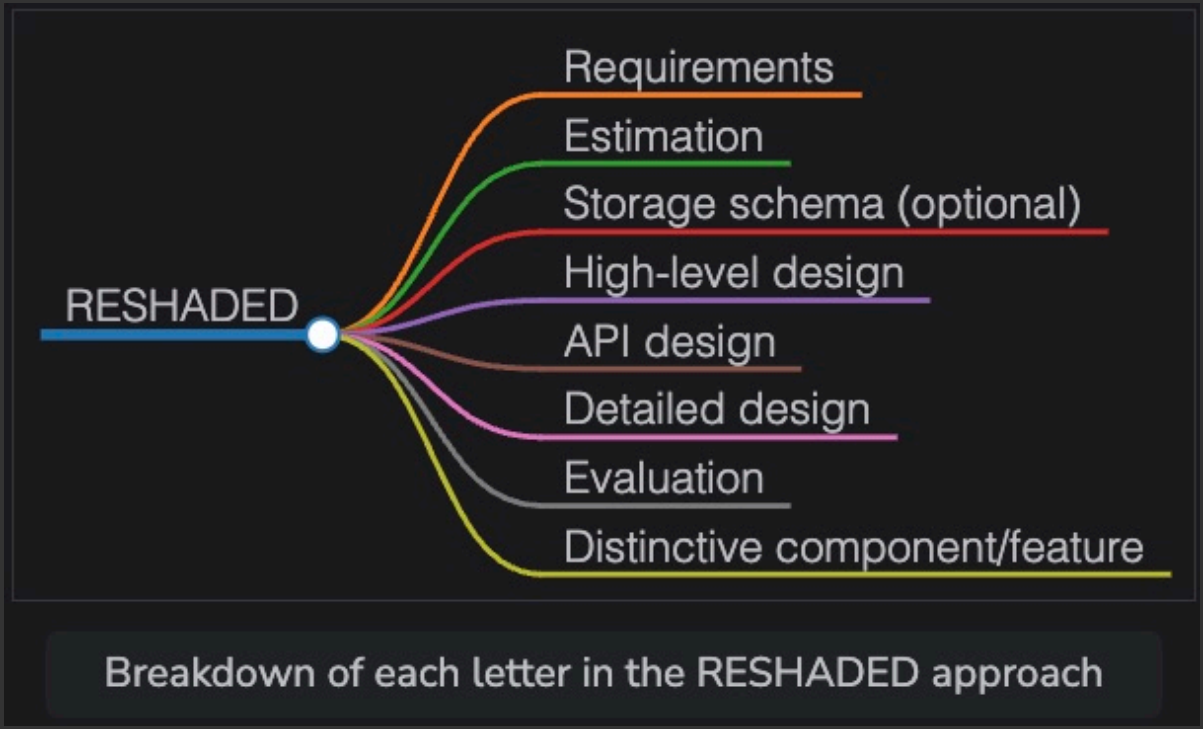
System design problems are not straightforward. We don't have a universal formula that we can use for all design problems. However, we can use a high-level common strategy to set the tone for a good solution to any design problem. We call this the RESHADED approach.



Breakdown of each letter in the RESHADED approach

Requirements: During this step, we gather all the requirements of the design problem and define its scope. Requirements include understanding what the service is, how it works, and what its main features are. Our goal in this step is to gather the functional and non-functional requirements of a service we are about to design.

Estimation: As the name suggests, this step estimates the resources required to provide the service to a defined number of users. By resources, we mean the hardware or infrastructural resources. Some sample estimation questions are the following:

How many servers will we require to provide smooth services to 500 million daily active users (DAU)?
How much storage do we need if we have to store 125 million tweets per day, and 20% of tweets contain media?

Storage schema (optional): This step involves articulating our data model—that is, we define which tables we need and what type of fields are part of each table. However, this is an optional step, and we may not exercise this effort in every design problem.

High-level design: This step involves identifying the main components and building blocks we'll use to design our desired system. We do this by getting inspiration from our functional and non-functional requirements.

This is considered the first step toward the complete design of our system and therefore requires further iteration and improvement. Primarily, this section will focus on fulfilling the functional requirements.

API design: The goal in this phase is to build interfaces for our service. Using these interfaces, users can call various services within our system. These interfaces are in the form of API calls and are generally a translation of our functional requirements.

Detailed design: The detailed design starts by recognizing the limitations of the high-level design. We'll capitalize on these limitations to evolve our design. During this step, we'll finalize our design by mentioning all the components and building blocks that we'll use.

Evaluation: This step will measure the effectiveness of our solution. In other words, we justify how our design fulfills the functional and non-functional requirements.

Distinctive component/feature: At the start of this lesson, we discussed how no one-size-fits-all solution exists. This step is to identify a unique aspect for each design problem and discuss it. For example, the Uber design problem has payment service and fraud detection as its unique feature. In contrast, Google Docs has concurrency control, which is required when different users want to edit the same section of a document simultaneously.