ToDo - System for managing user's To Do list

Bernardo Rodrigues

Supervisor: Nuno Leite

Report written for Project and Seminary
from the Bachelor's Degree in Computer Science and Computer Engineering

September, 2021

Contents

List of Acronyms

API Application Programming Interface.

DAL Data Access Layer.

JWT Json Web Token.

RDMBS Relational Database Management System.

REST Representational State Transfer.

SQL Structured Query Language.

UI User Interface.

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objectives

In today's fast and complex world people try to schedule days or weeks. People create small reminders, either electronically or physically, to do something, whatever that may be. Those little notes we give to ourselves, those "to-do's" are often forgotten and not fully fulfilled. There exists software applications in the market that partially accomplishes this goal. They will often remind the user of the task that needs to be done only using the time frame the user gave. This creates a problem where the individual makes a reminder, does not do it and the application will not keep notifying the user.
As such the proposed solution will attempt to mend that by creating an environment where a reminder is easily created but hardly forgotten or skipped over. The solution consists of a web application divided into three different components, a front-end web app, a back-end web API and a database. This application will allow the users to create an account and then make reminders. The users will be notified by the application until they mark said reminder as done.
The notification is a necessity to ensure the user fulfills the reminder. Since the user will be notified until the reminder is marked as "done" the person will be reminded every day until the completion of the task.

For this purpose a set of mandatory requirements was outlined:

- Account creation, login, logout and account deletion;
- Reminder creation, update and deletion;
- Notifications;

Some optional requirements were also outline to enrich the application:

- Translations
- Priority notifications

This report will outline the process of the application development, the choices made and the work that remains to be done.

1.2 State of the art

Through a brief research a multitude of application can be found to create reminders, although with different end goals. A few examples of some of these are *Todoist [1]*, Google Calendar's reminder system [2], *Notion [3]* and *Evernote [4]*.

Google Calendar allows the user to create events in the calendar and reminds the user a set time before said event. Even though the end goal is different it still functions as a reminder system of sorts.

Notion allows the user to create lists of reminders and has different visualization tools to accommodate the user's needs.

Evernote is one of the most complete of all the mentioned applications. It allows the user to create not only reminders but store different types of information for future use, such as hyperlinks, images, etc. It has templates for different types of reminders and it notifies the user on the due time.

Todoist is the application that most resembles the end goal established with this project. It allows the user to create tasks, sub-tasks and even recurring tasks. The application also notifies the user on due time.

These applications all allow the user to create reminders, some with more extra features, and remind the user on their due time.

Since all these solutions have a mobile applications (except for Notion), they make the notifications more effective, although they stop notifying the user when the reminder's due date is due.

Chapter 2

Proposed Solution

This chapter contains the details of the implementation of the solution as well as a description of the components involved in the application. The high level architecture of the solution was designed with the separation of responsibilities in mind. It's divided in three main components, the front-end application, the server web API and the database.

The database was needed to store the user data and reminders. Since the data would be structured it was decided to use a relational database. There were two databases considered for this: PostgreSQL and MySQL. Although MySQL has faster read speeds [5] which would advantageous for the task service, PostgreSQL was chosen since it was already known and easier to integrate with NodeJS.

2.1 database

2.2 title

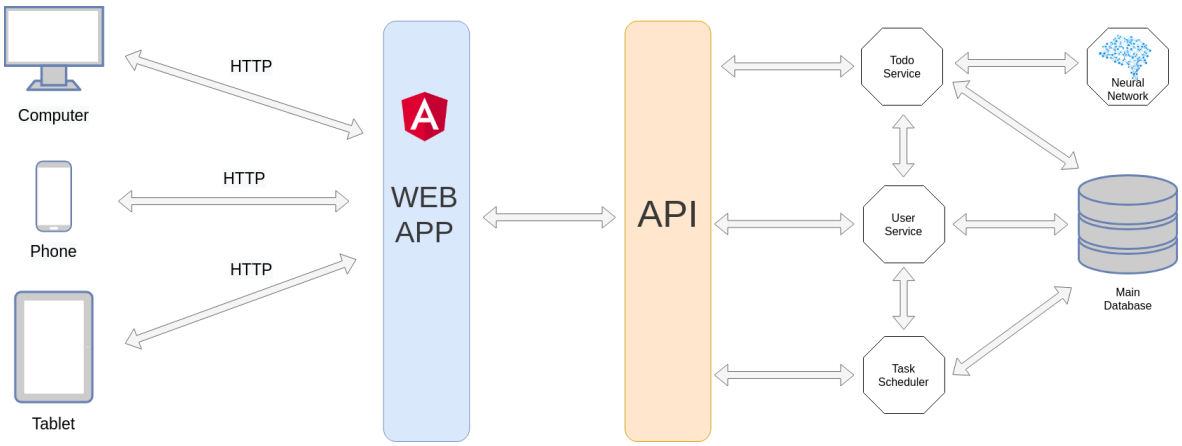

Figure 2.1: High level architecture

The architecture for the web API was thoroughly researched and studied. At first this project was gonna use a monolithic architecture [6] where the whole API would be implement on a single server as seen in the image below.

