Now onto our **technical overview** - Our **system design principle** is to be as **User-centric** as possible, and as well as leveraging as much as the **power of iterative design** through making **mvps** to get **market validations** and **fast feedback** from our users.

From the **integration** and **implementation perspective**, we’ve chosen to make a **web app** instead of a **chrome extension**. A **chrome extension** relies **heavily** on the **third-party website**, it has the ability to **modify** the third-party websites but it also requires our users to use our product **on** a third-party website. Our goal, however, is **not only** to help our customers have a better **amazon** shopping experience, but rather, a better way to find the things **they want**, **even across** the **entire universe of internet** and **all the e-commerce providers**. But for testing purposes, we decided to start with **just Amazon**, cause it has the **best documentation for its APIs**.

So **let me walk us through** how **our system's components** and how our **information flow works**, **starting** from how we **process** our users' **natural language queries**.

First, our **user**s would **access our web app** through their **browsers**. They would input their **preferences or search queries** in **natural language— just like the conversations** you’ve seen a **few slides ago**. As the user **interacts with the interface**, the **frontend** sends these inputs to the **backend server** through **REST API calls**.

Upon **receiving** a user's **natural language query**, the **backend serve**r would **utilize the OpenAI API** to **process** this **input**. And the model would help us **translate** the user's **descriptive language** into a s**tructured query** that is **suitable for Amazon's product searching scheme**.

With this **refined query**, our **backend** would interact with the **Amazon Product Advertising APIs** to **fetch relevant product** details **and user reviews**. This **ensures** that the products we **retrieve align closely** with what **our user is looking for**.

At the same time, the backend **manages authentication**, **validates requests**, and **interacts with our MongoDB database** to store and retrieve user profiles, preferences, and any **cached product information.**

Once we have the product data from **Amazon**, the backend **sends the product reviews to the OpenAI API again—this time** to **generate summaries and personalized recommendations**.

Then our backend would **compile a structured response** from what the **AI returns**. This response includes the **list of products matching the user's query, summarized reviews, and the recommendations.** It then sends this information back to the frontend.

Last, throughout this entire process, our logging and monitoring tools will work behind the scenes.

Based on that, you've probably also figured out our preferred tech stack:

For our user-interface development, we're using **React** along with **Material UI.**

On the backend server side, we've chosen **Python with Flask**

Regarding external services, we're integrating the **Amazon Product Advertising API** and the **OpenAI API**. These services enable us to fetch **real-time product data** and leverage **advanced natural language processing** to understand **user queries** and **summarize reviews effectively.**

For our database, we're going with **MongoDB** because of its **flexibility and scalability.**

For continuous integration and deployment, we're using **Jenkins** to automate our build, test, and deployment processes.

And Lastly, for deployment
we are considering using cloud services like AWS or Google Cloud Platform for the deployment, whichever one is cheaper…