DeepSink Partners

Give A Lot

Kids Next Door

Architectural requirements document

Team Name: KidsNextDoor

Team Members: ** - indicates team leader*

Name	Surname	Student Number
Kgomotso	Moroke	u19104546*
Nqobani	Nhlengethwa	u17297542
Tshilidzi	Nekhavhambe	u17090939
Joshua	Warneke	u18227369
Rashiidah Weira	Wanda	u17281017

Contents

1 Architectural design strategy

The chosen strategy is the decomposition strategy. We have divided our system up into system components by identifying the different subsystems. Each of these subsystems will require different functionalities which can be provided by different services. By performing system decomposition we have create multiple modular subsystems.

These modular subsystems are then refined into micro services by performing functional decomposition, hence why micro services is one of our architectural styles. Decomposition will allow us to create our system in loosely coupled services which work independent of one another which allows us to develop areas of the system without having to worry about areas which are still being developed.

2 Architectural quality requirements

2.1 Quality requirements

2.1.1 Security

- Security is the most important quality requirement due to the high secure nature of the generated certificates, special security measures must be deployed for securing a High value certificate.
- The system should be able to accurately validate a user's password and email address (authentication) to prevent hackers from accessing user account and to prevent fraudsters from verifying their organisations and generating a certificate, only Admins should be able to do that.
- The transactions need to occur securely and timely, to ensure a fully secure platform , Users data and sensitive information will needs to be encrypted before storing them to the database such that it cannot be accessed by an unauthorised person.
- One of the requirements of the Give Alot system is that there should be role based access control to ensure that only authorised parties should be able to edit/view contents of certain data resources. As our application will have multiple users each with different privileges to access specific resources, we are using the RBAC (Role Based Access Control) pattern to assign privileges to specific data resources as required by the roles and courses assigned.
- By making use of the client server architecture, It allows us to Enforce transport layer security, allowing SSL/TLS connection security across the backend servers as well as endforce various other security policies applied to the services that we will be exposing.
- Our system will make use of the RBAC (Role Based Access Control) pattern to assign privileges to specific data resources as required by the roles and courses assigned
- To protect the users information such as passwords, we can use an encryption algorithm such as MD5,The MD5 message digest hashing algorithm processes data in 512-bit blocks, broken

down into 16 words composed of 32 bits each. The output from MD5 is a 128-bit message digest value.

- we will be using two-factor authentication for users to confirm their identities when they are making a donation.

2.1.2 Performance

- Each component or service running on the server-side should efficiently support concurrency and parallelism. Furthermore, each service implementation should natively solve the C10k problem (problem of optimizing network sockets to handle more than 10 000 of clients at the same time) with reasonably low resource overhead.
- The web applications pages should load quickly on multiple users browsers within a reasonable timeframe of 5 seconds.
- The time between when the user initiates an action and when the computer starts to display the result (i.e System response time) should be around 200-300ms.
- The DB system should at minimum support 10 000 read operations p/s and 2000 write operations p/s.
- Micro service architecture has faster rendering and availability of applications with zero downtime.
- Performance can be achieved through control of resource demand of reducing overhead. The nature of micro service architecture of distributed web server caching reduces the database read load which helps to improve the performance of the system.

2.1.3 Scalability

- This refers increasing the system's capacity for work, while still performing well.This can be implemented through the use of a database, which will allow the users to make many changes in a short period of time.
- Scalability reflects the ability of the system to grow or increase in size with the user's demands of expansion of functionalities and capacity of the system.
- The system should be scalable in order to accommodate an increase in the number of users, the number of Organisations that can be added, as well as the number certificates that can be generated.
- The system should be able to handle 50 requests per second and support 50 000 daily unique visit
- Microservices architecture are independent in nature. Therefore, both the module and the entire system as a whole is scaled. But, the subsystems can be scaled up and down depending on the load. Therefore, microservices has the capability to cope up with the expanded and

decreasing load and can perform well in both conditions. Microservices enhances scalability apps by 64% and app resilience by 50%.

2.1.4 Availability

- The core of the platform should be built as a resilient distributed high availability cluster that can span and serve multiple geographies and data-centres
- Our system should have an up time of at least 99.5% hosting with services like AWS (Amazon web services)
- Availability is also related to fault and failure of the system. We can use fault avoidance, fault removal or fault tolerance (redundancy) to ensure high availability of the system. Exception handling- In case where an organisation is not registered with Give Alot, The system will make use of Other public APIs including Google search open API so search for an organisation.
- The presence of the fault must be identified or expected. This can be through a ping/echo. The user should be notified on the system when there is a fault also when the maintenance is to take place.

