<University Name>
<Department Name>

Summer Practice Report
<Course Code>

Student Name:

Organization Name:

Address:

Start Date:

End Date:

Total Working Days:

Video Presentation Link (optional):

Contents

1 Introduction

This document contains information regarding my internship in Siren Bilisim on summer of 2021. The document will consist of information about the organization, project I have worked on and my role in the project. Although its development process isn't perfecly aligned with software development lifecycle, the details regarding the project, namely PGMaster, will be organized as following:

- **Analysis:** In this section, I will be explaining the effort put into understanding and determining the requirements for the software.
- **Design:** In this section, the design choices that were made in accordance with the requirements will be demonstrated.
- **Implementation:** The development of the software in somewhat intertwined with the prior phases. However, explanations regarding the implementation specifically will be included in this section.
- **Testing:** This software's implementation involves thorough testing of its different modules. In terms of the timeline, the testing is not necessarily performed *after* implementation. Information regarding testing will still be written in this section of the document for sake of sticking with the outline demanded by the department.

The internship is performed in Siren Bilisim under the supervision of Koray Kocamaz. During that time, I have been mainly involved in web development activities. Before the internship, I have had developed various applications that utilizes web technologies. During the internship, I had chance to learn more about frontend development on top of backend. The project mainly involves active participation of three developers. While I have performed my day-to-day development activities under supervision of Koray, the team member that I worked with the most was Derviş Mahmutoğlu from HAVELSAN. In that regard, I could partially claim that my internship project included the contribution of multiple organizations although HAVELSAN itself is not officially involved.

Although my involvement in PGMaster was not limited to the frontend, majority of the source code I have written would allow users interact with the underlying system. Due to that, I will be sharing extensive demonstration of the user interfaces.

On top of web development, the project revolves around databases, containers, and clusters. Domain knowledge about these entities have been also learned during this internship. I will be mentioning them in detail within relevant sections of the report.

2 Information about Project

In this chapter, I will be explaining technical details regarding the project. To provide better context, I shall be introducing the project first.

PGMaster is an administration tool that mainly intends to help database and system administrators. PGMaster provides an interface for the systems that utilize containers which are homes to databases. Using Docker containers, our system creates and manages clusters. These clusters offer capabilities like high availability and data integrity. With various configurations, databases can be set up with modes like master, standby and more. Intended users of the product is admins that manage systems with high amounts of transactions. Whether it is private or official organization, many admins perform high level database management operations using tools on command-line interface (CLI). PGMaster can be considered as an abstraction over such interfaces as it allows critical operations to be initiated, observed and completed using a graphical user interface (GUI).

PGMaster, although trivializing many tasks for admins, provides a CLI over the web application as well. This is achieved using WebSocket as well as various libraries which will be explained further in detail later on.

Without any due, I will get into the phases I have been through during my internship.

2.1 Analysis Phase

On my first day of the internship, after being given a computer and setting up my development environment, I have been introduced to Derviş. Prior to my involvement, he was the one and only contributor of the project. It was determined that I was going to implement frontend application of the system. After discussing the high level requirements of the system, it was clear to be that library that I was going to use for the project, namely Reactjs, was suitable. The project didn't have any document prior to implementation.

Before my involvement, Derviş had already implemented some of the backend application modules which was using Java and Spring Boot framework. Being an experienced software developer (with 20+ years of experience), he was quite comfortable implementing Java and bash code. He also had extensive knowledge on PostgreSQL and database management systems in general. However, he needed a hand with the frontend application. There was only a JavaServer Pages (JSP) implementation with a form and some buttons written in HTML which were not sufficient for more sophisticated and dynamic nature of the system.

We have discussed the project briefly and I proceeded to learn Reactjs the following days. I will be telling about implementation related details that were discussed on

initial meetings on following sections. After understanding the general requirements of the system I have spent time analyzing the following:

- Candidate modules and entities.
- Nature of the data to be processed.
- Design related constraints.
- Documentation needs.
- Conventions to be followed during implementation.

I will try to explain considerations I have had for each of these items.

