E-CARE

Conceptual Architecture Report

Esra Kastrati - 215507205

April 21st, 2021

Abstract

This report examines the conceptual architecture of the E-care native Application. It starts by examining the problem domain and detailing the way it aims to solve it. The report details the users of the software system, and from that extracts all of the requirements. Those requirements are then used to decide on a software architecture for the system. After examining different options, we established that serverless architecture is the best architecture to allow for concurrent and persistent access to the database. A **serverless architecture** is a way to build and run applications and services without having to manage infrastructure. Our application will still run on servers, but all the server management is done by AWS. Using this information, we draw out the use case diagrams for our main users.

Introduction

This project was proposed by Dr. Susan Murtha and Professor Robert Allison. Dr. Murtha, had a personal experience where her mother-in-law had to be put in a long-term care home as a result of breaking both hips. Unfortunately, Dr. Murtha's mother-in-law also had dementia and had trouble remembering where she was or why she was put in a care home, which made this experience incredibly stressful for her and the rest of her family. We see this as an opportunity to improve the mental and physical wellness of dementia patients in these homes. As well as to support caregivers and families who often bear the brunt of this disease. We want to do this by using technology as a means to improve the communication between dementia patients in long-term care homes, their caregivers, and families, which would improve the quality of life of everyone involved as well as prevent family and caregiver burnout. By facilitating easy communication between all members in the circle of care, this will improve the caregiving that dementia residents living in long-term care homes receive, while also addressing the needs of the individuals that care for them. This report discusses the abstract architecture of our software system, it details the main subsystems and components of the system and how

they interact. Based on that description, it outlines the chosen architecture of the system. Initially, we will give an overview of the system's users as well as the requirements.

Requirements

Seeing as the aim of our application is to help connect patients, caregivers, and families through the creation of a virtual circle of care, we can outline three main users:

Patient: Person using the website for a better shopping experience

Family Member: Person using the application to have a better communication with the circle of care

Caregiver: Person using the application to provide better care for the patients

Functional Requirements

REQxF-1		user can login to the application
REQxF-2		user can view information of a particular Contact
REQxF-3		user can browse through the messages recently sent
REQxF-4		user can view the profile of the patient
REQxF-5		user can message and contact other users on the circle of care
REQxF-6		user can browse the contact list
REQxF-7		user can add/remove users from the Contact List
REQxF-8		user can view activities that are currently on the Calendar
REQxF-9		user can add/remove activities from the Calendar
REQxF-10		user can upload short stories on the system
REQxF-11		user can view pictures and videos uploaded on the application
REQxF-12		user can listen a song from the application
REQxF-13		caregiver can add/remove a patient from the patient list
REQxF-14		caregiver can browse the patient list
REQxF-15		caregiver can update hobbies of the patient
REQxF-16		family member can update facts of the patient
REQxF-17		family member can update health conditions of the patient

Non-Functional Requirements

Maintainability:

REQxUF-1: the system will be developed in a way to make adding functionalities easier

Security:

REQxUF-2: Users cannot make changes to database.
REQXUF-3: Users cannot access other user's information

Usability:

REQxUF-4: the system should be is simple and intuitive to use

Portability:

REQxUF-5: the system should run on OS versions: Android 5 and **iOS** 10.

Availability:

REQxUF-6: The system must run 24x7x365, with overall availability of 0.99.

Reliability:

REQxUF-7: Data returned should always be consistent with database.

Architecture

In our busy and fast-paced modern society, mobile application users demand quick access to information and data as soon as it happens —whenever there's any update or change, no matter where they are. They also want data to show up automatically on their screens as soon as there's a request sent by another user or by the backend itself, with no need to reload or refresh their client app. Taking into consideration all the above, non-functional and functional requirements, plus our mission to create seamless, effortless, and accessible communication between families,

caregivers and dementia residents in long-term care homes with the aid of technology we decided that severless architecture would best suite all our needs.

We first explain why serverless architecture (also known as serverless computing or FaaS — function-as-a-service) was our choice and the advantages this design gives our application. And then continue to talk about Amazon Cognito User Pools, GraphQL and how those fit within our architecture.

Serverless Architecture

1. There is no need to provision or maintain any servers. There is no software or runtime to install, maintain, or administer.
2. Our application can be scaled automatically or by adjusting the capacity through toggling the units of consumption rather than units of individual servers.
3. We do not have to pay for idle capacity. There is no need to pre- or over-provision capacity for things like compute and storage.