2.1.5 Reliability

- The platform should operate consistently under any traffic loads
- Our system should account for a peak traffic of 20 times it's average daily traffic, we'll also use past peak traffic data to add a percentage growth in traffic to arrive to a final number.
- We will determine the systems capacity by using load-testing tools such as "Load impact" to simulate traffic to the system, In that way we will be able to understand how our system performs under high traffic and tune the system before peak traffic arrives.
- The system will also use past peak traffic data to add a percentage growth in traffic to arrive to a final number.

3 Architectural design pattern and Architectural design styles

The Givealot system will employ the Model View Controller architectural design pattern, with the Controller employing the Microservices,Peer to peer and Client server architectural styles. The MVC pattern will work for Givealot as it provides the seperation of concerns this allows the developers to work on work on different ascpects of the application at the same time. The Micoservices design style is used for its reliability as it promotes loose coupling if a failure occurs on a service the failure will be isolated on that service will not affect any services that follow. The Client server architecture is used as it provides security as servers have better control access and resources to ensure that only authorized clients can access or manipulate data and server updates are administered effectively. The Peer-to-Peer design style is used for block-chain intergration as it consists if a decentralized network of peers.

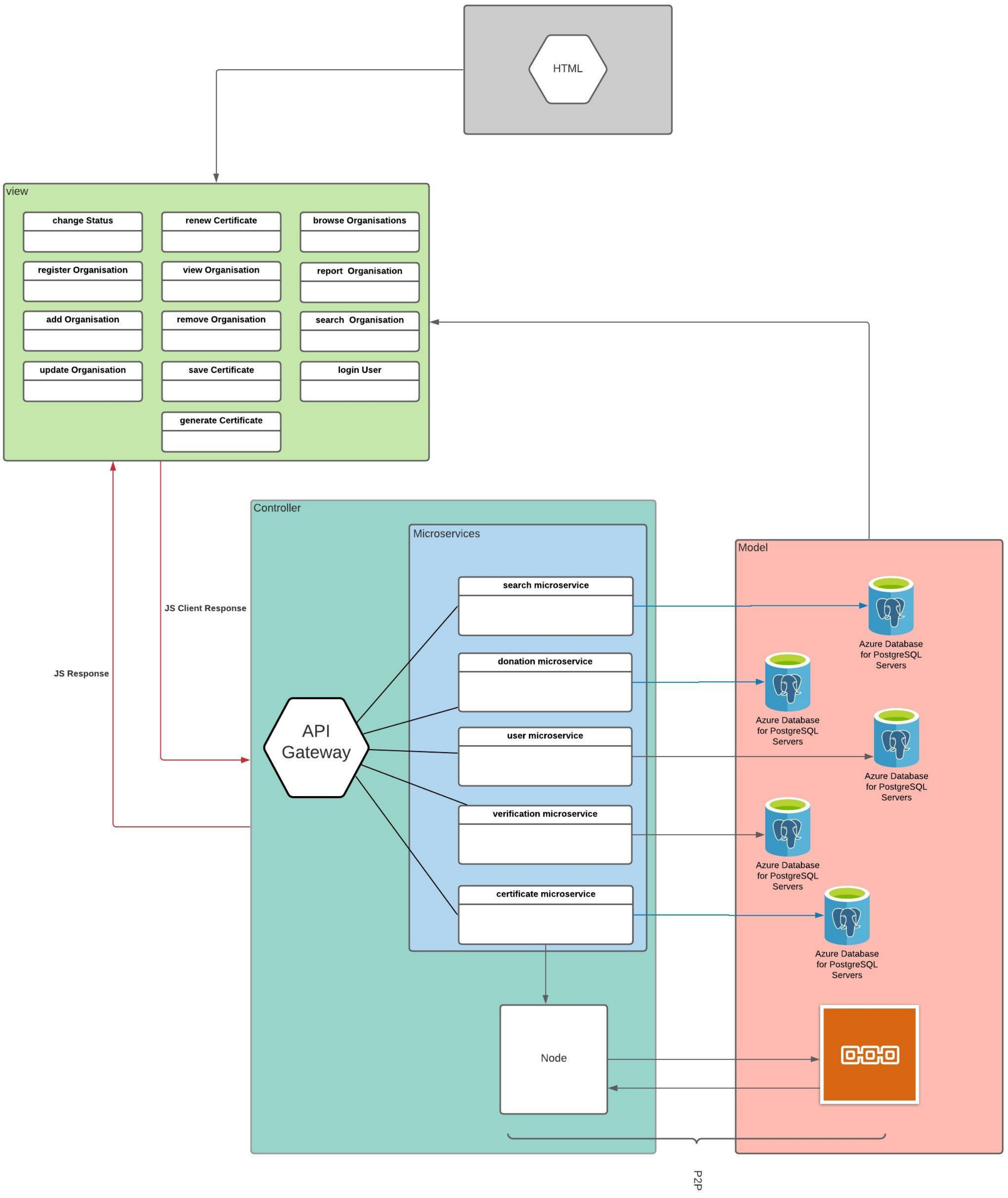

Figure 1: Architectural pattern and styles diagram

3.1 MVC Design Pattern

3.1.1 Model

- The model interacts with the controller when data needs to be fetch from or created in the system, or when it requires the controller to update, insert or delete data.
- The subsystems in the model each represent a unique system with its on functionality, each of which make various calls to the micro services to manage/handle data.

- The model encapsulates each subsystem and business information as a module that functions in a loosely coupled way independent of one another.

3.1.2 Controller

- The controller encapsulates the micro services of the system.
- Each micro service is connected directly to a database, and the controller can be seen as an interface between micro services and the database.
- The micro services in the controller can be seen as packages, each a modular functional service which the system can use independent of the others.
- Some micro services also interact with the server and block-chain and the controller provides centralization for outgoing and incoming data.
- The controller interacts with the model when data needs to be fetch from or created in the system, or when it requires the model to update, insert or delete data.
- The subsystems in the controller each represent a unique system with its on functionality, each of which make various calls to the micro services to manage/handle data.
- The controller encapsulates each subsystem as a module that functions in a loosely coupled way independent of one another.

3.1.3 View

- The view exchanges requests and response with the controller.
- Each item represents the functionality the user will have on the client-side which will make calls to the back-end functionality.
- The view interacts directly with an API gateways which communicates the user actions on the web page to the view.

3.2 Architectural Design Styles

3.2.1 Peer to peer

- The node of the system interacts with the block-chain to fulfill the creation and maintenance of the block-chain certificates

3.2.2 Client Server

