6/26/2024

Handover Document

Quantum TLC

Table of Contents

Introduction

This document will indicate how far the project is in terms of readiness, while also giving an idea of where the previous project left off.

Documentation

To gain knowledge on how we started this project and what requirements we followed, start by looking at the following documents.

- Project plan: this gives an indication of how we approached the project.
- GDPR: Get knowledge and insights on why this is important and what we had to do for it.
- Software Requirements Specification: Will show all the FRs and NFRs of the project with a small description of the important nonfunctionals.
- Technical Design: Get an understanding of how microservices work, while also understanding the important terms that come along with this.
- Security Design: This is important on how you want to integrate security into your development life cycle and what measurements we took.
- Research Report: See what the TRL is, with also the conclusion on the questions we asked.

Note reading the User Stories is also important to know what is important and what is left to do.

Background

From the research report, you can maybe see that we got different requirements from the client to do something else really quick. Therefore, all the products that are handed over, are not fully finished, they miss things. The client wanted us in the middle of the semester to switch, leaving us with a not-finished tournament system, and now a half working challenging system.

Important for group

If you read the cost analysis and cloud research, you will understand the benefits of cloud computing but also how expensive it is. Be aware that the cloud is expensive and do this with care.

Communcation with the client is important for this topic so be pro active for this!

Unfinished Requirements

First, have a quick chat with the client about what they found important.

To give you important information about the architecture, will show a C2 diagram.

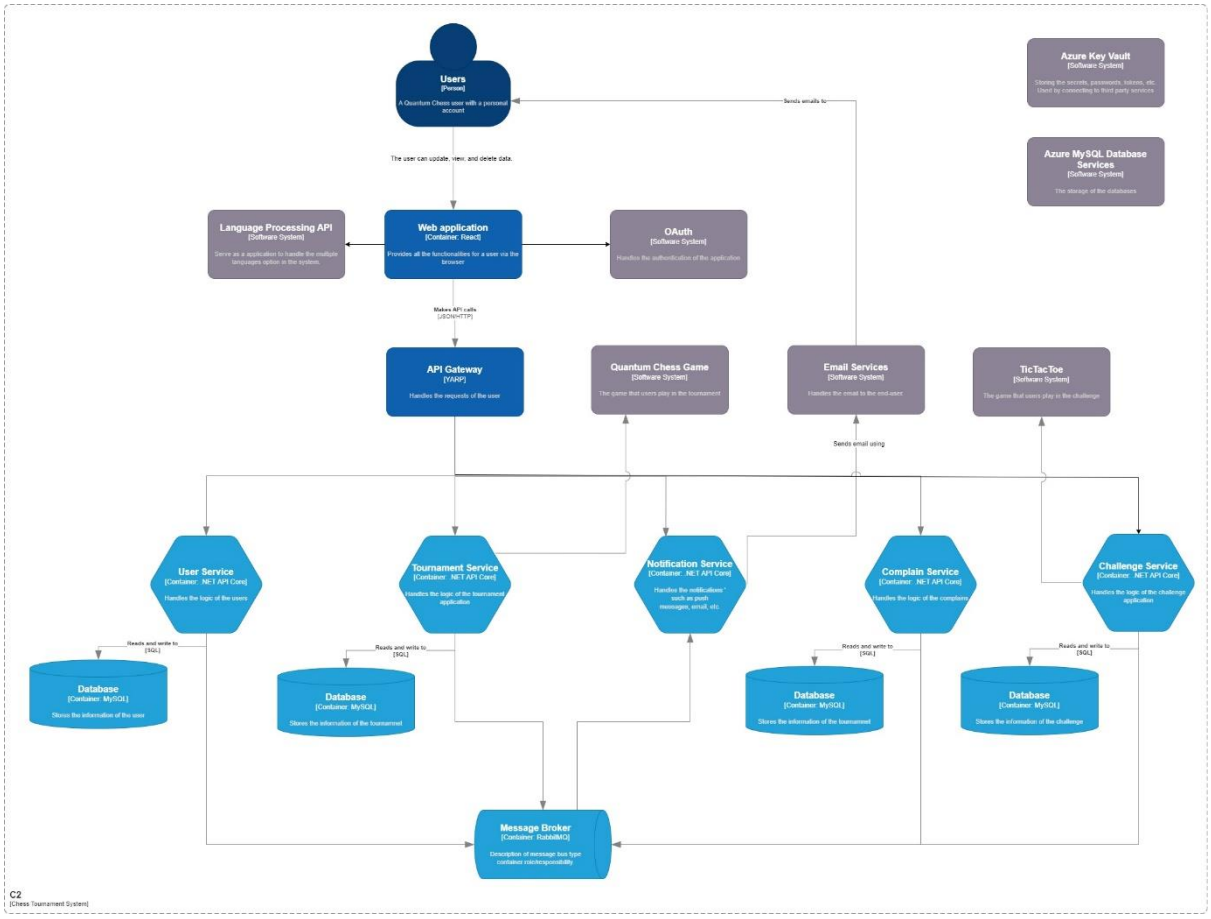

Figure 1: C2 Diagram

First give the unfinished requirements:

- Tournament: The database design is not done. A leaderboard is required, also with communication through the message broker to communicate with other services.
- Notification service: The whole service needs to be implemented, see requirements.
- Complain Service: The whole service needs to be implemented, see requirements.
- Message Broker: Currently the methods are made, for consumers you need to make a hosted service to start consuming from the background.
- OAuth: this is locally working due to local being seen as a safe connection. In deployment, we need to have a valid HTTPS certificate (lets encrypt) to make use of this service.
- Language Processing API: The application needs to be seen in multiple languages (English, Dutch).
- Challenge Service: All the requirements are implemented with the Database, make sure that API to TicTacToe will be properly working, since this is not always the case, ask the client.
- Keyvault: Will store the secrets of the application, make sure to check what is needed for this vault. Add if possible that it can be used for CI/CD.

- API Gateway: Add the important requirements described in technical design such as rate limiting, load balancing, authentication check, etc. Can modify this for specific requests as well.
- Azure SQL Database Server: This has the databases of User and Challenge with the design found in (Quantum TLC, Database Analysis). Make designs for tournaments, notifications, and complaints.

Extra considerations for architecture:

- Add Caching (Redis)
- Look into Cloud and if not to expensive, use Cloud Messagebroker and Redis caching for this.

DevOps

Currently, we have build, test (unit tests, locally E2E), security, package, and deploy stages in the CI/CD pipeline. Integration tests need to be added (look at the postman, Newman), and an analysis stage to show code coverage and quality of the code (see coding guidelines as well). Furthermore, look into E2E on deployment.

Transition Plan

Next steps:

- Read the documents
- Get an understanding of microservices.
- Outline based on the controller of the services (begin with user service, tournament, challenge, then start on new ones) what is done and check if it works. Check only if it first works on the service layer, when all are checked properly check it by calling via the API gateway. Finally, check it in the front end.
- Make a database for tournament service, and update the user service database if needed (based on the stored requirements).
- Understand Kubernetes services in Azure, how does this work (deployment, service, ingress, secrets, configmap, HPA, Cronjobs, etc).
- Make HTTPS work for deployment.
- Get an understanding of how services such as key vault, and database Azure work and how we connect them.

So it is important to first look at the documentation before even doing anything!