4. Serverless applications have built-in availability and fault tolerance. We do not need to architect for these capabilities since the services running the application provide them by default.

After deciding which architecture we would be using, we needed to find out where to store the data of our application. Our choice was DynomoDB which is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multi-region, multi-active, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. The data stored in this database are resources that GraphQL APIs can now interact with.

Amazon Cognito User Pools:

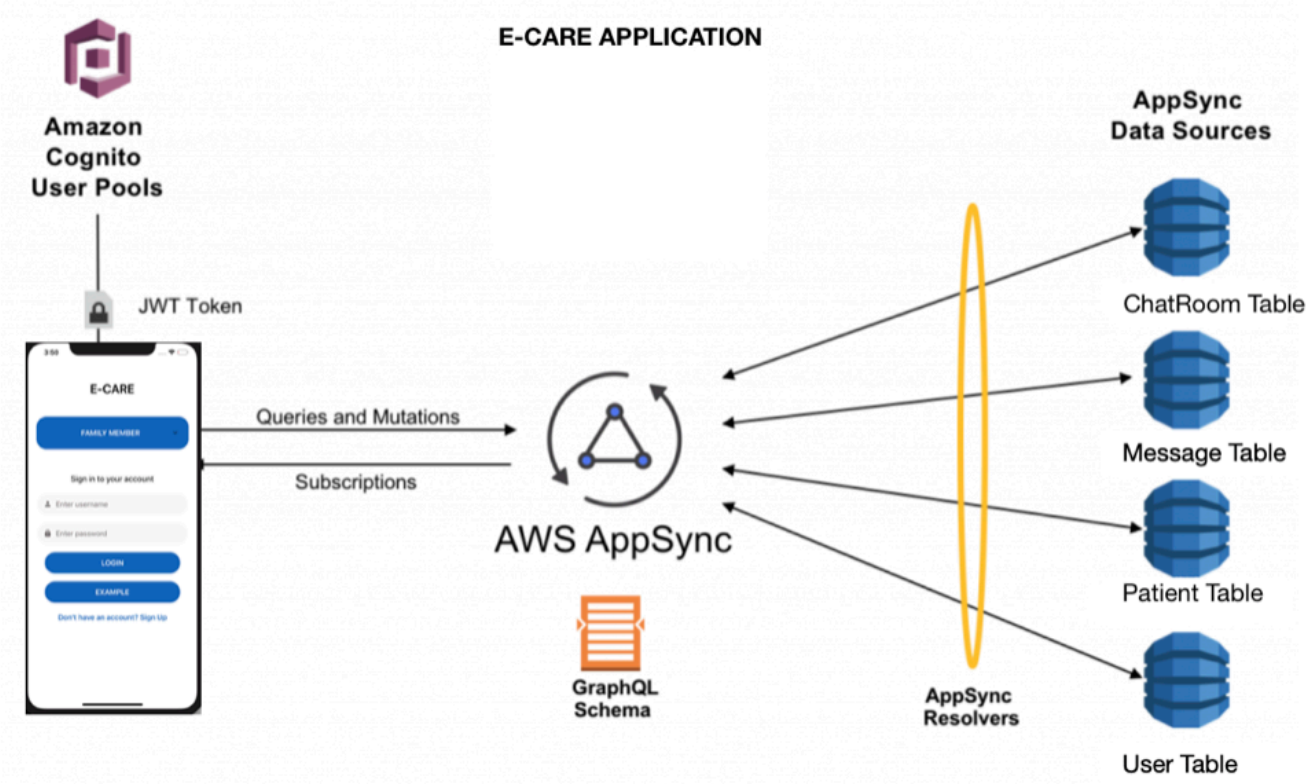
The E-Care application uses Amazon Cognito user pools, and allows users to register their account and sign in. After successful authentication, Amazon Cognito returns a JWT token to the application that we use to identify the user and authorize access to the GraphQL API.