- The verification and certificate micro services interact with the server to; store and retrieve hard copies of the created certificates, store and access report logs, and store user data such as profile pictures and gallery images for organisations.

3.2.3 Micro Services

- All of the micro services are stored in the model, each micro service has a unique function and works independently of each other.

4 Architectural constraints

4.0.1 MVC

With the the model view controller architecture as part of our architecture of choice, it's design means that either responsiveness or performance of the give a lot system has to be prioritised given constraints such as additional server cost or limited bandwidth usage.

Performance: In the MVC architecture - if we put performance as the first concern, more systems of the give a lot application will be on the model and as a consequence to this, the application will use more server resources and this will result in a monetary penalty to the stakeholders if a pay as you use service such as AWS was used for hosting. However this approach means that the give a lot's core systems will be allocated as much resources as they need hence increasing the overall performance however less resources will be allocated towards making the system responsive to the end user.
Responsiveness: In the MVC architecture - if we put responsiveness as the first concern, then more resources of the give a lot application will be put on the controller module (I.e animations, triggers to control animations amongst other events, pre-fetching data, cookies etc). This will not affect the server's performance however additional bandwidth usage will incur since more information will be transferred to the end-user at once without their intervention at times and the end user's with low-powered devices will not have the best experience.

4.0.2 Micro-services

The micro-services services architecture allows each system of the give a lot application to have a loosely coupled implementation and a loosely coupled database connection making the functionality of each system (excluding the correctness) independent of one another however this raises a problem of reliability, debugging and testing.

Reliability: In the micro-services architecture, each system is implemented as it's own application communicating with other systems via APIs - so a server malfunction is not only consequential to the application(s) it is hosting but also the other application(s) depending on the affected system.

Debugging and testing: With the generous number of services collaborating with one another in the controller, implementing the Micro-services architecture on the controller will render it hard for the developer to debug and test concurrency based issues due to the asynchronous nature of the micro-controller service.

4.0.3 Peer to peer

The peer to peer architectural pattern implemented in the give a lot system due to it's nature where a computer's network is accessed by many other's in the system slows down the network of the give a lot system which further has an effect on the responsiveness of the overall system.

5 Technology choices

5.0.1 Give Alot Web Application

The web application needs to seamlessly and efficiently accommodate growth. It should also be performant to so that users can access the services that Give Alot has to offer.
React

- React.js often referred to as React or ReactJS is a JavaScript library responsible for building several UI components or, in other words, responsible for the variants of UI components. It provides support for both frontend and server-side. React is developed by Facebook. It is most commonly used for handling the view layer for web and mobile apps. ReactJS/React allows users to create reusable UI components. It is currently one of the most popular JavaScript libraries and has a strong foundation and large community behind it.

