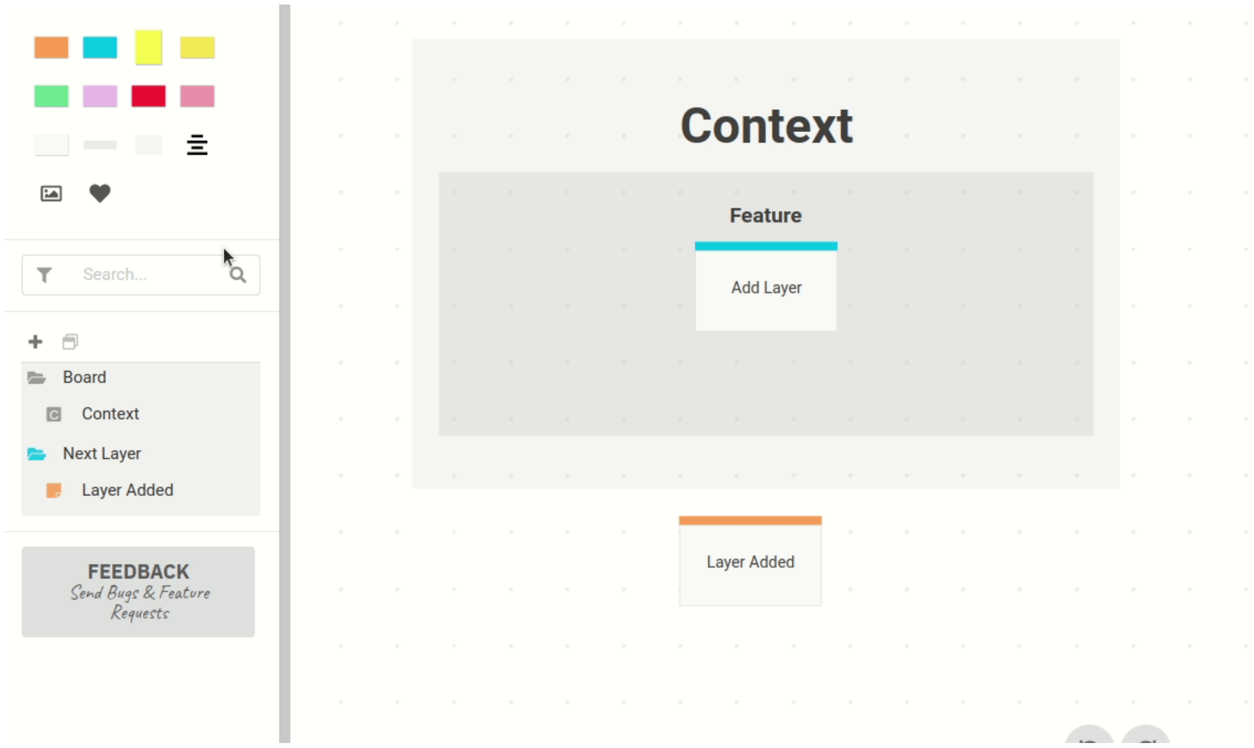
Switch active Layer and move Elements to it

A single click on a `folder` icon opens (or closes) it and marks the layer as active (folder icon becomes blue). New elements added to the board are basically added to the currently active layer.
Existing elements can be moved from another layer to the active one by selecting the elements on the board and then clicking on the copy icon next to the plus above the layer tree. The folder icon becomes orange for a second to signal success of the operation. You’ll now find the moved elements beneath the active layer.



Rename or delete a Layer

The `edit` button in the layer context menu actives an inline edit for the layer name. Rename it and press `Enter` or abort the operation with `ESC`.
Layers (and other elements) can also be deleted if you no longer need them, except the default one (see above). A quick confirmation saves you from fast unwanted clicks without getting to much in your way.

