Devdoc-Swissknife EN

Nikolay Gniteev

2021-09-30

Contents

Preface

This is a demo project for development documentation generation with R Markdown and Kroki

This approach allows you to create reproducible documentation in form of PDF, presentations, gitbook and some other HTML formats from simple markdown-like files (Pandoc flavor) with extra R chunks and textual diagrams description.

The goal of this project is to describe documentation via text files as much as possible, even graphic diagrams, and make it easy to use without overcomplication.

A textual description provides an easy way to version and merge diagrams, get differences between versions, leave comments that stays out from generated documentation, review changes, etc.

Also most often it's faster to create and **edit** diagrams as text, do things like theming and refactoring themes. Since the diagram's textual description often is more a model than just graphics, you don't have to track existing objects relations when just placement is changed or objects added (until you rename objects) which makes your workflow less error-prone.

R Markdown itself is a powerful and customizable tool, so you can fit docs to your needs. It would take efforts on start, to make things neat but it would repay you in the end.

> To produce awesome PDFs you'll have to provide awesome Latex book class. Since I don't have one I'm using Latex default so current PDF version looks a bit ugly to me. I hope I can fix this soon or later. Help is appreciated!

This project is also can be used as a template for your own documentation,

see this section

The project uses docker and docker-compose to build docs so it's ready to be integrated into your CI/CD flow.

The project's structure, docs generation and some files are based on R Markdown book

---

There are few demonstrations in the project for general development use-cases (which probably cover 95% of your daily needs). For more complicated use-cases - checkout docs on links below.

Diagrams code for most examples is borrowed straight from Kroki

Following links would be helpful to you:

List of supported diagrams renderers - https://kroki.io/#support

A few more diagrams examples - https://kroki.io/examples.html

R Markdown: The Definitive Guide - https://bookdown.org/yihui/rmarkdown/
created by one of R Markdown authors and generated with R Markdown

R Markdown: Cookbook - https://bookdown.org/yihui/rmarkdown-cookbook/

A nice R Markdown tutorial - https://rmd4sci.njtierney.com/

R Markdown cheatsheet - https://raw.githubusercontent.com/rstudio/cheatsheets/master/rmarkdow

R Markdown reference - https://rmarkdown.rstudio.com/docs/reference/index.html

Keenwrite - edit and preview R Markdown Live!

Chapter 1

Simple diagrams examples

Since this project is focused on embedding diagrams text descriptions into R Markdown let's start with few examples just to illustrate a principle.

1.1 Example with inline diagram description

This example shows how to embed diagrams, describing them right in Rmd file

This code:

```
```{r echo=FALSE, results='asis'}
  to_diagram("graphviz", "Hello World",
  "digraph G {Hello->World}"
  )
```
```

would produce following diagram:

> Take a NOTE: When diagrams are inserted like this it's **IMPORTANT** that ("Hello World" in this example) is **UNIQUE** for **WHOLE** doc.
> More details on usage is in ...

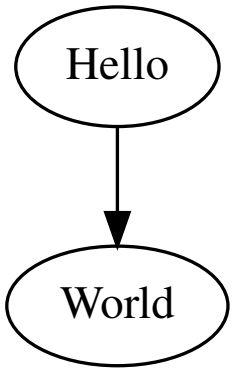

Figure 1.1: Hello World

1.2 Example with diagram data from file

This example shows how to embed diagrams with data from outer files.

This code:

```
```{r echo=FALSE, results='asis'}
  to_diagram("erd", "Entity Relation", src="../diagrams/examples/project.erd")
```
```

would produce following diagram:

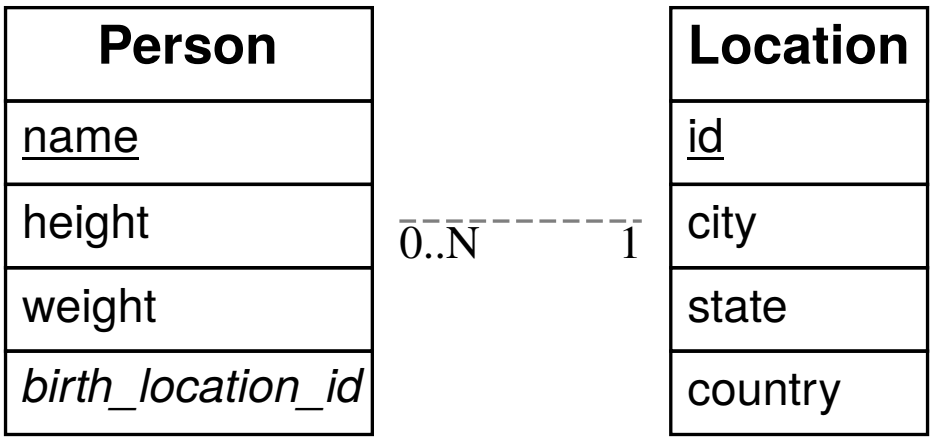

Figure 1.2: Entity Relation

File **diagrams/examples/project.erd** content:

```
[Person]
*name
height
weight
+birth_location_id

[Location]
*id
city
state
country

Person *--1 Location
```

1.3 More examples

More examples can be found in this section.

Chapter 2

Using as a template and making own docs

To use this project as a start point for generating your own documentation do the following:

1. Import project and prepare for making docs

   You'll need to have already `docker` and `docker-compose` installed

- Import or fork main branch of this repo: https://github.com/Godhart/devdoc-swissknife
  All the sources for docs are contained in `docs_src` folder.
- Make a new folder in `docs_src`. I suggest you name it using the following pattern: `doc-<subject>`
- Copy all the files from `docs_src/docs-template` into your brand new folder
- Replace following keywords `<Author Name>`, `<author>`, `<repo>`, `<Document Title>`, `Document_Title`, `<Document Description>`, doc- within files in your new folder with actual values. Don't forget also to adjust `Document_Title` in the `.gitignore` of new folder, as it protects from build garbage.
- If you don't want to see devdoc-swissknife documentation in your repo:
  - Remove `docs_src/devdoc-swissknife-*` folders
  - Empty `docs_src/diagrams` folder.

- Adjust `docs_src/Makefile` to fit your needs (for the first time - use an existing pattern to add your folder).
- Make docker image `devdoc-swissknife` if you don't have one already. Simply run `make_docker.sh` from `docker` dir.
- Try to make docs with `make_docs.sh`. Output docs should appear in `docs_out/doc-<subject>` folder if you followed a pattern in the Makefile.
  Take a NOTE: `docs_out` folder and all it's content is ignored by git.

  NOTE: in many cases of errors doc's sources folder is polluted with temporary files, named as specified in `_bookdown.yml` (field `book_filename`) and may break following docs generation runs.
  These files are removed with make routine, but in some cases, you'll have to remove them by yourself.
  Changing `book_filename` field in `_bookdown.yml` after an error has happened may be the case.

2. Create content

- Update `index.Rmd` in your folder to your needs (contains Preface section).
- Add your own docs into your folder, naming files like `<number>-<chapter-name>.Rmd`. Check the R Markdown and Kroki docs for understanding things. Also, you may rely on shown examples.
- If you already have docs in markdown format you may already use them like this:
  - copy markdown files into your doc-folder
  - copy necessary local images to location in `docs_src/diagrams` or wherever you like most
  - change name extensions of markdown files to `.Rmd`
  - change names of markdown files so they would correspond to pattern `<number>-<chapter-name>.Rmd`
  - change references to local images in markdown files
- If you already have text description of diagrams for supported rendering engines you may already use them like this:
  - copy necessary files to location in `docs_src/diagrams` or wherever you like most
  - in Rmd files replace image embedding with diagram embedding as described TODO

- Most probably you would like to use your own Latex class, so add `<your_latex_class>.cls` file into your folder and specify it in `index.Rmd` file (replace `documentclass: book` field with `<your_latex_class>` name).

  Adding custom latex class may require you to add some latex packages to Docker image. Same is true if you do use some special R packages in your docs etc.
  If this is the case you'll have to modify `docker/Dockerfile` and build again docker image with `make_docker.sh`.

Chapter 3

Kroki usage examples

I've omitted some examples from Kroki and left only those that most useful in a daily life of most developers (IMHO).

If you check docs for supported diagrams renderers then you'll find for sure a few more interesting usecases.