GraphQL

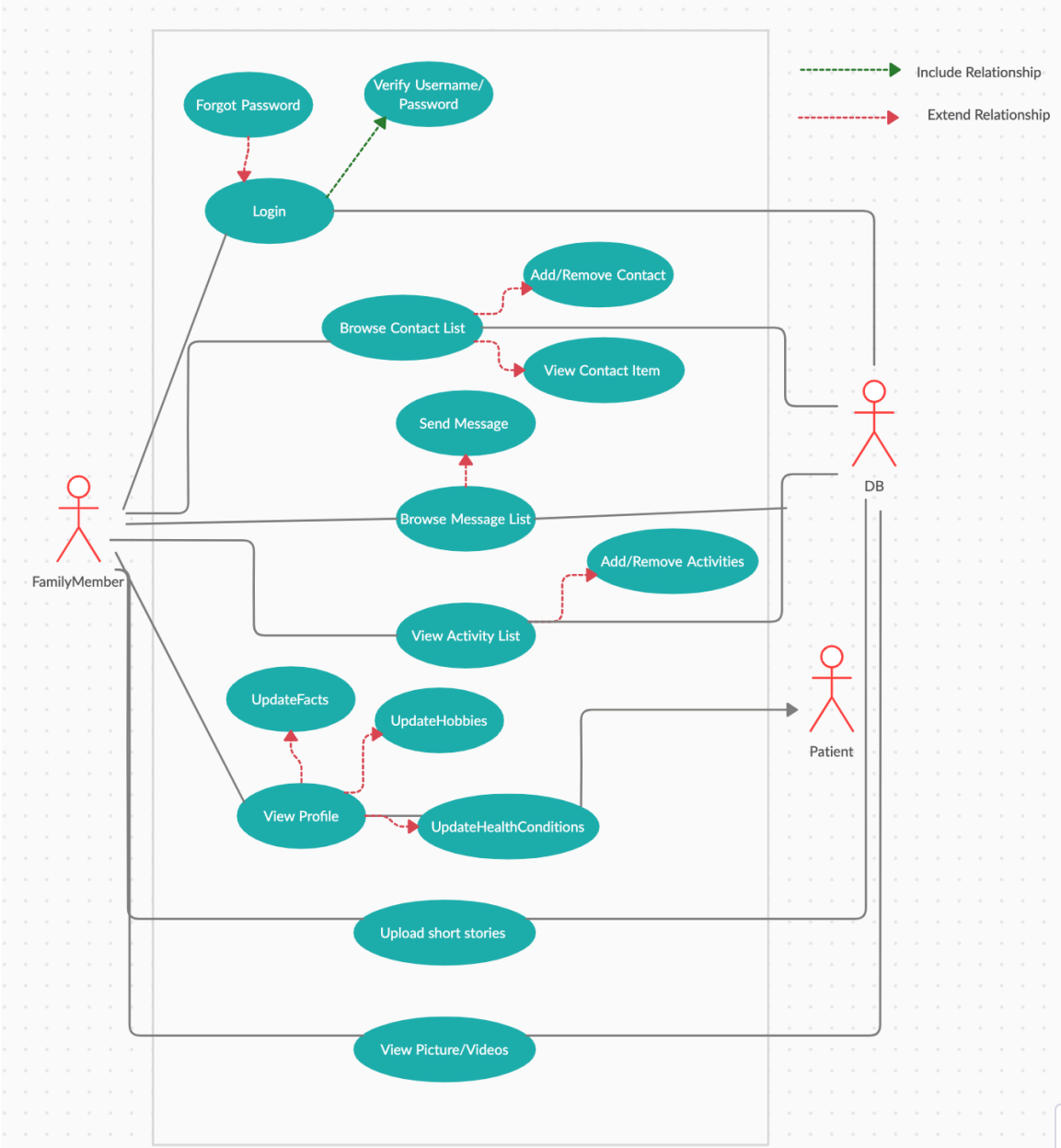
GraphQL lets us shrink our multitude of APIs down into a single HTTP endpoint, which we can use to fetch data from multiple data sources (DynomoDB). Some of the advantages of GraphQL are

