Authentication and Identity Management of the Microservice Proposal

Taking or delegating responsibility of authentication within the system architecture

Definition of a Microservice

- Small service decomposed from a monolith
- Isolated and independently deployable
- Stateless and less fragile when changes are introduced
- Single responsibility
- Advice was provided in the Phase I - Pregame document

Aspects of Authentication and Identity Management

Name	Explanation	Example
- Responsibility	System concerned of authenticating users?	Core, external service or microservice itself
- Locality	Where is user data stored?	A single database, all databases
- Authorization	How do we identify user and roles?	CompanyId, DaAppInstall ...
- Statefulness	How is the state of authentication shared between services?	In the request, in shared or separate sessions

Aspects of Authentication and Identity Management

Name	Explanation	Example
Responsibility	System concerned of authenticating users	Core system, external service or the microservice itself
Locality	Where user data stored	One single database, every database, a separate database
Authorization	How users and roles are identified	CompanyId, DaAppInstallId, a separate combined id
Statefulness	How the user state is synchronized between services	In the request, in shared or separate sessions, via a token

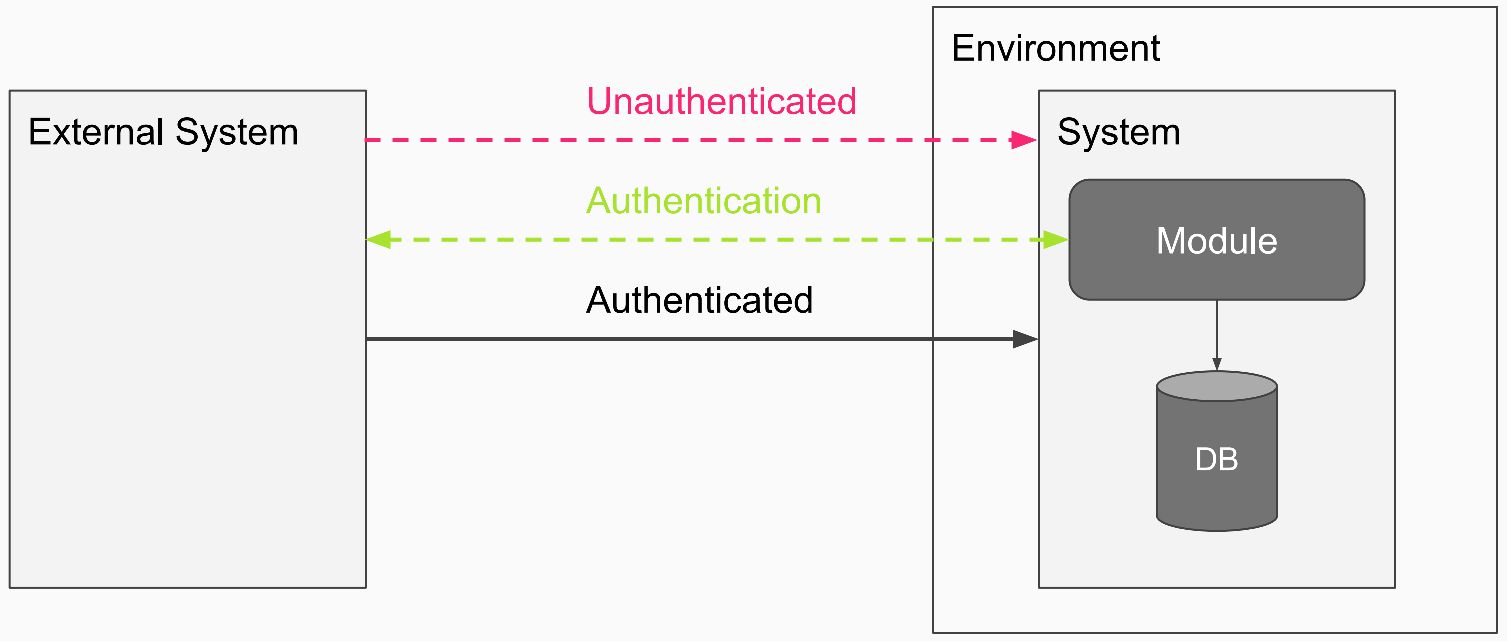
Examples

Here are four examples 1 … 4 increasing from basic to more extreme implementations.

