Architecture research week 1

Enterprise software is software created for the purpose of managing data and applications of an organisation. This is not to be confused with typical software solutions which have as a goal to satisfy the end-user.

Examples of enterprise applications are:

* Billing systems
* CRM: customer relation management systems
* ERP: enterprise resource planning systems
* E-commerce environments

# Architecture

## Why

When combining the typical applications with its counterpart the enterprise applications you get an enterprise architecture, a model which defines all important applications and their relations to each other. In martin fowler’s keynote he explains that this definition can be vague, what is important, who decides which is important but more importantly he explains why it is important.   
According to fowler’s hypothesis by maintaining, refactoring, and cleaning your codebase will allow you to create more functionality in the long-term allowing your software solution to be more competitive over a longer period even if this means a loss in the short-term.

## How to start

### Stakeholders and ideas

So how does one create a quality software architecture which is maintainable in the longer-term. In the start of a project, you will need to figure out what the core requirements and functionalities are which are only derivable from your stakeholders’ expectations. In Paul Rayner’s talk about event storming, he starts off with an explanation on how to create engaging ways to find user stories. The problem with for example user story mapping, a system where stakeholders start drawing out various parts of a system to create requirements, is that even though they are highly engaging they only focus on user interaction and often ignore the background systems which can be complicated.

In comes event storming, event storming starts with placing domain events on a timeline, domain events are processes which are a fact. They can happen in a step-by-step process; they can happen on a schedule, or they happen as the result of another event. All these events are part of a domain your experts might care about. To create a timeline, you simply write events on sticky notes and put them in chronological order. Because you do not know how big your timeline might be, create enough space which will not limit creativity. If during the process a question or discussion might arise that does not involve the current topic or is not resolved within a certain time allow stakeholders to simply put a note next to the event with their questioning. This allows to see in one visual where most of the problems might arise. To get the best results get different domain experts, if everybody is a software developer your result will eventually mimic a technical event timeline.

### Requirements

User story mapping is the process of creating requirements through the e yes of the end user. These stories can than later be used in agile teams to create a sprint backlog the scale and size of each story is figured out by how far into a project you are. A single user story can later be divided into a larger set of smaller stories, the larger story will than become an epic.

### Describing initial architecture

How to describe your architecture will come down to which stakeholder you are describing it to. A non-technical stakeholder might not know how or why you are using a certain language to compile your codebase. In Simon Brown’s guide to C4 software architecture models he explains how diverse levels of model might help in describing your project to a stakeholder.

Starting with C1 level this type of diagram puts personas and users central describing the distinct parts of your project into broad strokes and how different actors and systems might interact with each other during the process.

C2 is the level where application will be visible a banking system in C1 will now be separated into a front-end webpage, a mobile app, a back-end API which connects to a database model. Furthermore, each container of the application will start being labelled with the types of codebases it will use.

In C3 the user will completely be removed from the diagram and the different containers will be dissected into separate parts. The original back-end API will now be described into different sections like the REST controller, the security and email component and an account management system. The C3 and C4 models are meant for developers and architects to describe to each other how the next phase of a project might shape out.

Lastly the C4 model, in this version various parts of a container in C3 will be described. What kind of methods and classes will be called, where will errors be logged, which kind of abstractions might be used, how do the different models interact with each other. These types of models are useful if documentation will be longer lived, and new development teams or groups will be later added or replacing the current team.

# Architecture Theory

## Frameworks

At the start of a project a few choices will need to be made: one of the main choices is whether a project will be relying on a framework to help with development. Choosing between frameworks comes down to a set of different trade-offs: (Stukalov, 2018)

* Range of Applicability
* Development Speed
* Manageability & Flexibility

Diagram, timeline

Description automatically generatedAccording to Stukalov you can classify frameworks into 5 categories. Low code, Full-stack, Aggregators, Narrow Focussed and No Frameworks. With each there up- and downsides, it all comes down to the early 3 trade-offs. If a project has a limited timeframe with a clear scope which doesn’t need much customizability a full-stack framework will allow you to speed up your work process and create a product at a faster speed than a pure codebase.

Figure 1 5 levels of framework

To determine all the trade-offs you can create a matrix with all the different technologies weight against each other depending on what criterium one might find important.

## Architecture styles

### None functional requirements NFR

First off what are functional requirements. Functional requirements describe what a system must do. Simply said they are facts: send a message, create an log, update password. None functional requirements don’t describe a process but describe the parameters around a requirement. The message needs to be send within 10seconds, the password must be secured with SHA, the request needs to be 99% accurate.

### Trends and styles

There are many different ways to manage your architecture, in this chapter I will compare different architectures and how they might benefit which type of application. In the conclusion of this chapter I will defend why I might choose a certain type of architecture over something else.

