**ARCHITECTURE**

Considerations when choosing an architecture

The first and foremost thought of using a monolith architecture was discarded after a couple of discussion regarding the future scope that the project might contain. As stated in this report, the product is expected to be expanded to market outside of the United Kingdom. Such future requirements might cause changes to the certain structure of the business logic that will be implemented. In event of decision to build the application based on monolithic architecture, we came across the issue of scalability.

A simple case from the application perspective was the login and registration functionality. If the application was to be initially designed for UK region, specific login and authentication API service like Microsoft authentication library(MSAL) can be incorporated, or the usage of Google authenticator as a Multi factor authentication (MFA) vendor can be utilized. Given a scenario of expansion into other countries, some of these MFA vendors might not be available, for example google services are not fully and legally available in China[1]. This could result in changes to the code base, by considering region specific product restriction and availability. In such cases, if the login and registration action can be built and deployed as a single entity then, refactoring it in future might be easier without impacting any of the other services. But if the initial implementation includes a single code base for all the entire application feature, then complexity with be multiplied when making changes.

This led us to consider alternative approaches, which made us to weigh in the benefits of **Service oriented Architecture (SOA)**  for our scenario. As the need for scalability, reliability and resiliency were considered as a primary need, we were convinced at choosing other architecture pattern over monolith. SOA is a response to building a software that can adapt to distributed and heterogenous environment as in our case. It is an approach to distributed system architecture that employs loosely coupled service, standard interface and protocol to deliver seamless cross platform integration. It is used to integrate widely divergent components by providing them with a common interface and protocol to integrate.

Another supporting factor was that the actors in our application have varied uses, such as. a student might use this application for searching and applying for opportunities posted by an employer. Given the case where the recommendation system was built as a separate service and independently maintained, and it failed for some reason, then the student should still be able to apply to job if he searches for a specific job using the search options which can be an independent service of its own. Such kind of failures are gracefully tackled in this architecture, allowing the application to be reliable. SOA is a loosely coupled architecture, which means that a failure in one system will not make an impact on another.

In SOA, all applications must be able to simultaneously receive and change data at the source level. As a result, complicated data synchronisation mechanisms are not necessary for SOA services. This strategy, though, also develops inter-service dependencies, which is undesirable in a microservices design.

As stated earlier, we adopted the agile methodology for this project and building independent abstract services promotes agile ways of working, with independent teams building segments of the application. This has proved to improve development process, and allows each service to be developed, tested, and updated without affecting any of the connected services.

Why didn’t we choose microservice.?

When comparing this with the widely used microservice architecture, we realised that in a SOA model, developers can reuse components which relates to of enhancing efficiency and scalability. However, if reusability was followed in microservice architecture then it would result in reduced agility, resilience and fault tolerance, since reusing a component will create dependencies across different services. Instead, to improve efficiency and maintain high degrees of independence, developers use duplicate data or reuse code in a microservices design.

Given the diversity of the application and assumption on the timeline set by the client, we decided to go with SOA architecture, as the initial investment and setup time for microservice is relatively large and might not be suitable in our case. By leveraging the advantages of sharing a common architecture, SOAs simplify development and troubleshooting. This sometimes compromises on the speed of operation, but favours in discarding code duplication.

Since every data in the system is interdependent, and every service will act upon the same data set, like in case of recommendations, the data from job posted from the organization and the preferences added by the student or external candidate are correlated and presented to the user, it was a well-suited choice for us to adapt Service oriented architecture.

Identifying relevant services

As per the plan we decided to divide and develop the entire application into the following services. These services have a distinct objective and their own business logic relying on a single data source. Each of these services access and update data at source level at the same time, therefore services do not need to include complex data synchronization models. Each service undergoes updating, testing, deployment, and scalability. With SOA, each of these following services share a common communication mechanism called an enterprise service bus (ESB).

The enterprise service bus, often known as an ESB, stated in the below architectural diagram is a design pattern that uses a centralised software component to carry out application interactions. It handles connectivity and messages, executes routing, transforms data models, and even control the composition of numerous requests. For reuse by new applications, the ESB can make these connections and transformations accessible as a service interface. To achieve the highest level of efficiency, the ESB pattern is often implemented utilising a specially created integration runtime and toolset.