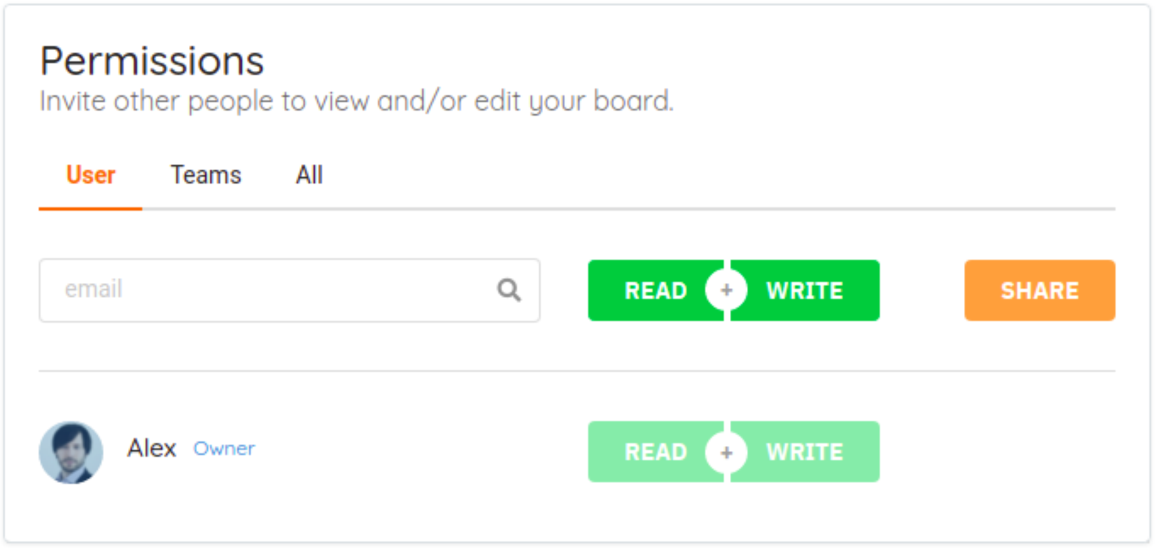


ACCESS RIGHTS

prooph board offers 3 different levels of setting access rights for boards. This can be done on the board settings page that you can either reach from the board overview (by clicking on the edit button of a board) or from the top menu on a board workspace.

Please note: Only board owners and organization admins can access board settings. If you don’t have this role, you won’t see the edit button or settings entry in the top menu.

The 3 Levels can be found in the Permissions section:



User Level

You can set **read** and **write** access for a specific user on this level. An input field allows you to look up members of the organization by username or email. If a member is found your settings become active right away.

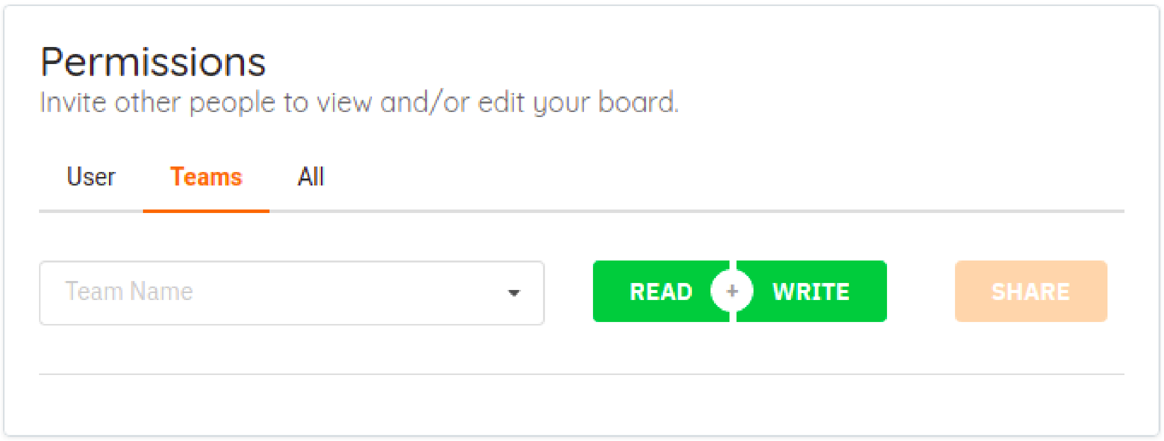
If the person is not part of your organization, you can invite them by providing their email address. In this case the person will receive an invitation email with an activation link.