2.1.1 Modules and Entities

Before jumping into these items, it seems necessary to talk about PGMaster modules and entities. The modules can be listed as follows:

- **PGManager:** Even if the name suggests something similar to PGAdmin, a web-based GUI tool that is commonly used to interact with PostgreSQL databases, it is actually the backend of PGMaster application. It is written in Java (using Spring Boot framework) and it has two main purpose: Performing CRUD operations on pgm database, the database that contains "meta" data regarding the application itself and manage operations that affect PGMaster entities. These entities will be explained below.
- **Grafana:** Grafana is a multi-platform open source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to data sources. Our web application's frontend is intended to be embedded into one of Grafana's custom panels. Within the context of PGMaster, this is its most relevant purpose. Other than that, various services and endpoints that works on the container of PGMaster provide the data Grafana shows. As understood, the application PGMaster is also containerized. The content of this container will also be mentioned later on the document.
- **WeTTY:** WeTTy (Web + TTY) is an npm module that makes it possible to have terminal access in browser over HTTP(S). As mentioned before, PGManager is intended to offer necessary tools that would help advanced users to perform various actions on CLI. We have decided that using a library for this purpose would be beneficial over implementing our own module. It basically utilizes WebSocket protocol to establish an SSH connection. In PGManager, we intend to offer access to PG hosts and containers using this library.
- **PG-Web:** This is the application I have implemented from the beginning using Typescript and Reactjs. It basically provides a GUI to PGManager users where

they can perform various actions such as creating and initializing PGContainers, taking backups and restoring them, monitoring clusters and their nodes' status etc. PGMaster has over 100 such operations. While most of them needs only one parameter to be provided, some of them require more sophisticated data to be provided. For sake of simplicity, only some portions of these operations will be explained on the following sections of the document.

These four modules are the main "pillar stones" of a PGMaster ecosystem. Organizations that have systems with multiple databases which needs to deal with high volumes of transactions while maintaining high availability and also security and integrity of their data may unfortunately not have admins that are capable of performing operations that were mentioned above trivially. These modules intend to offer tools that are necessary to maintain, operate and administrate these systems whether they have advanced database administration skills or not.

I have mentioned that PGMaster have different entities. I will be listing what they are and what purpose do they serve below:

- **PGContainer:** As mentioned briefly before, databases within PGMaster ecosystem lives inside Docker containers. These databases have different configurations depending on their cluster modes. These modes can be simple listed as (I am not creating another list!) master, standby and replica. Intuitively, these names indicate whether a container contains the "main" database or its "copies". Depending on the status of a master database (and its container), various operations can be triggered using PG-Web. These containers come together and create a cluster, more specifically PGCluster. It is worth mentioning that PGMaster itself is a container as well. This container is intended to contain modules mentioned above. While the main benefit of using a container is to be able to deploy PGMaster on an organization easily, other benefits of using a container could also be considered as the added advantages.

- **PGHost:** Naturally, a host is required to run Docker engine and PGHost is the host that houses these containers. While PGHost mainly indicate a dedicated, physical server, it can also be a virtual private server (VPS). These hosts houses PGContainer and their availability directly impacts health of a PGCluster. Naturally, a PGContainer may have nodes that lives inside different PGHosts. I think for the scope of this document, this should suffice explaining what a PGHost is.

- **PGCluster:** PGCluster, to be brief, is the combination of multiple PGContainers. Its nodes (PGContainers) may or may not live inside different PGHosts. Containers alone do not provide hi(gh availability and clusters are utilized to provide it. PGClusters are shown with a topology inside PG-Web. Entity management view, which will be explained later on, allows users to configure their nodes inside a PGCluster.

- **PGBackupServer:** As understood from the name, it is simply a server that contains backup files of a PGContainer database. Backup data is transferred to

this entity on desired intervals and when the need rises, this entity provides the data that needs to be recovered.

These four entities were the main entities that were necessary to have a functional PGMaster ecosystem. I have spent first two weeks creating views to create, update and delete show these entities. The main benefit of using Reactjs over regular HTML/CSS/JS stack for the GUI was to be able to render these entities optimally and dynamically without having to update the page. While Javascipt makes a regular web page dynamic to some extend, Reactjs allows users to manipulate data and have Document Object Model (DOM) to be synchronized seamingly. As stated on the official documents of Reactjs, Web Components and Reactjs are complementary, i.e., they solve different problems. While Web Components provide encapsulation of reusable components, Reactjs provides declarative implementation. So it is safe to say that real benefit of using Reactjs is felt by developers, rather than users. I don't mean to go out of scope of this document and explain benefits of React, so I will keep it brief. In PGMaster context, React allows us to utilize having declarative implementation, meaning the developer can set the "rules" for a view and the view updates automatically with the rules provided by the developer without needing to refresh the page user interacts. I have already mentioned that PG-Web, the module I have implemented is embedded into Grafana. It goes without saying that user should be able to observe the changes on PGMaster GUI without needing to refresh the web page, especially considering that Grafana already allows this for other views than PGMaster. In that sense, usage of a library such as Reactjs was vital during the implementation.