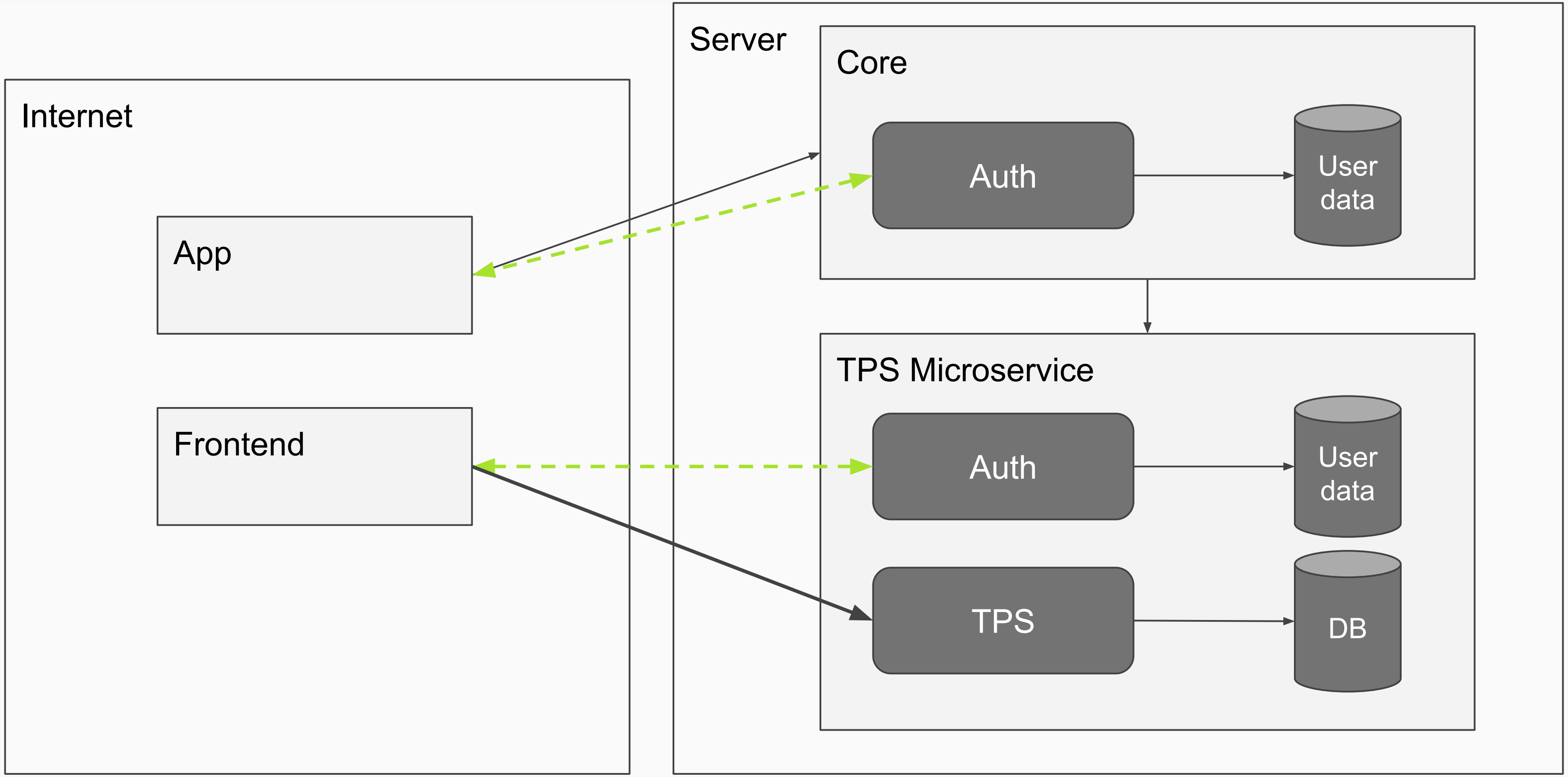
The four aspects are discussed after each figure:

- Responsibility
- Locality
- Authorization
- Statefulness

Symbols used in the Examples



Ex 1


Internet
App
Frontend
Server
Core
Auth
User data
TPS Microservice
Auth
User data
TPS
DB