Access will be granted only if the invited person clicks on the activation link in the email and if you have enough user quota available in the organization. The invited person will be registered as a guest in the organization. For more information see **Guests**

Open invitations are also listed in the Permissions section. You can resend an invitation email or revoke an open invitation.

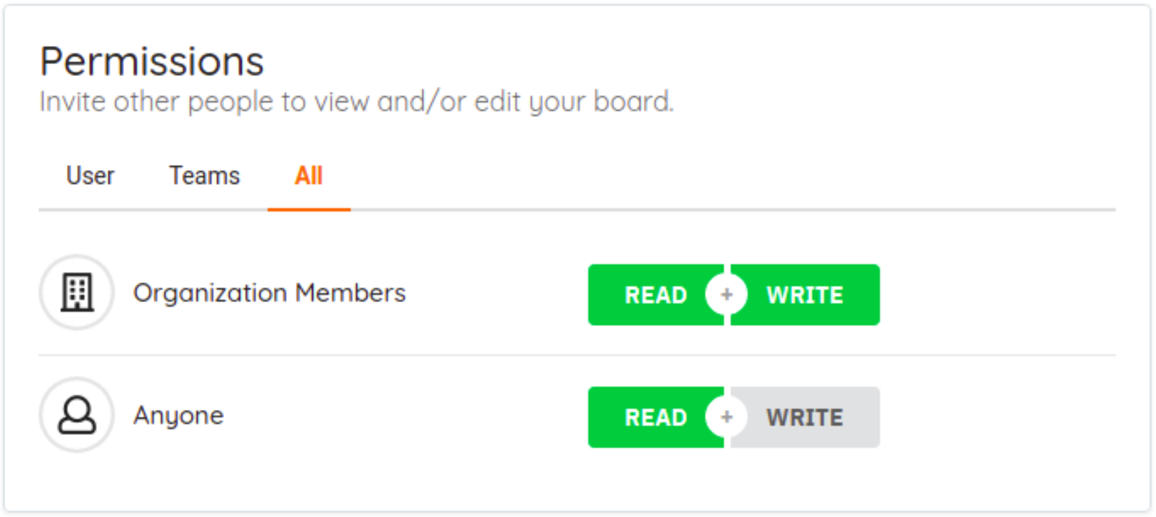
In the top bar of a board workspace you'll find a **share** button (again, only if you're owner or admin). This button gives you quick access to user level permissions. If you forgot to invite someone, you can quickly do it right there.

Team Level



read and **write** access to a board can be granted to an entire team. Select one from the dropdown and click on the **share** button. Everyone in the team will now have access.

All (Others) Level



On the **All** tab you can set organization-wide permissions or even allow anyone to access your board.

While user and team permissions will also make the board visible on the board overview for every person, this is not the case for **All** permissions. So people need to know the direct link to the board.

Access for anyone works as long as enough user quota is available in the organization. Each unknown person who opens a board is added as **guest**.