All diagrams data for examples of this section resides in docs_src/diagrams/examples dir of this repo. Each diagram is included into doc by adding following code section into document file:

```
```{r echo=FALSE, results='asis'}
  to_diagram("from_src", "<Drawing name>", src="../diagrams/<src_file_path_within_diag
```
```

Full code for this whole section is here

This usage pattern is described in the following section TODO

3.1 C4 Context Diagram (PlantUML+C4)

Engine: `c4plantuml`

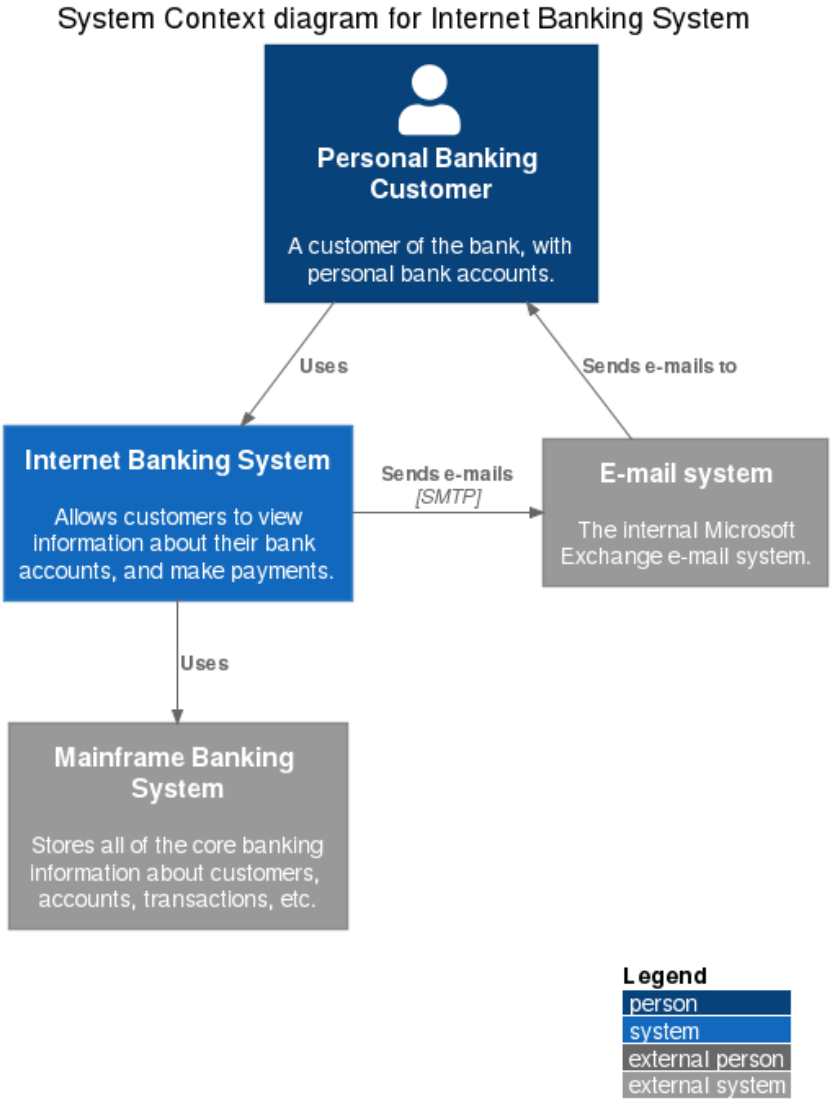

Figure 3.1: Example - C4 Context Diagram

*NOTE: inserted as PNG image due to errors in SVG to PDF conversion

3.2 C4 Container Diagram (PlantUML+C4)

Engine: `c4plantuml`

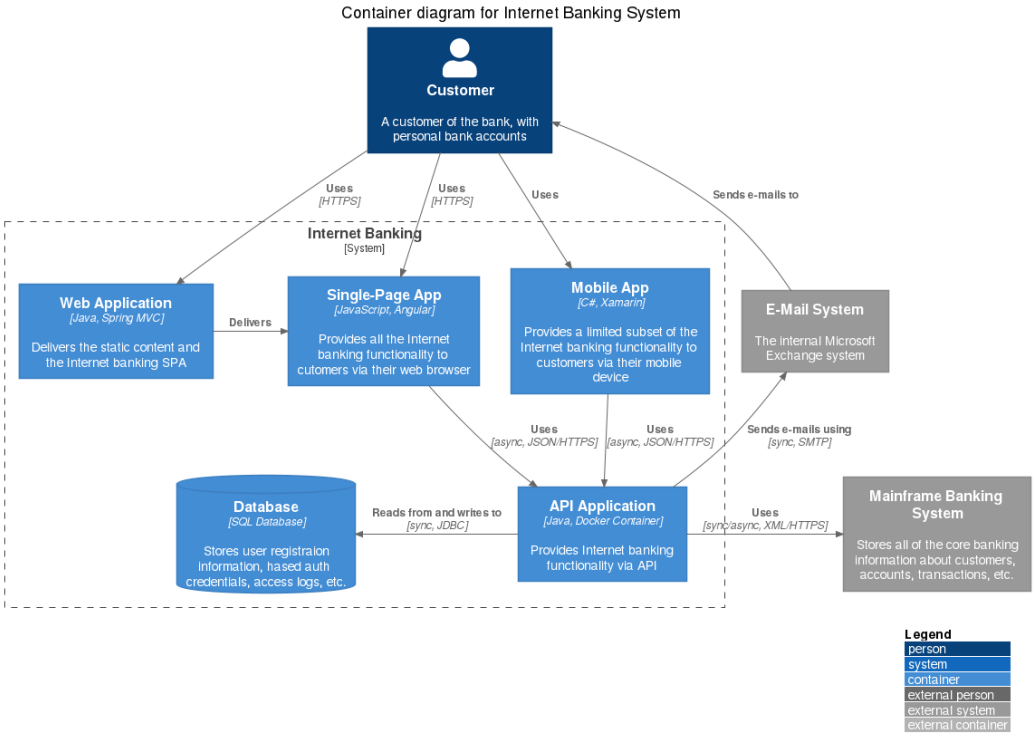

Figure 3.2: Example - C4 Container Diagram

