Use Cases and Logical Architecture

* X00130180
* Jordan Williams
* Leveraging Istio to Enforce Rules-based Authorization

## Section 1: For Each Use Case:

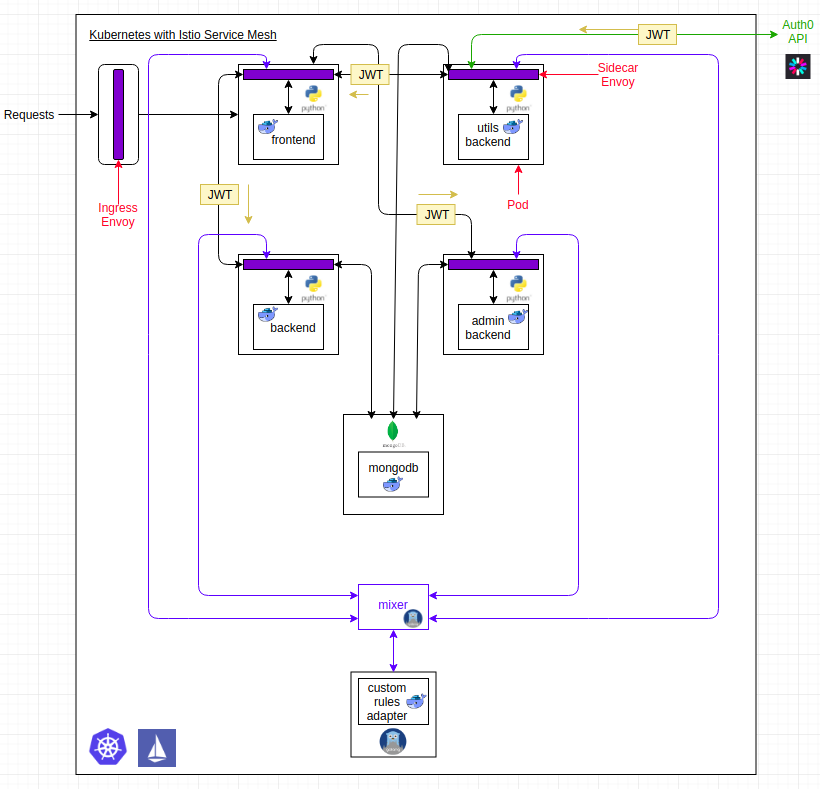
|  |  |
| --- | --- |
| Title (goal) | View Homepage |
| Primary Actor | User |
| Story | Any registered user will be able to view information regarding the service and navigate the UI. |

|  |  |
| --- | --- |
| Title (goal) | View Medical History |
| Primary Actor | Member |
| Story | A member will be able to view their own medical history. |

|  |  |
| --- | --- |
| Title (goal) | Update Patients Medical History |
| Primary Actor | Doctor |
| Story | A doctor with elevated privileges will be able to view and edit any of their assigned patients’ medical records. |

|  |  |
| --- | --- |
| Title (goal) | Data Restricted Access via Time |
| Primary Actor | Doctor |
| Story | A doctor can only access the data during the designated hours of 9am-5pm Monday to Friday. |

## Section 2: Logical Architecture



## Logical Architecture Discussion

The **minion** is the node on which all the services run. You can have many minions running at one point in time. Each minion will host one or more POD. Each **POD** is like hosting a service. Each POD then contains the Docker containers. Each POD can host a different set of Docker containers.

The sidecar pattern plays a big part in how Istio chooses to enforce its access control. This is done via an envoy sidecar. It allows for the deployment of components of an application into a separate process or container to provide isolation and encapsulation. This pattern can also enable applications to be composed of heterogeneous components and technologies.

If sensitive info is required, it is fetched from the DB and displayed to the user on the webpage required that they indeed have the correct level of privilege.

This pattern is named *Sidecar* because it resembles a sidecar attached to a motorcycle. In the pattern, the sidecar is attached to a parent application and provides supporting features for the application. The sidecar also shares the same lifecycle as the parent application, being created and retired alongside the parent. The sidecar pattern is sometimes referred to as the sidekick pattern and is a decomposition pattern.

A sidecar service is not necessarily part of the application but is connected to it. It goes wherever the parent application goes. Sidecars are supporting processes or services that are deployed with the primary application. On a motorcycle, the sidecar is attached to one motorcycle, and each motorcycle can have its own sidecar. In the same way, a sidecar service shares the fate of its parent application. For each instance of the application, an instance of the sidecar is deployed and hosted alongside it.