MANAGING AN ORGANIZATION

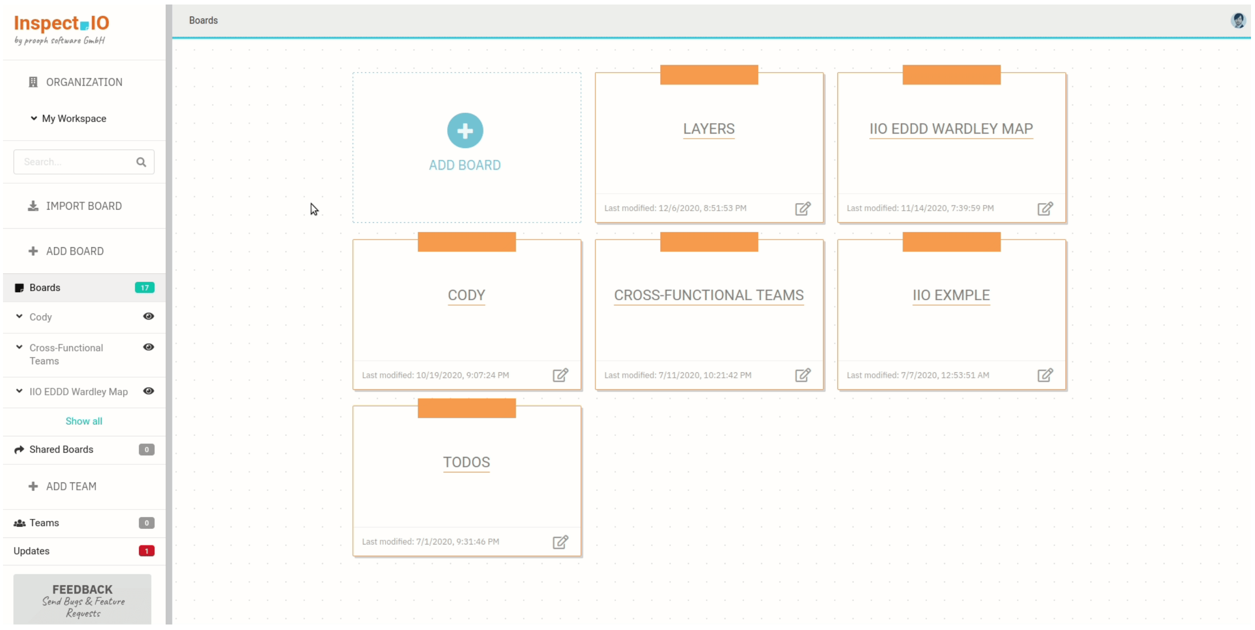
With an organization you can share your board and user quota with colleagues. They don’t need to worry about limitations, can easily create boards and share them with members.

Create an Organization

Each account can have exactly one organization. It’s connected with the selected payment plan and therefore with available board and user quota.

To create an organization navigate to “Organization” management via sidebar, click “Create Organization” and give it a name. That’s it!

Please note: GIFs on this site are recorded with the beta version of prooph board, which was called Inspect.IO. So don’t be confused by the different logo and name. It has changed, but functionality for organizations is still the same.

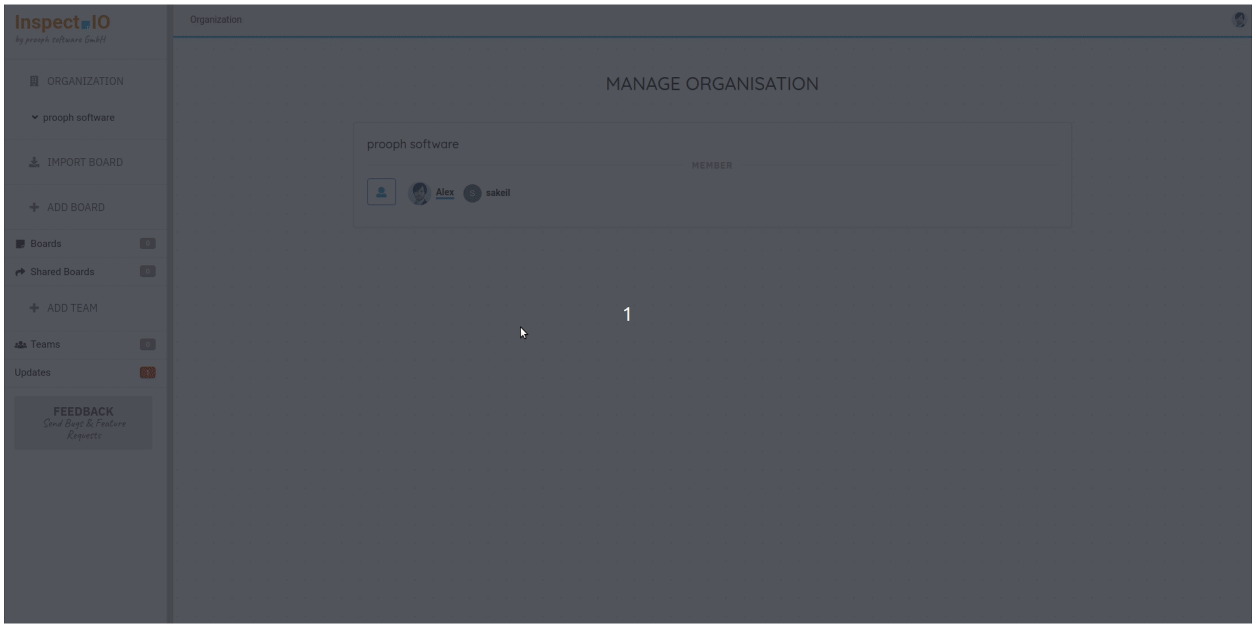


Invite Members

Click on the blue user button next to your avatar. A dialog will open asking you to enter the email address of the member you want to invite. They’ll receive an email with an invitation link. If they’re not registered yet, they can do it for free. Once a member has joined an organization, they can create boards as long as the organization has enough quota.