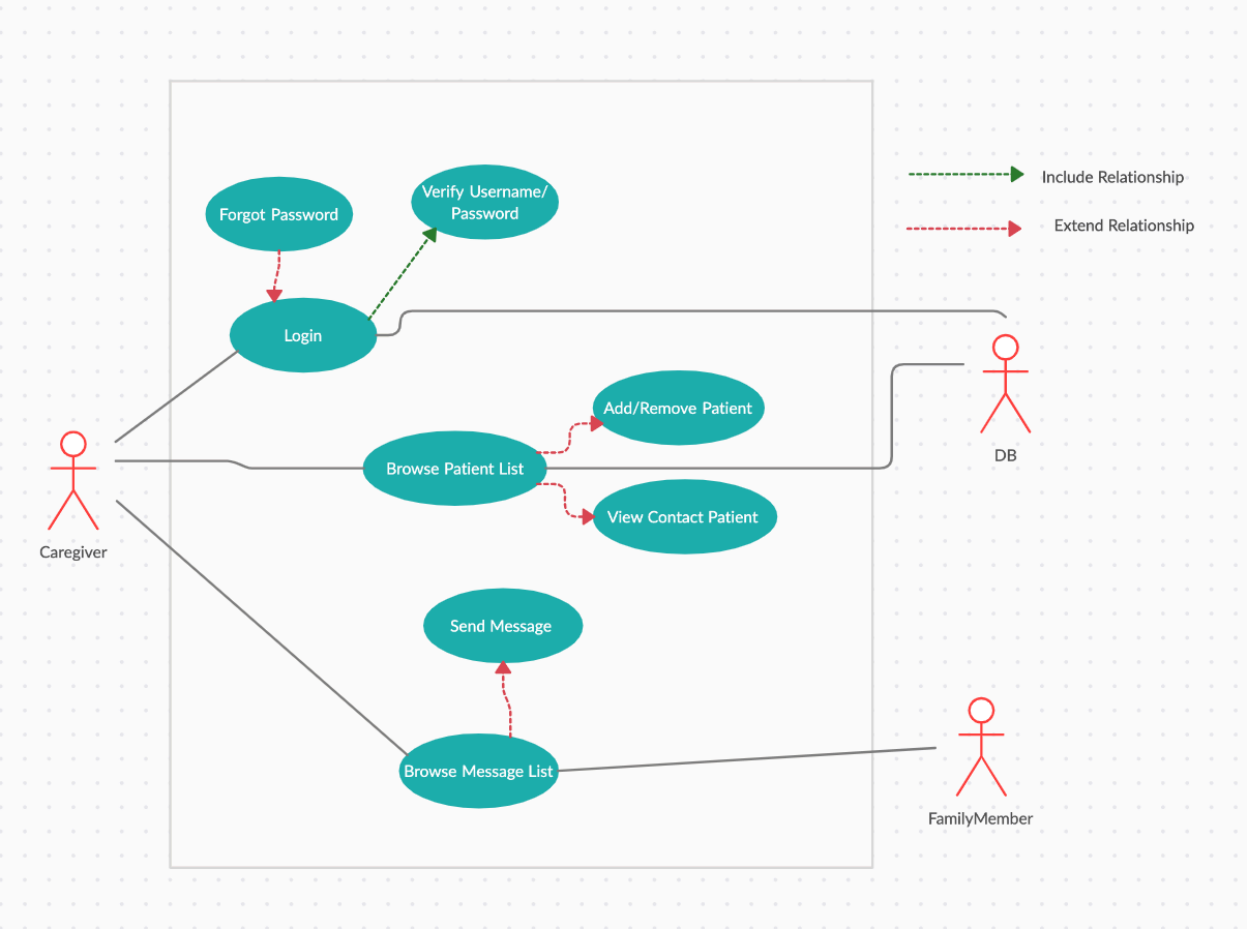
1. Reduce network costs and get better query efficiency.
2. Know exactly what the response will look like and ensure we are never sending more or less than the client needs.
3. Describe our API with types that map our schema to existing backends.

