Extracted from:

Creating Software with Modern Diagramming Techniques

Build Better Software with Mermaid

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic exPress

Creating Software with Modern Diagramming Techniques

Build Better Software with Mermaid

Ashley Peacock

Edited by Michael Swaine

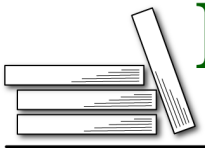
Creating Software with Modern Diagramming Techniques

Build Better Software with Mermaid

Ashley Peacock

The Pragmatic Bookshelf

Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Michael Swaine
Copy Editor: L. Sakhi MacMillan
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-983-0
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—February 2023

Creating a System Context Diagram

Let's start with the highest-level view, the system context diagram. It is the 50,000-foot view, so contains the minimal level of detail of our architecture. It should be nontechnical and simply acts to model the interactions between users of the system you're designing and any other systems in play.

The litmus test you can use to determine if the level of detail is correct is whether I could show it to my product manager, or anyone nontechnical, and they would be able to understand what the diagram is trying to convey. If they can, you know it's the right level. It's not useful just for product managers though; it's key in understanding at a glance what your system's key dependencies are and how it undertakes its responsibilities.

Continuing on the journey of Streamy, we're going to design an architecture for the service that is responsible for displaying the lists of titles available to view on the platform. In essence, when a user goes to the platform, what does the listings service need to do to show the titles, and how does it do it?

A second requirement is that when a title is displayed, the reviews for that title are also displayed, and the user has the option to submit a review for that title if they wish.

Thirdly, listings are great, but the user should also be able to search for specific titles if they want to watch a particular one.

Finally, other engineering teams at Streamy have been busy creating services to support the upcoming launch. Three services are already available for us to use: a title service, a review service, and a search service.

In a real-world scenario, this kind of information would be either readily known by you or your team or it would be discovered as part of this exercise.

As mentioned in Using the C4 Model, on page ?, there are three main elements in a system context diagram:

- People
- Your software system (that you are designing)
- Supporting software systems

Let's work from top to bottom and start forming our system context diagram. We'll be using a flowchart, supported by Mermaid. Flowcharts are extremely versatile and, unlike the prior diagrams we covered, don't have a single use case. We're going to use them to create C4 diagrams, but you could also use them to diagram a business use case, as they support branched logic

quite well. If you need to create something that doesn't fit in any of the use cases covered in this book, you can likely create it using a flowchart.

We now understand what we're trying to accomplish and how we plan to do it, so let's get started!

Add Nodes

Let's get started by defining a flowchart and our first element, the user:

```
flowchart TD
   User["Premium Member
   [Person]

   A user of the website who has
   purchased a subscription"]
```

Nodes also support adding in new lines (\n) in text, which I prefer to use rather than actual new lines in the diagram's code, as I find it harder to read. The lines aren't actually that long, but for a book, they need to be smaller, so I've used this format.

Once generated, it looks like the image shown on page 7.

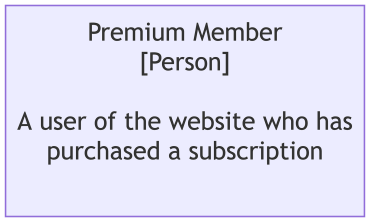
In this example only one person is being shown, but it's not uncommon to have multiple users interacting with your system in different ways, so make sure to include everyone.

As with all Mermaid diagrams, the first line defines the type of diagram we're creating—in our case, flowchart. There's an optional parameter you can define after flowchart that defines the direction, which accepts the following options:

- *TB*: top-to-bottom
- *TD*: top-down (same as top-to-bottom)
- *BT*: bottom-to-top
- *RL*: right-to-left
- *LR*: left-to-right

We'll be using top-down as that suits our use case.

The second line defines a *node* in our flowchart. You can think of this like defining a class before you use it in your codebase: the node is only defined once and then used where it's needed later on. This allows you to define certain characteristics for a node once, and it will be used automatically throughout the flow.



All node definitions follow the format `id["Label / Description"]`. The ID is the reference you'll use when defining the interactions later on, so I tend to keep them short, and the label/description is the text that goes in the box that's rendered.

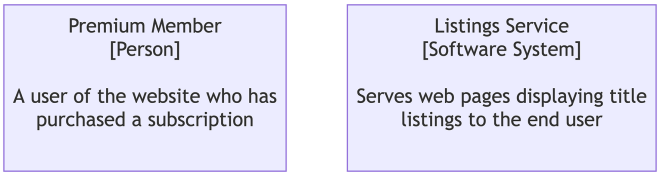
Following Simon Brown's notation, each node should contain a title, label, and description. The title should clearly outline the node, the label is what type the node is, and the description briefly describes what that node represents. At this level, the label is perhaps superfluous, but in more detailed views that follow, it's essential, and I like to keep the layout consistent between the different levels of the C4 model.

We can't make much of a diagram with just a single node, so next we add our system—the system we're designing and documenting with the C4 model. The Mermaid definition now looks like this:

```
flowchart TD
   User["Premium Member
   [Person]
```

```
A user of the website who has
purchased a subscription"]

LS["Listings Service
[Software System]

Serves web pages displaying title
listings to the end user"]
```

As nothing is connected yet, we now have two isolated nodes displayed:



Nodes on their own don't provide much value though, so let's try connecting these nodes.