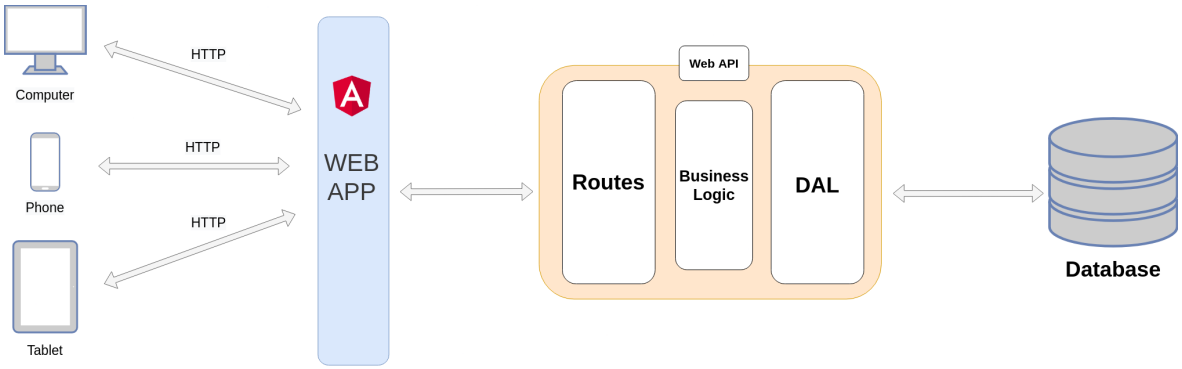

Figure 2.2: Abstract monolithic architecture

This was later changed due to the fact that it wasn't as easily scalable [7]. After this it was decided that the API would be implemented using a micro-service[8] architecture. This allows for a more select scalability. When needed, more instances of the same services can be launched and improve overall performance.

There are three different services:

1. User
2. To-do
3. Task

There was also the idea of combining both the monolithic and micro-service architecture in a mixture of both patterns. This idea was later scrapped and the micro-service system was then chosen.

The web API follows a Representational State Transfer (REST) pattern[9]. Each of these services are independent from one another and self sufficient, each with it's own responsibilities RESTful web API

2.3 User

The user micro-service will have the operations needed for an user to utilize the application. These operations consist of the sign up, login, logout and, delete. The micro-service will use Json Web Token (JWT) [10] as a way to make sure the users only access their information. The service is split into two different areas, the repository[11] that accesses the database and, the routing.

2.4 To-do

The to-do micro-service handles all the operations necessary to manage the to-do's. These operations embedded in this service are to-do creation, deletion and update. This micro-service will interact with the user service to validate and extract the JWT information.

2.5 Task

The task micro-service has three main objectives. It offers a way to subscribe and unsubscribe users to receive notifications as well as checking the database periodically to ensure the notifications are sent to the subscribed users. The notifications will be sent according to the push protocol[12].

The front-end application was designed using Angular 12[13] since it allows for a fast development as it has a robust set of tooling and components. The User Interface (UI) was developed so that the user experience will be smooth and enjoyable.
Since the data structure is fixed a Structured Query Language (SQL) database would be the preferable choice. As such PostgreSQL[14] was the chosen Relational Database Management System (RDMBS) since it's reliable and flexible.

Chapter 3

Database and Data Modeling

The database model is described, in this chapter. It's a fairly simple model although additions can be made to augment and improve the application's features.

Figure 3.1: Data Model

A user can have multiple to-do's and subscriptions (tasks). The need for multiple subscriptions stems from the fact that the user can have multiple instances of the application running in different desktops.

There also is a priority system which hasn't yet been implemented. In the future each to-do will have a priority with a specific value and this value will be used to calculate how many times the user will be notified.

Chapter 4

Web API

The REST web API is composed of three micro-services, an API gateway and a neural network[15].
The micro-services all have different responsibilities but they mostly follow the same pattern. Each service is divided into two main layers, the routing and the Data Access Layer (DAL) which is comprised of a repository. The whole API was built utilizing NodeJS[16] and Typescript[17]. All services use *node-postgres*[18] to interact with the database. The gateway and some services utilize the *axios*[19] package to send HTTP requests.

4.1 API Gateway

The API gateway's purpose is to receive incoming requests and redirect them to the intended services. To make it follow the micro-service pattern it requires it to be stateless. The gateway exposes a REST API to be used by the front-end applications.