*NOTE: inserted as PNG image due to errors in SVG to PDF conversion

3.3 C4 Component Diagram (PlantUML+C4)

Engine: `c4plantuml`

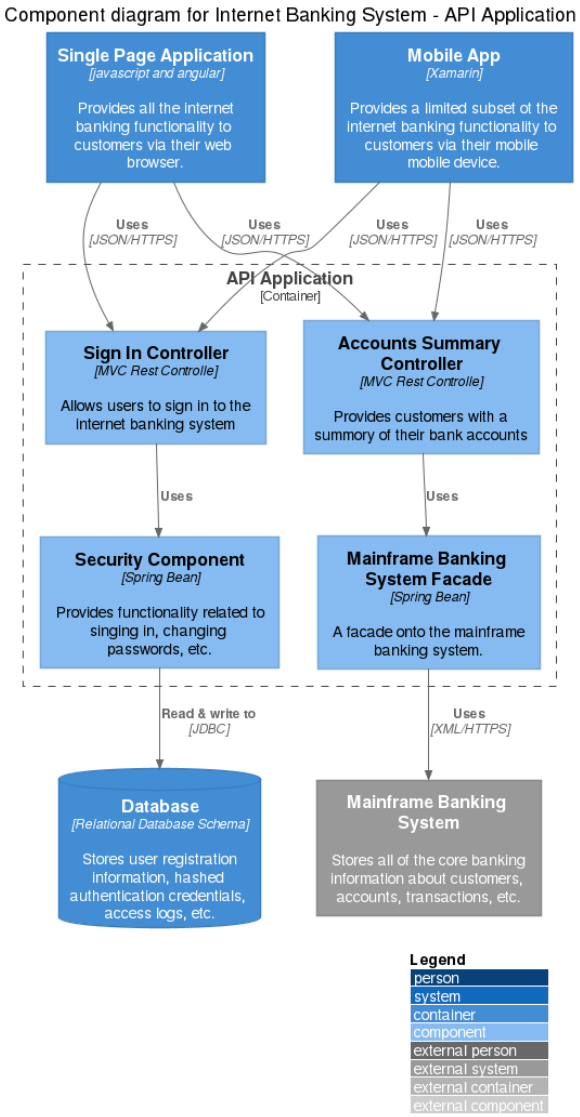

Figure 3.3: Example - C4 Component Diagram

*NOTE: inserted as PNG image due to errors in SVG to PDF conversion

3.4 Block Diagram

Engine: `blockdiag`

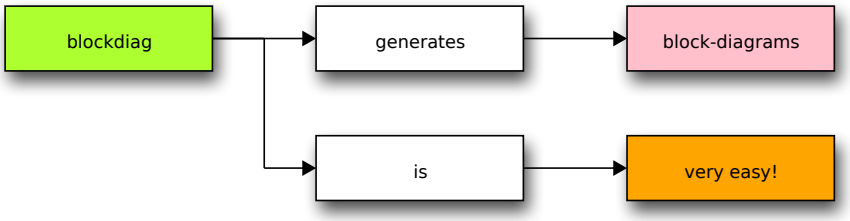

Figure 3.4: Example - Block Diagram