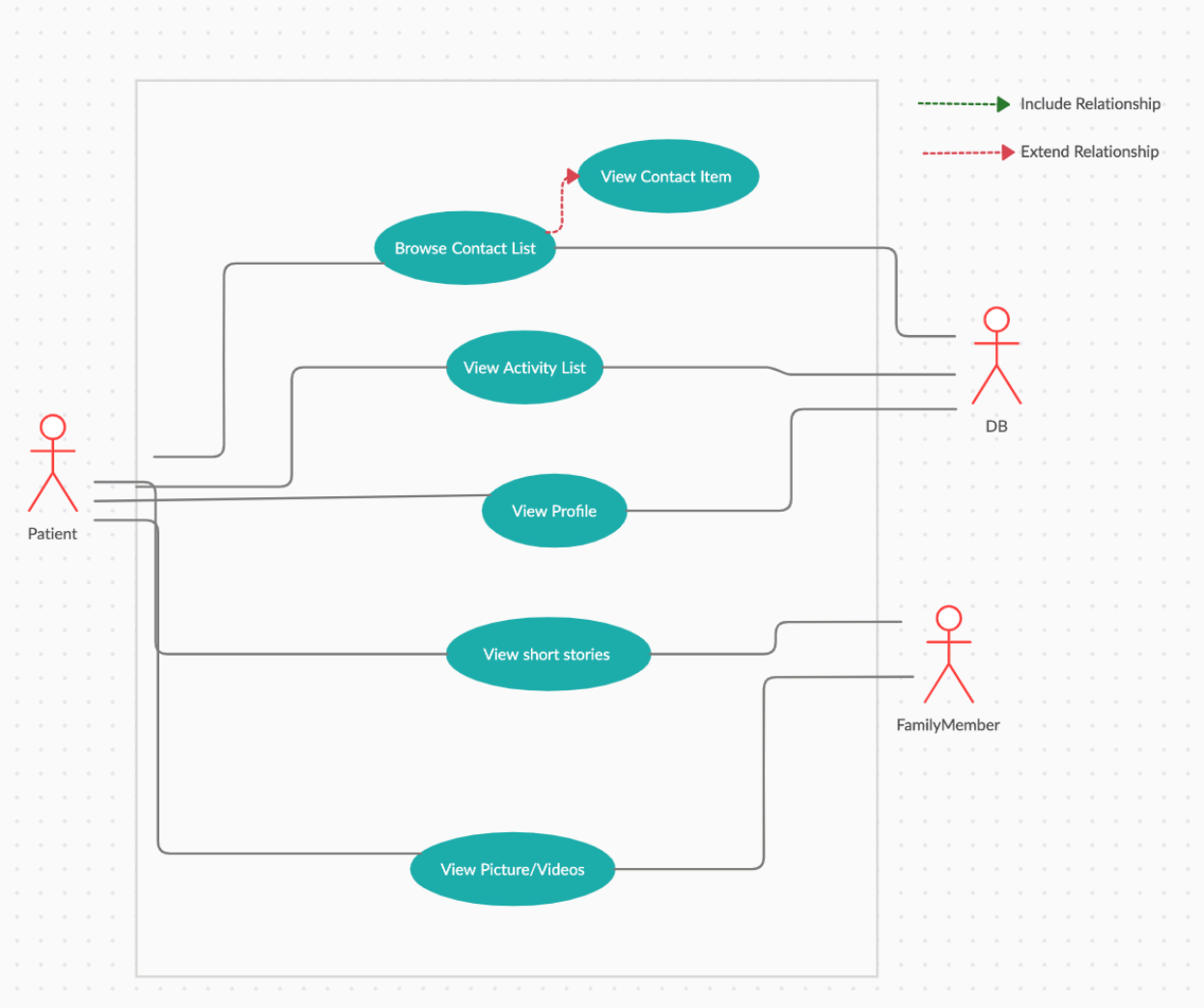
Below is a component diagram showing the high-level architecture of our system:



Use cases


Include Relationship
Extend Relationship
Forgot Password
Verify Username/ Password
Login
Add/Remove Contact
Browse Contact List
View Contact Item
Send Message
Browse Message List
Add/Remove Activities
View Activity List
UpdateFacts
UpdateHobbies
View Profile
UpdateHealthConditions
Upload short stories
View Picture/Videos
FamilyMember
DB
Patient


Include Relationship
Extend Relationship
Forgot Password
Verify Username/ Password
Login
Add/Remove Patient
Browse Patient List
View Contact Patient
Send Message
Browse Message List
Caregiver
DB
FamilyMember



Conclusion

As software systems get larger and larger, the need for established and tested software architectures increases as well. While we were conceptualizing our software, we had to think again and again about what architecture styles are available to us, and compare and contrast between them. Initially, we determined the users of our software, and from that elicited all the requirements. That allowed us to start examining the needs of the users and what software architectures would be best suited to fill them. Since the paramount function of the system is to be available, we decided on a serverless architecture. This development activity really outlined the importance of following the design method, for instance defining the users at the beginning is what allowed us to come up with comprehensive requirements, in turn, those requirements were critical for determining what software architectures would be viable in our system.