4.2 User Service

The user service is responsible for all the operations regarding user accounts as well as verifying each request. The service exposes a route for other services to verify the JWT authenticity. This route is only known by the other services and not available to the gateway.

4.3 To-do Service

The to-do service serves the purpose of creating, updating and canceling to-do's. The service exposes a route for the task service so it can get all undone reminders. It also utilizes the verification route exposed by the user service as previously mentioned.

4.4 Task Service

The task service periodically checks for undone to-do's. It does by sending an HTTP request to the to-do service to get all undone reminders. It also utilizes the user service to

verify the authenticity of the requests. When the server is starting it schedules an event using the Node Schedule package[20] to periodically make these requests. As of right now it only checks every day at a specific time but eventually it will be able to receive a granularity value and be dynamic. This service also is the one that sends the notifications to the users. It does this by using the Web Push package[21]. If there is a failure when sending the notification it will erase the endpoint stored in the database as it probably means the user left the session or the endpoint is no longer available. The task service utilizes VAPID[22] keys. These keys are essential to utilize the Web Push Protocol.

Chapter 5

Conclusions

Creating a reminder system with specific granularity and prioritization has it's difficulties. Even though the API was implement and is functioning the challenges of the development were more than previously analyzed. The implementation of the notification system was arduous and the micro-service architecture, even though useful and extremely scalable, proved to be challenging. The application itself has a lot of potential future work since it can be improved and is more of a proof of concept.

The front-end application can be improved but as previously mentioned it was a proof of concept. Future improvements to the application would make it more user friendly and usable.

References

[1] Doist Inc. Todoist. https://todoist.com/ [Online accessed; 14-June-2021].

[2] Google Inc. Google Calendar. https://calendar.google.com/ [Online accessed; 14-June-2021].

[3] Notion Labs, Inc. Notion. https://www.notion.so/ [Online accessed; 13-September-2021].

[4] Evernote Corporation. Evernote. https://evernote.com/ [Online accessed; 13-September-2021].

[5] Mark Drake. Sqlite vs mysql vs postgresql: A comparison of relational database management systems. 2014. https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems.

[6] Richardson, Chris. Pattern: Monolithic Architecture. Technical report, 2019. Pattern: MonolithicArchitecture [Online accessed; 14-June-2021].

[7] André Bondi. Characteristics of Scalability and Their Impact on Performance. In *Proceedings Second International Workshop on Software and Performance WOSP 2000*, pages 195–203, 01 2000.

[8] Richardson, Chris. Pattern: Microservice Architecture. Technical report, 2019. https://microservices.io/patterns/microservices.html [Online accessed; 14-June-2021].

[9] Fielding, Roy. *Architectural Styles andthe Design of Network-based Software Architectures*. software, University of California, Irvine, 2000. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

[10] M. Jones, J. Bradley, and N. Sakimura. JSON Web Token (JWT). RFC 7519, RFC Editor, May 2015. http://www.rfc-editor.org/rfc/rfc7519.txt.

[11] Mary Shaw David Garlan. An Introduction to Software Architecture. *School of Computer Science Carnegie Mellon University*, pages 12–13, 1994.

[12] M. Thomson, E. Damaggio, and B. Raymor. Generic Event Delivery Using HTTP Push. RFC 8030, RFC Editor, December 2016. http://www.rfc-editor.org/rfc/rfc8030.txt.

[13] Google Inc. Angular, 2021. https://angular.io/ [Online accessed; 14-June-2021].

[14] The PostgreSQL Global Development Group. PostgreSQL, 2021. https://www.postgresql.org/ [Online accessed; 14-June-2021].

[15] Gurney, Kevin. *An introduction to neural networks*. UCL Press Limited, 1997.

[16] OpenJS Foundation. NodeJS. https://nodejs.org/ [Online accessed; 14-June-2021].

[17] Microsoft Inc. Typescript. https://www.typescriptlang.org/ [Online accessed; 14-June-2021].

[18] Carlson, Brian. node-postgres. https://node-postgres.com/ [Online access; 14-June-2021].

[19] The Axios Project. Axios. https://github.com/axios/axios [Online accessed; 14-June-2021].

[20] Patenaude, Matt. Node Schedule. https://github.com/node-schedule/node-schedule [Online accessed; 14-June-2021].

[21] Castelluccio, Marco. Web Push. https://github.com/web-push-libs/web-push [Online accessed; 14-June-2021].

[22] Martin Thomson and Peter Beverloo. Voluntary Application Server Identification for Web Push. Internet-Draft draft-thomson-webpush-vapid-02, Internet Engineering Task Force, January 2016. Work in Progress.