Victor

2201741946

1. The notion of patterns, can be applied to software architecture. They are several of the most important **architectural patterns** (architectural styles), including: *The Multi-Layer, The Client–Server and other distributed, The Broker, The Transaction Processing, The Pipe-and-Filter, The Model–View–Controller (MVC), The Service-Oriented, and The Message-Oriented.* **Chose one** from those architectural patterns list which you might clearly understand, and describe it in detail as much as you can.

Model

Model berfungsi untuk mengatur data, fungsi dan aturan dari aplikasi.

View

berfungsi untuk mengatur tampilan atau output yang tampil di layar, tidak hanya berupa data, namun juga termasuk komponen lain, seperti gambar, video, diagram, dan sebagainya.

Controller

Controller merupakan program yang mengatur menerima input dan menjalankan beberapa perintah untuk dijalankan di model.

1. Design documents serve two main purposes. Firstly, they help you, as a designer or a design team, to ***make good design decisions***. The process of writing down your design helps you to think more clearly about it and to find flaws in it. Secondly, they help you ***communicate the design***to others. Mention thoroughly how to write a good design document.

**Overview and stakeholders**

Start at the beginning. What problem are you trying to solve? If you jump straight into solutions, it will be hard for people to orient themselves and will inevitably lead to misalignment and misunderstanding. It’s worth spending 2 or 3 sentences to effectively set the context for your spec.

Then, briefly state your proposed solution. This should be enough for most people to decide whether they should continue reading and should be understandable by someone who is not familiar with the project. Between a few sentences and two paragraphs should be enough.

**Background**

It’s unlikely that writing the design document is the first time you’ve thought about the problem. The background section is an opportunity to bring readers up to speed, and share the context you have on the problem space.

What are the motivations for the project or design? Is there any historical perspective that will help people understand the proposal? Has someone tried to solve the problem in the past? If so, why are those solutions no longer appropriate? Are there any other things going on that will affect the design?

**Goals, non-goals, and future goals**

In order to build alignment and communicate a definition of done, it is important to clearly articulate the goals of this work. The best goals are simple, truthy sentences that describe a future state of the world. Unlike OKR, it's fine for these goals to be hyper-specific. Projects will often have 3-5 goals.

**Non-goals**

As well as explaining what you want to achieve, it is equally important to say what you are explicitly not addressing. These can sometimes be hard to identify, but imagine what another person might expect to be coupled with this work.

**Future goals**

Future goals are an opportunity to list things you want to do in the future, but have descoped for this phase of the project. In other words, these are things you want to make sure your solution doesn’t accidentally make difficult or impossible.

**Detailed design**

This is the meat of the document and also the most variable. Depending on the project, the size of your team, or number of stakeholders, it may be a few paragraphs or a few pages. It will often contain pseudo-code, schema definitions, or flow diagrams.

**Third-party considerations**

Today it is common to rely on 3rd party platforms to support our development work, whether this be part of AWS or GCP, or a whole separate service. It’s worth thinking through the implications of using a third-party and looking ahead for potential future issues.

The cost of these services is often less than an engineer’s time, but sometimes it can scale unexpectedly. Quickly think through how the service is billed and do a back-of-the-envelope calculation of what could be expected once the service is fully rolled out.

While security and privacy considerations were addressed in the detailed design, when using a 3rd party there are specific things to think about and call out.

For example, if the 3rd party is being used to perform operations on customer data they will likely be considered a subprocessor under the EU’s General Data Protection Regulation (GDPR). So, do you need a data processing addendum? Do you need to collect and review SOC2 reports? Sometimes customers will require notification of new subprocessors, which will affect the roll-out plan below.

## **Work estimates**

For non-trivial changes, provide a breakdown of the work and tasks. How long will each phase take? What work is parallelizable? What dependencies on other teams are there?

## **Roll-out plan**

It’s unusual that you can roll out your project as a single change. In this section, discuss how changes to models and APIs will need to be staged. Will you be rolling out incrementally to your users using feature flags?

Discuss your revert path. If something goes wrong, how will you back out partway through the process while leaving systems in a healthy state? Identify the biggest risks you see, and spell out how you’ll detect and mitigate them.

## **Alternative approaches**

It’s not enough just to share your chosen solution. By explaining approaches that you rejected you can reduce time handling objections from other stakeholders and focus the discussion on your chosen design. Make sure to explain why other approaches seemed inferior or wouldn’t work.

It’s also not unusual for information to emerge during the design process that escalates an alternative approach to the primary approach. If this happens, try and avoid the [sunk cost fallacy](https://en.wikipedia.org/wiki/Sunk_cost).

## **Related work**

Are there products — internal or external — that are similar to this project? Are other teams being faced with similar challenges?

As an example, if you are building your own NoSQL database this section might include a feature matrix comparing your requirements to existing database offerings.

## **Future work**

This is where you can help prevent [bike-shedding](https://en.wikipedia.org/wiki/Law_of_triviality) and scope creep. Identify anything you’re not addressing with this particular design but that should happen in the future or that would be a logical follow-on project? This is often a more detailed description of the future goals, or perhaps how some of the non-goals might be addressed in the future.