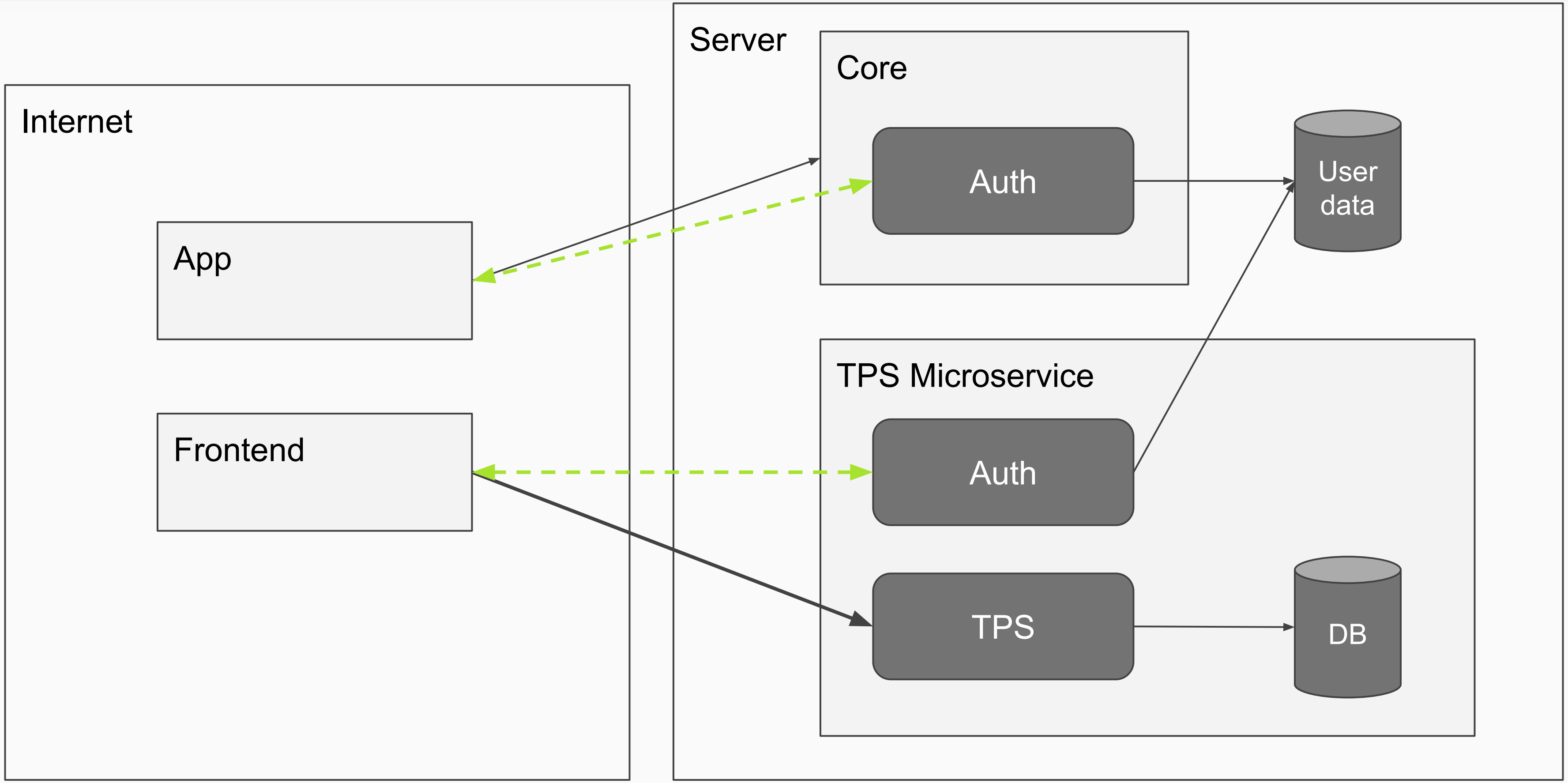
Ex 1 - Aspects

The TPS microservice authenticates its users. It has its own database with user and company data.

Authorization is handled by checking the user data in the database. Sessions are handled by the microservice. So the state resides in the microservice.

The microservice is totally independent, except for the fact that the data that is mutated in other systems must be synchronized in some fashion.