Promote Member to Admin

Promoting members to organization administrators helps you to distribute administrative tasks like cleaning up no longer used boards, invite or remove members and more.



An administrator can:

- Invite or remove members
- Has access to all organization boards
- Has access to all organization teans
- Can manage access rights
- Promote a guest to member

Only the organization owner can:

- Promote/demote administrators
- Change payment plan

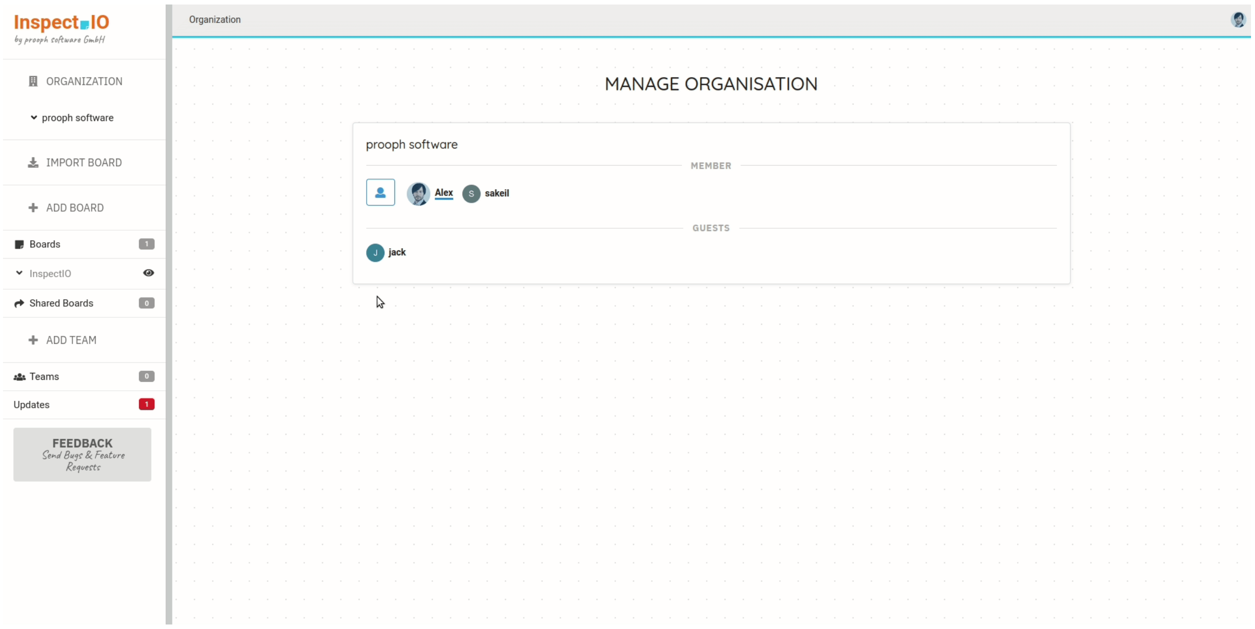
Member Lookup

Boards can be shared and people can be invited to join teams. Within an organization this is very easy. When asked for an email address just start typing to get suggestions for members. You can lookup email addresses and names.

Guests

It’s also possible to invite external people to organization boards and teams. They will only get access to the shared board or in case of a team to all team boards. A guest cannot create additional boards within the organization context. And a guest has no access to the member list.

You can promote guests to organization members.



Migration

You’ve created an organization and now want to move boards and teams to it? No problem, switch to your private workspace using the workspace select in the sidebar below the organization item. You’ll find an option to move your boards and teams to an organization on the “Board Edit” and “Team Management” pages.