3.5 Digital Timing Diagram

Engine: `wavedrom`

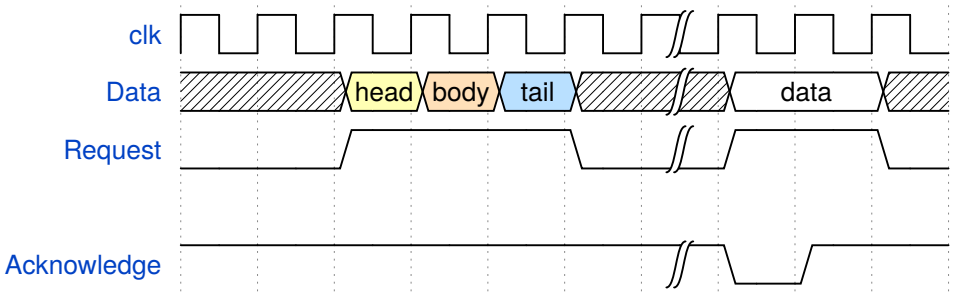

Figure 3.5: Example - Digital Timing Diagram

3.6 Bytefield

Engine: `bytefield`

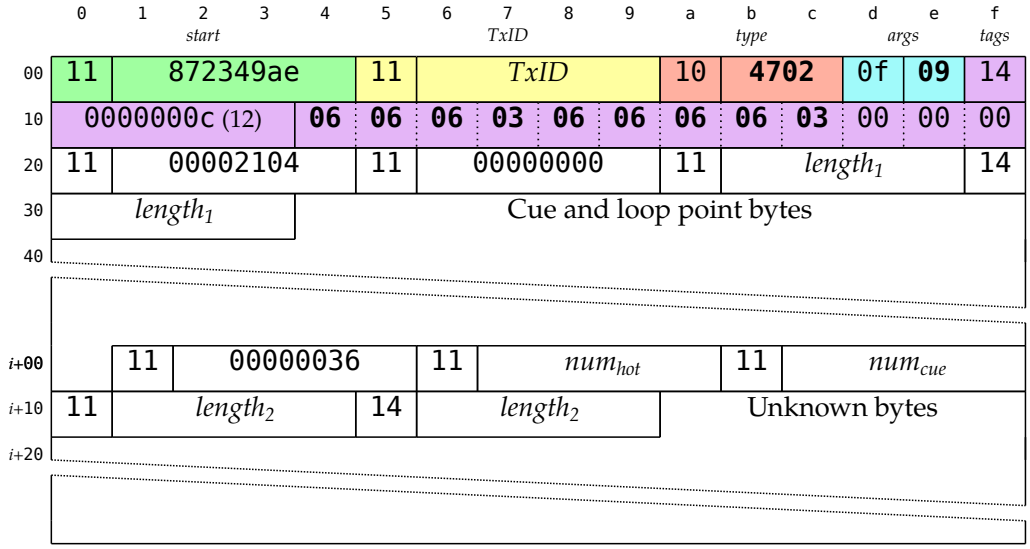

Figure 3.6: Example - Bytefield

3.7 Packet Diagram

Engine: `packetdiag`

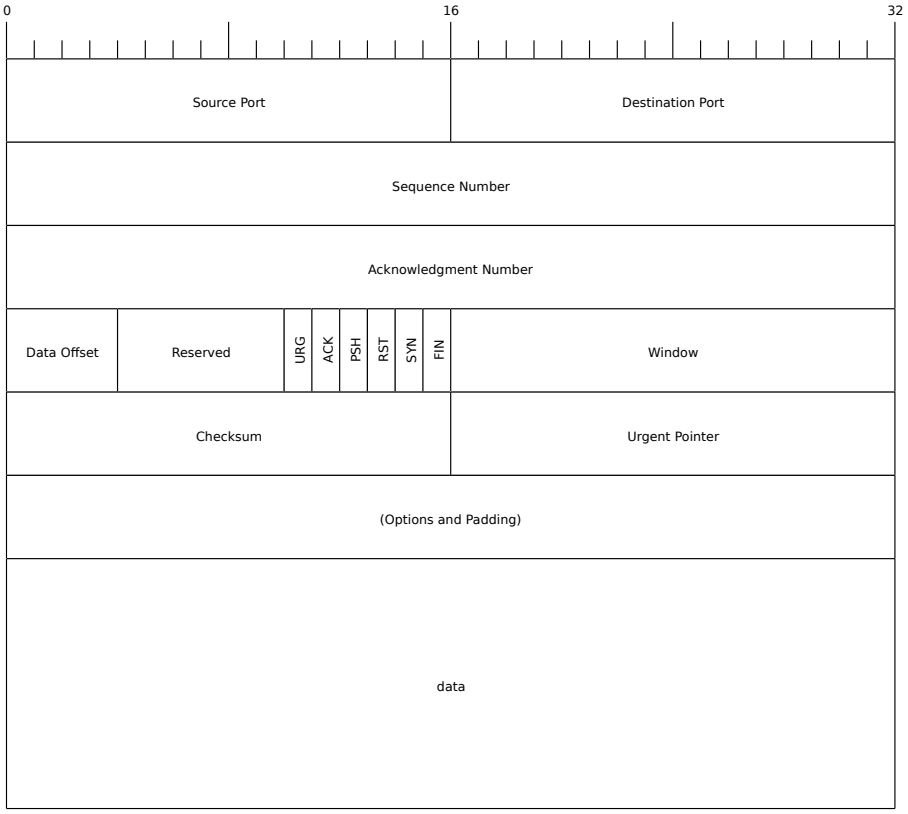

Figure 3.7: Example - Packet Diagram

3.8 Sequence Diagram #1 (PlantUML)