Ex 2

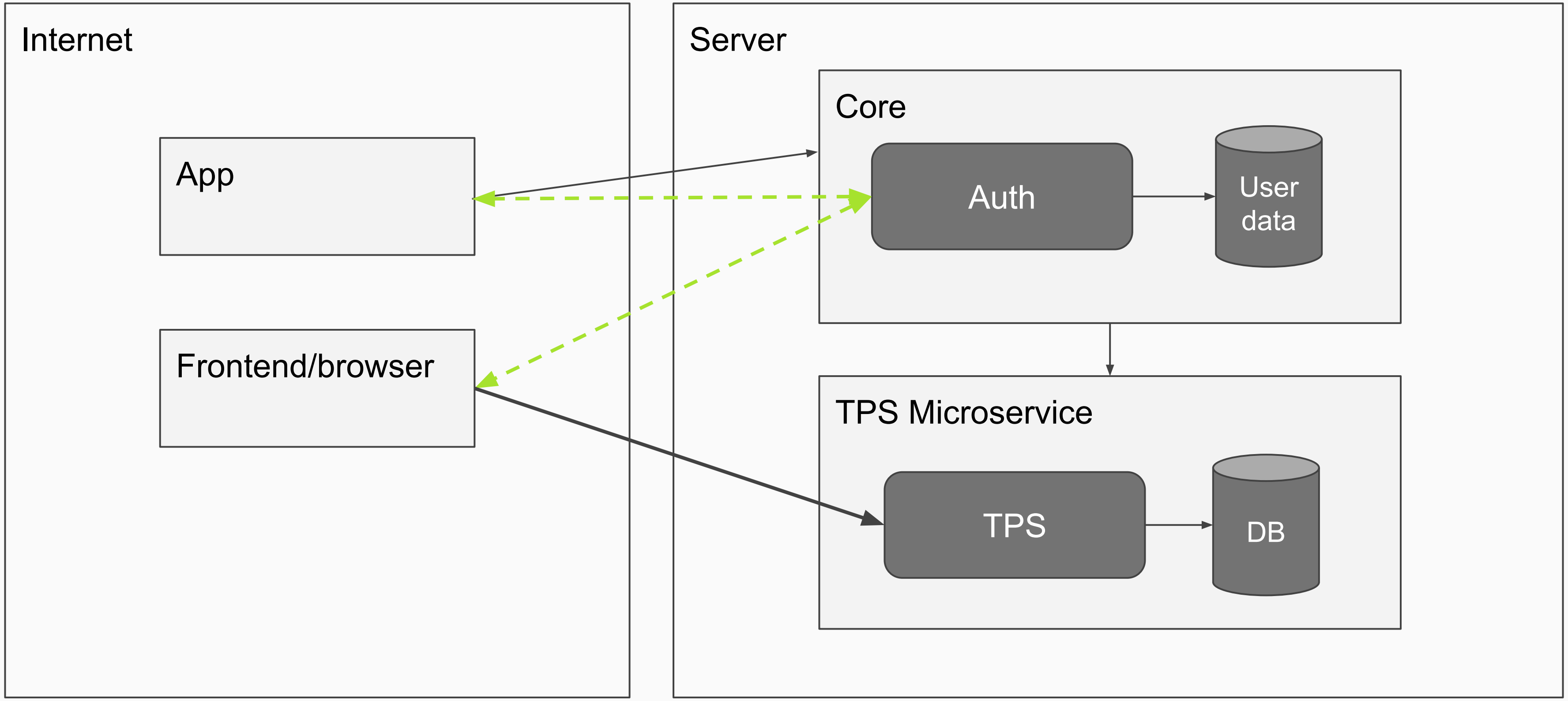

Internet
App
Frontend
Server
Core
Auth
User data
TPS Microservice
Auth
TPS
DB

Ex 2 - Aspects

Like example 1, authentication is handled by the microservice. Only this time it connects directly with a database that stores user information.

Depending on how sessions are handled, the state can be shared amongst systems. But when other systems are required to make use of the microservice, more and more sources that contain the state of users need to be shared with the microservice.