Vue

- Vue is a model view view-model JavaScript framework that provides declarative rendering and component composition. Like Angular, Vue has its own routing and management with a CLI. Vue is easier to code but harder to understand as it makes use of directives with HTML bindings to send and receive data from DOM. Vue is a lightweight however, Vue's core library focusses on the view-layer.

Angular

- Angular supports Typescript which supports linting and OOP by design making it easier to code larger applications in faster iterations. All functionality and features of Angular are supported in Ionic, which makes it easier to re-use components and features in a cross-platform application

Technology choice for FrontEnd is React

- React is our frontend technology c because it offers component re-usability which speeds up the application development and improved performance due to the virtual DOM. React is also an ideal view framework to build re-usable and scalable components.

  On the other hand, React popularized the idea of a single component containing JavaScript and markup. This approach contrasts with the model view controller norm. While there are definite benefits to React's approach, some may prefer separation.

5.0.2 Give Alot Server

The application server used must support rolebased access control. There should no direct access to the database. All services provided by the server accessed through the given role.
Spring boot

- Spring boot is an application framework and inversion of control container for the java platform,The main goal of the Spring Boot framework is to reduce overall development time and increase efficiency by having a default setup for unit and integration tests, the framework's core features can also be used by any Java application. it makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

NodeJs

- Node.js is an open-source, cross-platform, back-end, JavaScript runtime environment that executes JavaScript code outside a web browser. The reason why we chose this technology is because Node.js is primarily used for non-blocking, event-driven servers, due to its single-threaded nature. It's used for traditional web sites and back-end API services, but was designed with real-time, push-based architectures in mind. It is also useful for developing heavy-load applications and e-commerce sites that depend on the speed of processing.

Python

- Python is an object-oriented, high level, dynamic and multipurpose programming language. Python's syntax and dynamic typing with interpreted nature, make it an ideal language for scripting. The reason why Python is not chosen is because it is not exactly the best option for asynchronous programming as our system should be non-blocking.

Technology choice for backend is Springboot

- This is because Spring Boot utilises the MVC module which provides full RESTful capabilities.

5.0.3 Give Alot Databases

PostgreSQL

- PostgreSQL is an open source, object-relational database system. It's a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, just as with pure relational systems, it supports extension of the data model with custom data types and methods. It boasts higher read and write speeds and this is directly related to the performance of the system which we have also identified as a quality requirement.

Mysql

- This is a purely relational database. MySQL is still very fast at reading data, but only if using the old MyISAM engine. If using InnoDB (which allows transactions, key constraints, and other important features) ,differences are negligible (if they even exist). These features are absolutely critical to enterprise or consumer-scale applications, so using the old engine is not an option. The reason we didn't choose MySQL is because it is no longer free to use or open source which is a constraint that needs to be adhered to.

MariaDB

- MariaDB is a lightweight database software that puts minimal strain on technical resources. For businesses with less data to store, or that can't support heavy database software. MariaDB also offers flexible table partitioning, allowing for horizontal partitioning of tables. Lastly, MariaDB supports many programming languages, including JavaScript, Python, Ruby, and more. The reason for not choosing this technology is due to the limitations it has. MariaDB doesn't match the high read and write speeds of other databases and it does not support OS X.

Technology choice for Database is Postgres

- Give A lot has a constraint that the system should only make use of open source technologies of which PostgreSQL fits the criteria.

6 Testing Technologies

6.0.1 Front End

- Jest and Enzyme. Jest is a unit test framework designed by Facebook to test react applications. Jest allows to write unit tests using the Snapshot feature. Enzyme allows to write unit tests using regular assertions. Using Enzyme with Jest makes writing tests for React applications a lot easier.

  alternative

- For Front end testing, we will be using Jest and Enzyme,
  Jest is a library for testing JavaScript code, it's an open source project maintained by Facebook, it's especially well suited for React code testing, it's strengths are: it's fast and it can perform snapshot testing.
  Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate and traverse React Components.
  Using Enzyme with jest makes writing tests for react application a lot easier.

6.0.2 Backend

- Mockito. For back end testing, Mockito will be used since it is an effective tool for JAVA application unit tests and integration tests. It is especially useful for our unit tests because of the modular way in which we coded our systems. With Mockito being simple to use with an intuitive language, as well as our team having experience with it in the past, it was an easy choice for our testing framework.