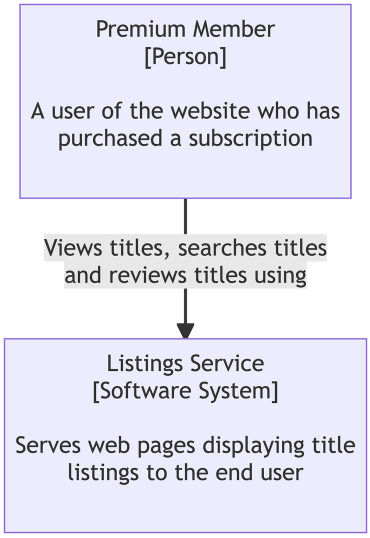
Connect Nodes

Now the fun begins! We can connect our two nodes in a flowchart using a variety of arrows, but for a system context diagram we just need simple solid arrowheads that show dependencies. We can add an interaction between nodes like so:

```
flowchart TD
   User["Premium Member
   [Person]

   A user of the website who has\npurchased a subscription"]

   LS["Listings Service
   [Software System]

   Serves web pages displaying title
   listings to the end user"]

   User-- "Views titles, searches titles\nand reviews titles using" -->LS
```

You can define arrows in two different styles, so pick your preference. We can use the one shown previously that follows the format `ParentNode-- "arrow label" -->ChildNode`, or we can use the format `ParentNode-->|"arrow label"|ChildNode`. I personally find the former easier to read, but both work in exactly the same way. The double quotes are optional but are required later on for more detailed labels, so you'll want to get into the habit of using them now.

If we generate this diagram, we can now see the two nodes linked, as shown on page 9.



I've modeled the arrows as dependencies, so for the arrow label I simply describe what the parent node relies on from the child node. You don't need to go into large amounts of detail, especially at this level, so try to keep the descriptions brief.

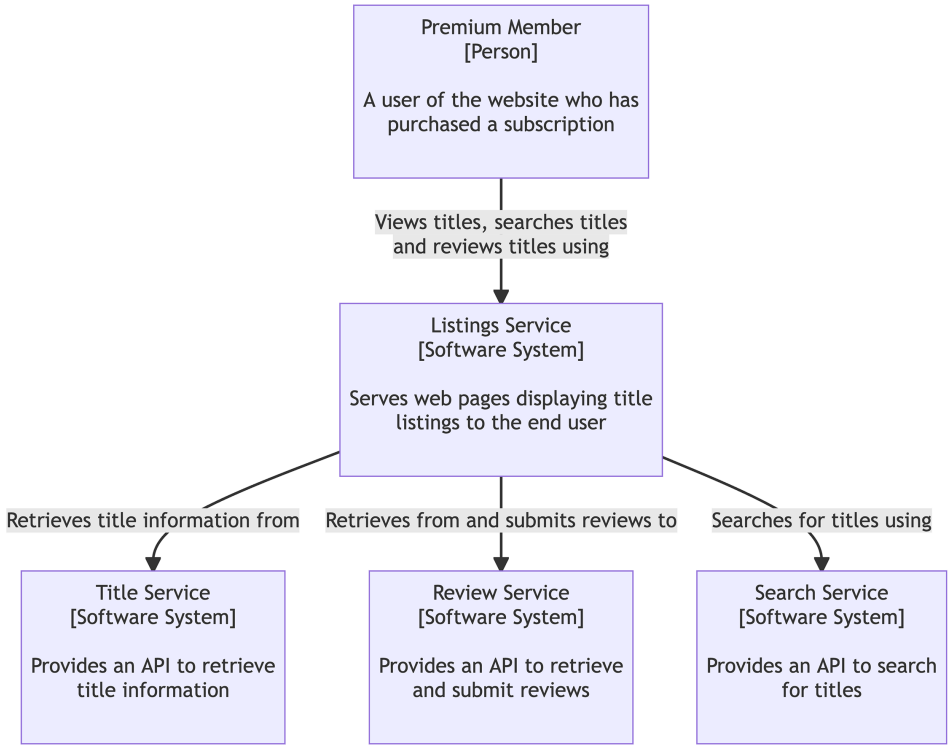
We now have the people interacting with our system. And our new system, on the system context diagram, just one more thing is left to add: supporting systems. These are any systems that your system interacts with and that are required for it to do its job. They can be other internal systems or external systems provided by another company, such as Salesforce if you use that for your customer relationship management (CRM).

In our case, we determined earlier (Creating a System Context Diagram, on page 5) that there were three supporting systems already available to use at Streamy: the title service, the review service, and the search service, so let's add them to our system context diagram.

```
flowchart TD
   User["Premium Member
   [Person]

   A user of the website who has
   purchased a subscription"]

   LS["Listings Service
   [Software System]

   Serves web pages displaying title
   listings to the end user"]
```

```
TS["Title Service
[Software System]

Provides an API to retrieve
title information"]

RS["Review Service
[Software System]

Provides an API to retrieve
and submit reviews"]

SS["Search Service
[Software System]

Provides an API to search
for titles"]

User-- "Views titles, searches titles\nand reviews titles using" -->LS

LS-- "Retrieves title information from" -->TS
LS-- "Retrieves from and submits reviews to" -->RS
LS-- "Searches for titles using" -->SS
```

Once generated, we have a completed system context diagram!



Using this diagram, both technical and nontechnical colleagues can understand at a high level who uses your system, what your system does, and how it does it in combination with other systems. That documentation alone is *probably* more detailed than most repository READMEs and will answer several initial questions anyone has when they want to know about your new system.

Adding HTTP Links to Nodes

Do you remember that for class diagrams you can add links to nodes? You can do exactly the same for flowcharts, using exactly the same syntax!