Major business and domain-specific concerns are decoupled and placed in different, independent code bases. By breaking down activities into smaller processes that run independently of one another and contribute to the overall whole, SOA make any complexity visible and easier to manage, without reducing it.

* Service for authorization/permission management – Registration and login – As stated in the justification section for choosing this architecture, we stated that the client has plans to expand the application to different regions outside UK in future, it is anticipated that the library and the vendor used for building the authentication and authorization feature vary for each country based on their restriction imposed by the government. Therefore, the app-registration and login for each user is developed and maintained as a separate service.
* Service for candidate and organisation contacts search/filter – as the number of search parameters are bound to change (increase in most likely case) depending on various factors, and result of user testing feedback, the search and filter action will be built as an independent service.
* Service for Candidate and Organization contacts Recommendation : The Recommendation feature will be based on multiple criteria such as region in which the users are located, wages as per the norms, ongoing market conditions, and much more. Therefore, changes are expected to come in terms of business logic. Building this as a separate feature will lead to ease of updating the recommendation criteria whenever . Teams can test out new features and undo them if they don't work. These speeds up the time it takes to roll out new features and makes updating code simpler and deploy without interfering with the working of other services.
* Service for job posting and candidate job application – One of the primary features of the system is to allow student to apply to opportunities posted by organization. This feature should be developed, maintained and deployed independently to allow a **Highly maintainability and testability** along with reliability of this service, as a dedicated team can be used to actively monitor and rectify any issues if it occurs.
* Service for report generation – This feature is served only to certain section of the application user and need not be a feature available to all the connected clients.
* Service for notification – We realized that a notification service to perform omni channel capabilities will be needed to inform the users about the progress of their application. This service is for future implementation as our database consideration don’t track the response from the notification service.