Engine: `plantuml`

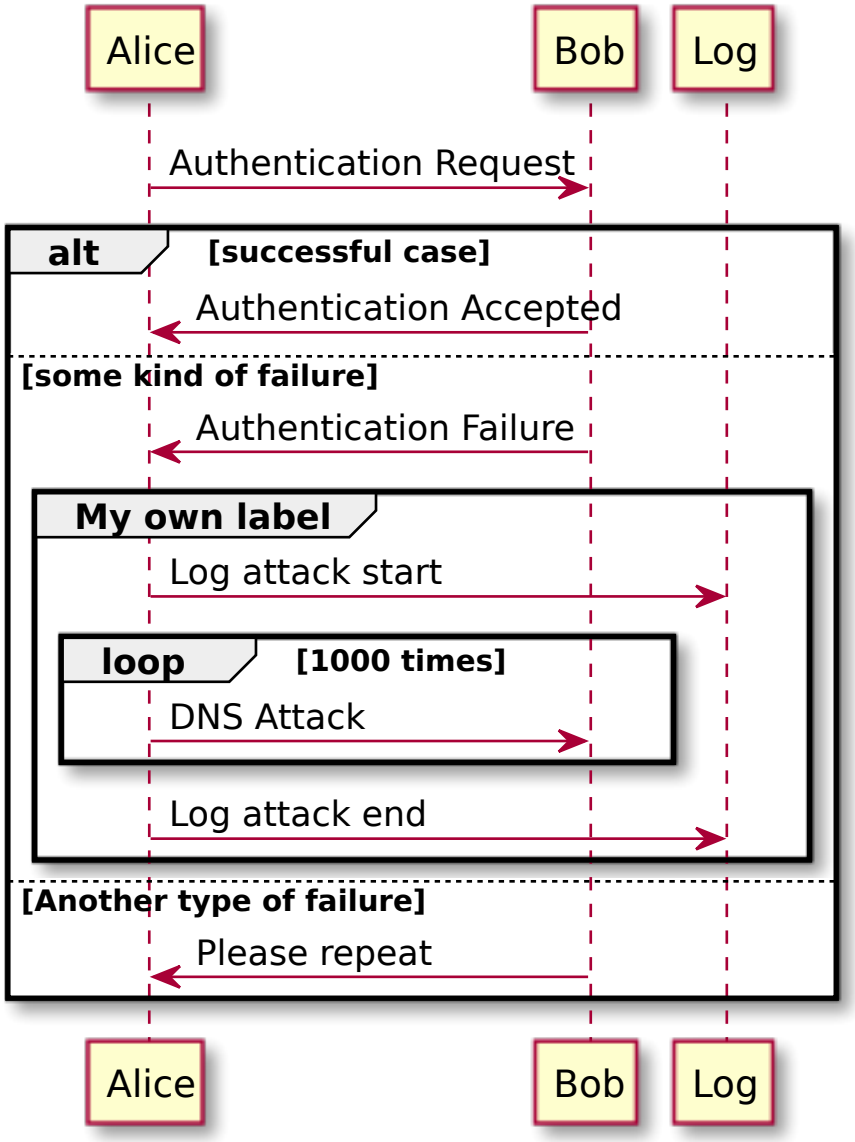

Figure 3.8: Example - Sequence Diagram - PlantUML

3.9 Sequence Diagram #2 (SeqDiag)

Engine: `seqdiag`

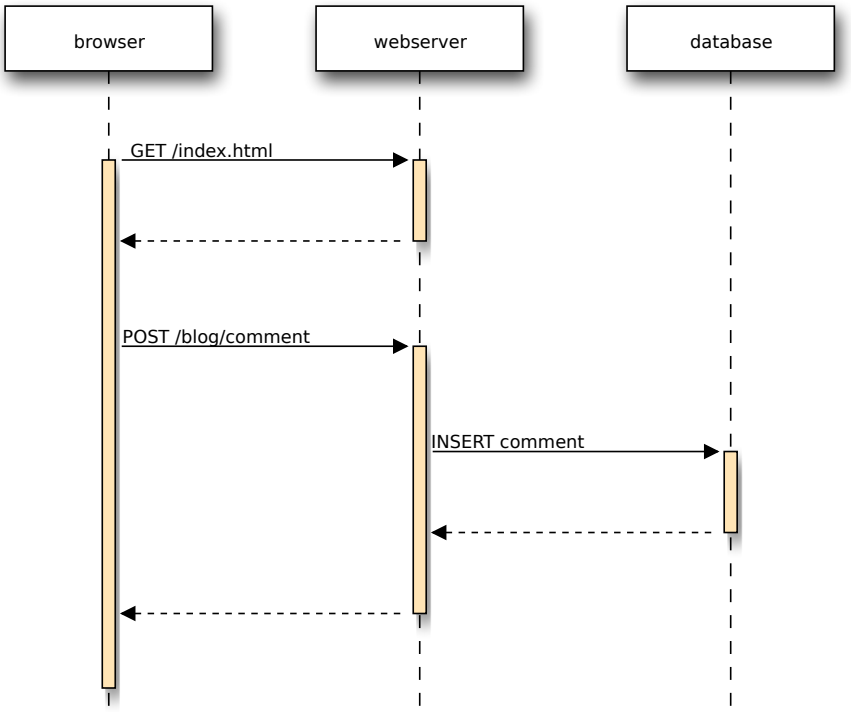

Figure 3.9: Example - Sequence Diagram - SeqDiag

3.10 Commit Graph

Engine: `pikchr`

> NOTE: pikchr is giving troubles in PDF/PNG (produced SVG output is only web-browser friendly)