Ex 3


Internet
App
Frontend/browser
Server
Core
Auth
User data
TPS Microservice
TPS
DB

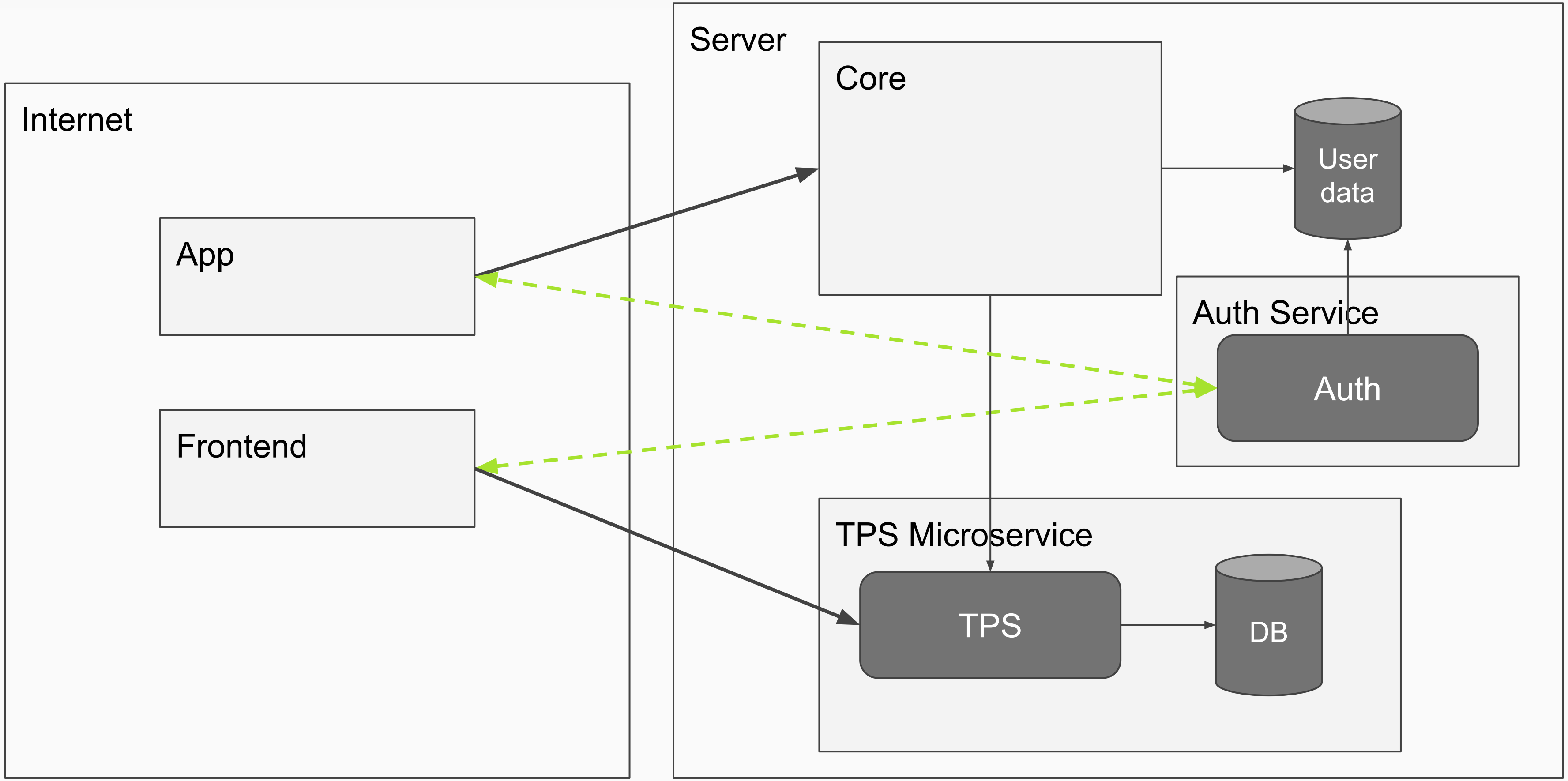
Ex 3 - Aspects

In example 3, the Core system is the only system able to provide authentication tokens. This token must be used to transfer authorization and identity information to the microservice in a stateless manner, because the microservice has no concept of the state of authentication.

A JWT can be used to transfer state in this case.

If more systems have to make use of the microservice in the future, they depend on the core system anyhow.

Ex 4


Server
Core
User data
Internet
App
Auth Service
Auth
Frontend
TPS Microservice
TPS
DB

Ex 4 - Aspects

In this example, future systems don’t depend on the core system. But the core system does depend on the Authentication service in order to make use of other microservices.

Concluding

- Managing invalidation of JWT’s
  - Invalidating individual tokens conditionally
  - https://stormpath.com/blog/token-auth-spa
- Keeping payloads up-to-date when data changes
  - User switches from company, therefore the frontend must act!