#### N-layer

Graphical user interface, application

Description automatically generatedN-layer or layered architectures are architectures where physical tier of the architecture is separated with its own logical layers. These layers manage there own dependencies where a higher layer cannot be called by a lower layer. The tiers are physically separated are hosted on their own machines. Separating the tiers allows for easier scalability and makes the system more resilient but might add latency due to distance and network communication.

Reasoning to use layered architecture:

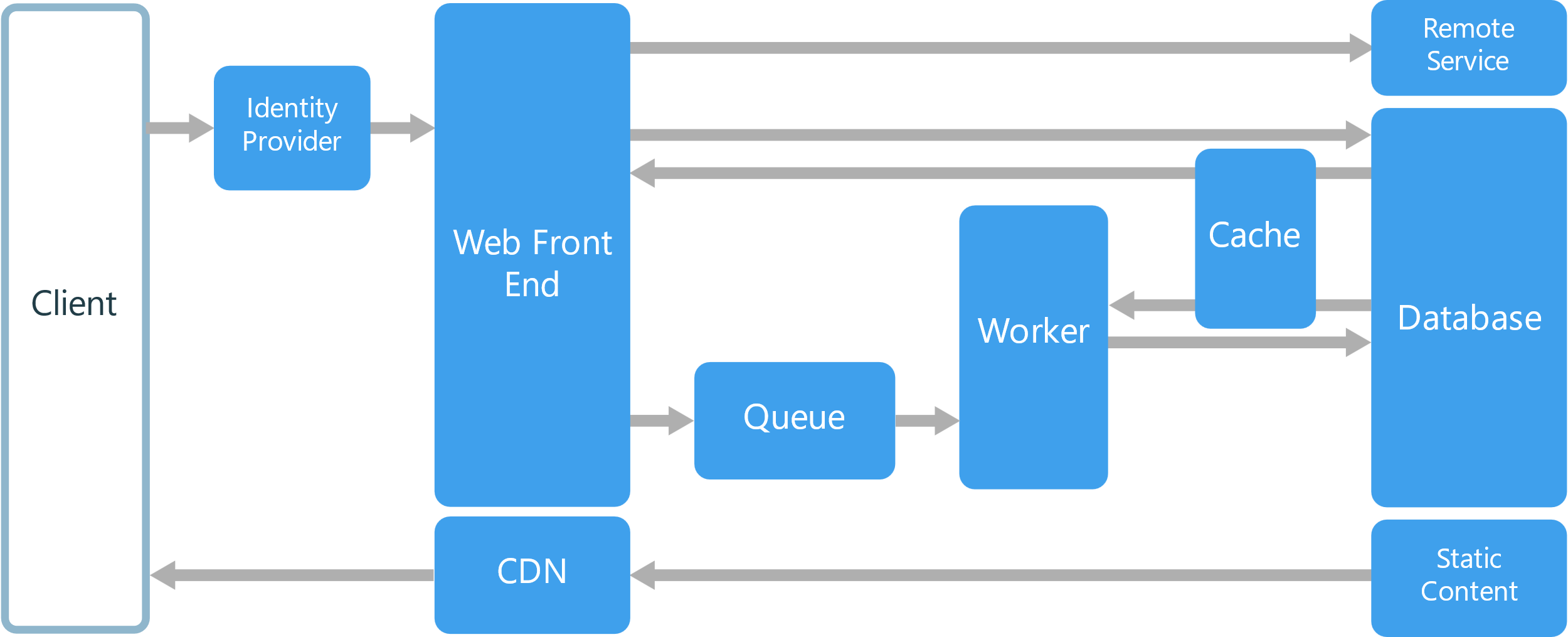
* Ease of use
* Easier to integrate with cloud platform
* Not environment or OS bound
* Typically used in IAAS

Down sides:

* Useless middle tiers
* Incompatible with monolithic design
* Network security

#### Web-queue-worker

This architecture consists like many out of a web front end and a database, unlike comparable architectures the server requests are handled of by a worker which receives long intensive tasks or large batch jobs. Trade marks of this type of architecture are multiple databases, a quick reading cache and is often paired with a identity provider and email service. The batches and jobs the worker receive are handled asynchronously from the web front end and other workers.



* Simple architecture
* Easy to deploy and manage
* Front end is stand alone and can be made as a SPA
* The worker can be scaled separately

Drawbacks:

* The front-end and workers can become too large without careful management

#### MIcroservices

Microservice architecture is a form of architecture where each business case or part of an application a standalone separate service is. This allows for small maintainable code bases which each can be handled by a small team. A service within the architecture has no dependency on the other services within the project and thus might operate with a their own models, database and codebase.

#### event-driven

A system in which events are delivered in real time, so when a consumer receives the event it can act immediately. Due to consumers and producers being decoupled the system can scale with ease. But this can lead to problems when events must be processed in order or if the logic that’s being processed isn’t changing.