3.11 Use Case Diagram

Engine: `plantuml`

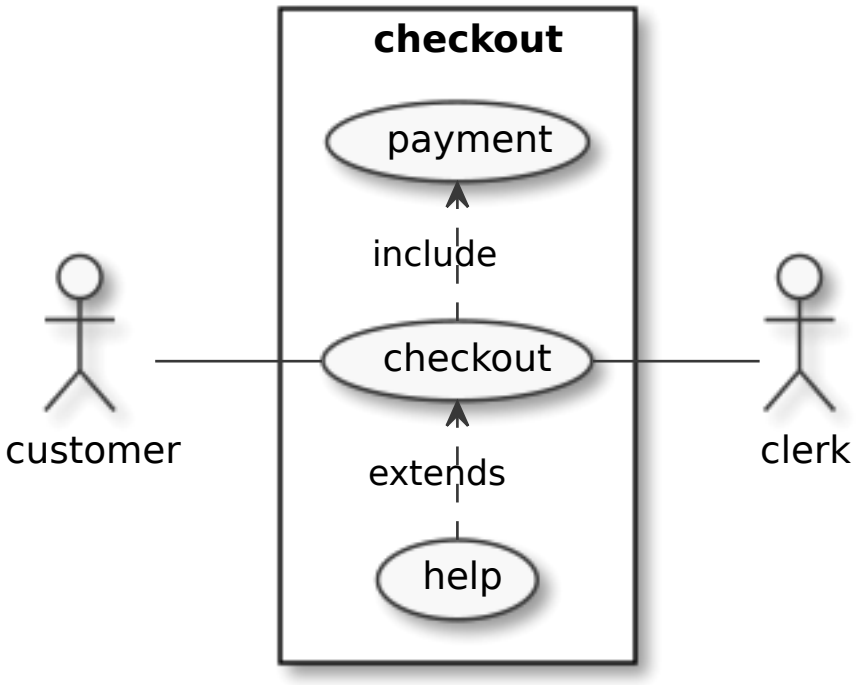

Figure 3.10: Example Block Diagram

3.12 Mind Map

Engine: `plantuml`

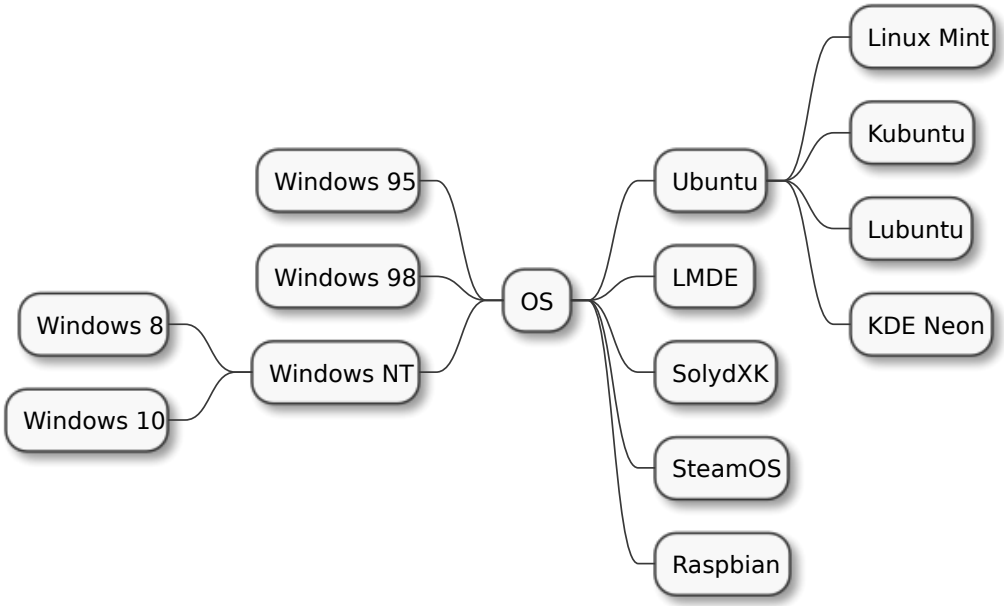

Figure 3.11: Example - Mind Map

3.13 PlantUML (More examples)

PlantUML supports more diagram types like timing diagram, gantt and many more.

You can use any of them just like in previous examples.

Check PlantUML docs https://plantuml.com/ for filling in diagram data.

3.14 Gantt

Engine: `mermaid`

> NOTE: mermaid is giving troubles in PDF/PNG (produced SVG output is only web-browser friendly)

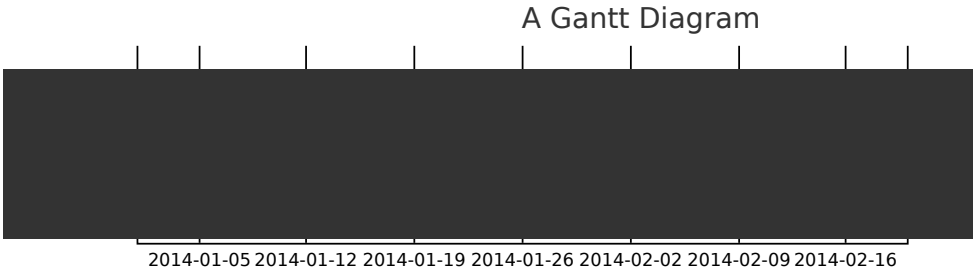

Figure 3.12: Example - Gantt