Diagram

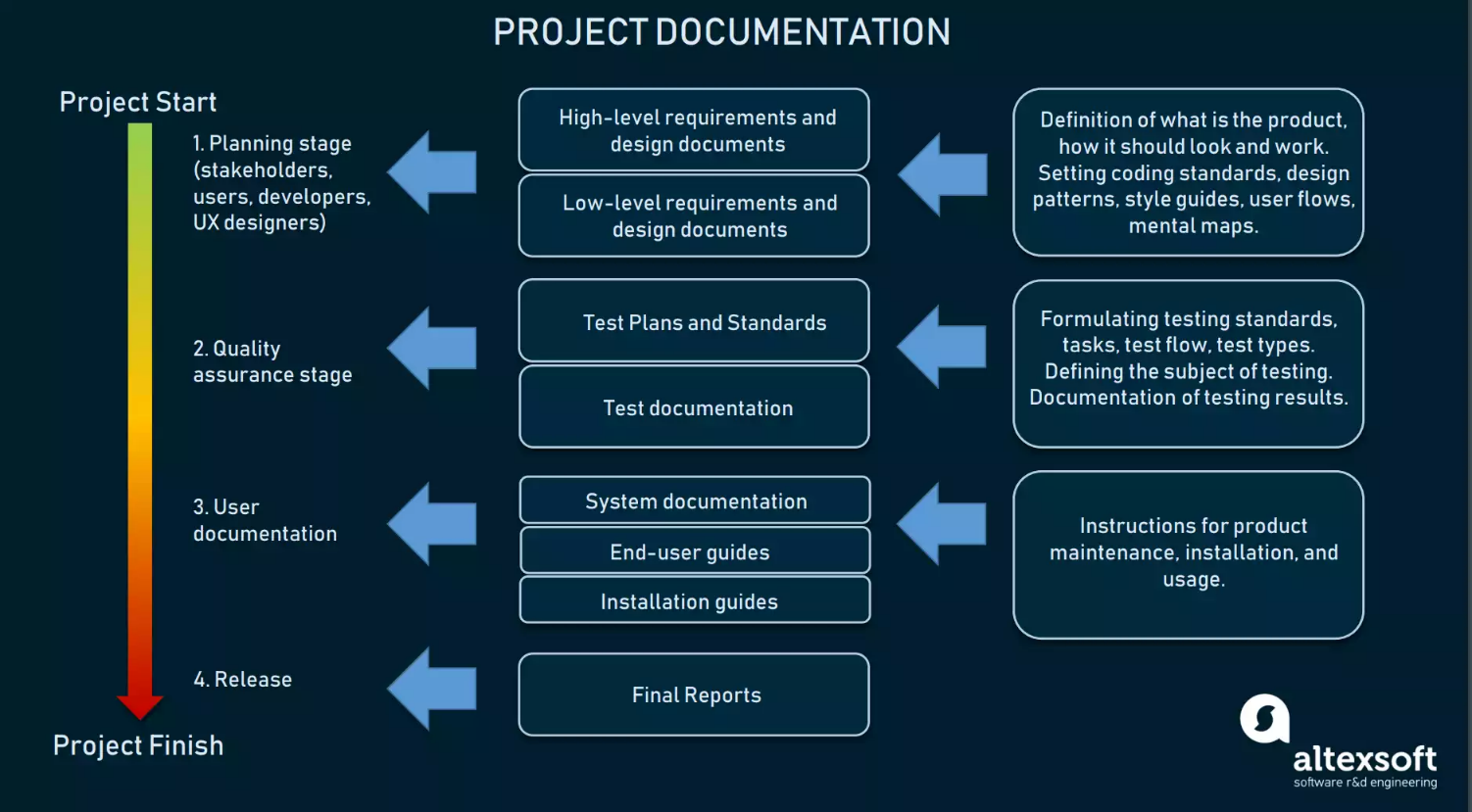
Description automatically generated

Certain factors for choosing Service oriented Architecture were:

Autonomy and loose coupling - Because of this loose coupling and the way the services are published, development teams can save time by reusing components in other applications across the enterprise[3].

Interaction over components- Each module or service will interact with each other, to perform certain data validation or for data gathering.

Abstraction – The end user or client need not know how the recommendation system works, therefore abstracting the service level code. The client will just have to show the computed result of recommendation done by the server.



https://www.altexsoft.com/blog/business/technical-documentation-in-software-development-types-best-practices-and-tools/

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

What questions were put when choosing an archietecture

1. Page 171 - how does your functional requiremnte suit ths

CLIENT – SERVER ARCHIETEXTURE

The main characteristics of is as under-the differences in their performing tasks the client and server can be differentiated from one another.

The client and server generally function on different computer platforms.  
One or more servers may be connected by the clients.  
Multiple clients may be connected by the servers at the same time.  
Without affecting each other the client or server may be upgraded.  
By requesting a service the clients always initiate the dialogue.

In a client/server environment a client PC almost does the following

screen handling  
menu or command interpretation  
data entry  
help processing  
error recovery

**Learn about rchitecture sinkhole anti-pattern**

Why not use Layered archietcture

Every layered architecture will have at least some scenarios that fall into the architecture sinkhole anti-pattern. The key, however, is to analyze the percentage of requests that fall into this category. The 80-20 rule is usually a good practice to follow to determine whether or not you are experiencing the architecture sinkhole anti-pattern. It is typical to have around 20 percent of the requests as simple passthrough processing and 80 percent of the requests having some business logic associated with the request. However, if you find that this ratio is reversed and a majority of your requests are simple passthrough processing, you might want to consider making some of the Considerations | 7 architecture layers open, keeping in mind that it will be more diffi‐ cult to control change due to the lack of layer isolation.

Another consideration with the layered architecture pattern is that it tends to lend itself toward monolithic applications, even if you split the presentation layer and business layers into separate deployable units. While this may not be a concern for some applications, it does pose some potential issues in terms of deployment, general robust‐ ness and reliability, performance, and scalability.

References :

[1] Multi-factor Authentication (MFA) workarounds for Chinahttps://www.coloradocollege.edu/offices/its/guides/mfa-china.html

[2] SOA vs microservice - <https://www.crowdstrike.com/cybersecurity-101/cloud-security/soa-vs-microservices/>

[3] https://www.ibm.com/cloud/blog/soa-vs-microservices