2.1.2 Data

The data that circulates on PGMaster could be divided into two:

- **Entity Data:** PGMaster has its own database to contain details regarding entities. While adding/updating/deleting these entities, a PostgreSQL database (pgm) is used. PGManager (the back-end) has numerous endpoints (listHosts, listContainers etc.) that feed the front-end application PG-Web. Entity data is fetched to render initial view of PG-Web and it is updated performing various CRUD operations on the said database. When a user updates entities, the change is immediately reflected on PG-Web. This ensures users to have the latest updates on PGMaster to be synchronized on the GUI they are using. I have implemented the GUI in a way that a user can have sufficient feedback and information whether operations they have initiated were successful or not.

- **Operation Results:** While most of the configuration a user can make is intended to be performed once or twice, the operations on a PGMaster entity is expected to be more repetitive. Depending on the status of a PGMaster entity, these operations may fail or succeed. I have implemented a view where users may see the results of the operations they have initiated. We have realized that HTTP status codes wouldn't suffice since they don't provide information regarding what went during

the execution of a PGMaster operation. Then Derviş decided that we can use slightly more sophisticated response objects, namely Rx, to provide user more than just a status code. These objects contain fields like error code, command, result text, hint, execution time, container name, host name, cluster name, path, details and more. I have implemented various views that allow users to view these. I'd like to mention a detail regarding the implementation of these views. When a user triggers a command to execute, this command have potential to trigger other commands. These commands may or may not fail depending on a lot of factors (such as permissions, status of the target entity etc.) and user should be able to identify when did an operation has failed with precision. I have offered two different alternatives for users. Firstly, users may see these commands and their results with indentation on a table. As an alternative to this, users may also see a command and its "child" commands in a tree-like view. Upon discussions with Derviş, we have decided to use the first, as it would be practially impossible to show the results of these commands' results on a tree view, considering Grafana can only provide so much space.

2.1.3 Design Constraints

As mentioned before, PG-Web is intended to be shown among other views Grafana has. I must admit that I have faced the challenge of "unknown requirements". To elaborate, PGMaster is intended to be used by "power users" that have access to hardware such as 4K monitors. However, Derviş occasionally has tested the views on his personal mobile device and this forced me to consider another constraint which was being responsive. Fortunately, the library I have used (Semantic UI React) provided predefined solutions which allowed users to interact PG-Web seamningly. Prior to my internship at Siren Bilisim, I have considered web development rather trivial. However, I have learned that implementing GUIs that just work on different devices is indeed a challenge where you have to consider countless edge cases. I am not sure how viable the end product will be on different mobile devices (tablets and movile phones running Android or IOS) but I have spent decent amount of time on making our Uproduct work on different devices. Unfortunately, Siren Bilişim didn't have any UI/UX designer which was assigned to work on this project, so I had to come up with my own designs and implementations.

2.1.4 Documentation

Having had an introduction to software requirements specification (SRS) and software development lifecycle in general in our Software Engineering course, I usually try and understand the needs of any development activity prior to actual work. Aiming to follow the same trend here, I must admit that I was mildly disappointed, since although we didn't have a strict timeline to deliver the proof of concept (PoC) version of the application, the project was solely maintained and developed by one person. This should have -expectedly- obscured the need for any form of documentation.

I have actively communicated with my supervisor (Koray) and colleague (Derviş) on

the subject and I was given permission (and responsibility) to write SRS for the project. I did not start writing the document right away, because it took me a while to fully grasp the overall needs and constraints of the system and the rest ofthe requirements naturally needs further discussion with stakeholders and the potential users. I still had chance to write some of the document by myself and it will be attached to the report. Final version of the SRS is expected to be completed after it is presented to The Ministry of Health.

2.1.5 Conventions

Reactjs, the library that I have used extensively during the implementation of PG-Web is a libarary that was open-sourced in 2013. Since that time, with the help of community (that consists of developers from all around the world) Facebook switched from class-based components (and its implemenation) to function-based components. For sake of not getting into too much details, I will skip how it happened, but I adopted the function-based component implementation approach. With the additions of "hooks", Reactjs eliminated the need for most of the boilerplate code and I made use of this approach.

2.2 Design Phase

2.3 Implementation Phase

2.4 Testing Phase

3 Organization

3.1 Organization and Structure

3.2 Methodologies and Strategies Used in the Company

4 Conclusion

4.1 Appendices