UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and
Information Technology

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS301 - Software Engineering

Atbash

Architectural Requirements

August 17, 2021

Item: Capstone Project - Demo 1

Team Name: Bit by Bit

Team Members:

Name	Surname	Student Number
Liam	Mayston	u19027801
Dylan	Pfab	u19003961 *
Connor	Mayston	u1906936
Joshua	Reddy	u19196042
Targo	Dove	u15020275

* - *indicates team leader*

Contents

1 Architectural Design Strategy

There are three main design strategies.

The decomposition design strategy consists of decomposing designs into sub-designs in a divide-and-conquer approach to dealing with high complexity. This approach is most suited to highly complex problems that come with a large number of requirements and/or require a great deal of functionality.

The quality requirement design strategy consists of ordering quality requirements based on priority and then selecting approaches/solutions whose characteristics best align with the prioritized quality requirements. This approach is most suited for simpler problems that consist of hard constraints and critical quality requirements.

The design by 'generating test cases' strategy consists of initially treating the solution as a black box and specifying the way in which it will be interacted with. This consists of describing the inputs and resultant outputs for given scenarios. This approach is most suited for problems where the functionality is more important than how that functionality is implemented.

The design strategy chosen for designing Atbash's Architecture is the 'Design based on quality requirements' strategy. This strategy was chosen due to the Atbash project consisting of a well understood problem with critical quality requirements and constraints. The aim of the project is designing a solution that addresses the specified quality requirements and constraints.

2 Architectural Styles

The Atbash system comprises of multiple architectural styles within the separate categories that best encompass the design we needed to implement the core requirements of the project. While the Atbash can broadly be described as a Client-server 4-tiered architecture, the chosen architecture does draw upon elements from multiple styles.

2.1 Considered Styles

2.1.1 Structure

- Monolithic application: A monolithic application is a single-tiered software application that consists of a single program from a single platform that incorporates the user interface and data access all in one. A monolithic application is self-contained and is designed without modularity. This type of architecture is the simplest to create but makes changing the software very difficult.
- Component Based: Decomposes application design into reusable functional or logical components that are location-transparent and expose well-defined communication interfaces.

2.1.2 Shared Memory

- Database-centric: Database-centric Architecture is an architecture in which databases play a crucial role. A standard, general-purpose relational database management system is typically used. A shared database usually forms the basis for communicating between parallel processes in distributed computing applications.

2.1.3 Messaging

- Event-driven (Implicit Invocation): Event-driven architecture (EDA) is a software architecture paradigm promoting the production, detection, consumption of, and reaction to events.

2.1.4 Distributed Systems

- Multi layer (2-tier, 3-tier, n-tier exhibit this style): A multi layer architecture is a client–server architecture in which presentation, application processing and data management functions are physically separated. This allows developers to create flexible and reusable applications. Layers can be modified, added or removed without having to rework the entire application.
- Client-server: A client-server architecture segregates the system into two applications, where the client makes a service request to the server.
- Shared nothing architecture: A shared-nothing architecture (SN) is a distributed computing architecture in which each update request is satisfied by a single node

(processor/memory/storage unit). The intent is to eliminate contention among nodes. Nodes do not share (independently access) the same memory or storage. In databases, another term for SN is sharding. A SN system typically partitions its data among many nodes.

- Peer-to-peer: Peer-to-peer (P2P) computing or networking is a distributed application architecture that partitions tasks or workloads between peers. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes. Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts.

- Representational state transfer (REST): Representational state transfer (REST) is a software architectural style that was created to guide the design and development of the architecture for the World Wide Web. REST defines a set of constraints for how the architecture of an Internet-scale distributed hypermedia system, such as the Web, should behave. The REST architectural style emphasises the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching components to reduce user-perceived latency, enforce security, and encapsulate legacy systems. REST has been employed throughout the software industry and is a widely accepted set of guidelines for creating stateless, reliable web services.

- Service-oriented: Refers to Applications that expose and consume functionality as a service using contracts and messages. Microservices are a modern form of service-oriented architectures. Services in a microservice architecture are individual processes that communicate with each other over the network in order to fulfill a goal. Microservices provide benefits such as modularity and scalability. Different services can be worked on, updated and scaled independently of each other.

2.2 Chosen Styles

2.2.1 Shared Memory

- Database-centric: The Atbash system required some form of central communication management in order to locate and keep track of users and temporarily store messages when the recipient user is offline.

2.2.2 Distributed Systems

- Client-server (4-tier): The 4-tier client-server is well suited for the Atbash system. On the mobile-client side, separating the user-interface (Presentation tier) from the underlying services (Application tier) allows making changes to those layers without having to make changes to the entire application. Allowing the client software to directly edit the database (Data tier) on the server would result in a major security risk due to the server database being shared by all users. Another layer (Application tier) is required on the back-end side between the client software and back-end database to ensure secure management and data consistency of the central database. A 3-tiered architecture is also implemented on the client-side for the local database. Due to the client being the only one with access to the local database, a middle tier is not required.

- Representational state transfer (REST): REST is a software architectural style that was created for the World Wide Web and is thus well suited for the communication of components through the World Wide Web. This style has been incorporated into the Atbash architecture to provide uniform interfaces for communication between the client and back-end layers.

- Service-oriented (Microservices): Due to the possibility of the Atbash system needing to scale to handle a large number of requests per second, the microservices architecture is well suited to allowing the server to scale resources dynamically when needed.

3 Architectural Quality Requirements

The Atbash system has a number of Architectural quality requirements that directly co-relate with the most important aspects of the project.

3.1 Security

Security is the most important quality attribute for the Atbash system. The system needs to be completely end to end encrypted and no one should be able to read the contents of the message other than the sender and receiver.

- This means the application should make use of the current state of the art standards for secure communication. such as that provided by the open source Signal protocol which combines the Double Ratchet algorithm, prekeys, and a triple Elliptic-curve Diffie–Hellman (3-DH) handshake, and uses Curve25519, AES-256, and HMAC-SHA256 as primitives. Data security is ensured as it would take an exponential amount of time to crack the algorithms and access the data. (The AES-256 algorithm would take $2.29x10^3$[2] years to crack)

- This means that the server should not be able to read or decrypt any information in a message package that is not critical for delivering the message. i.e. The server should not be able to extract any information other than the sender and receiver numbers.

- The server should not store or collect any information other than what is necessary to provide its intended functionality. e.g. Message packets should be deleted once delivered and encryption keys must be deleted once fetched from the server.

- All communications with the server should be encrypted, critical messages should be signed to prevent tampering and message contents between two users should also be encrypted.

Justification: This addresses the core system requirements to enable peer to peer encryption and follow the best security practices.

3.2 Portability

The system needs to be fully operational on both IOS and Android and have the same look and feel on both systems. **Justification:** This ensures that the application can be used on and to communicate with as many mobile devices as possible.

3.3 Availability

The core functionality of the front-end should still be usable even when there is no internet connectivity (for example new messages can be created and sent and will be queued and sent when internet connectivity is restored). The system should be available at least 99% of the time. **Justification:** This will ensure intermittent and weak connections will not degrade the user experience and make using the application a frustrating experience. This will also ensure that application will always be usable when required.

3.4 Performance

Instant messaging applications are common place today and users have come to expect responsive applications that provide instant results:

- Navigation to different screens should take no longer than 1 second.
- User input should result in a visible change in the interface within 500ms.
- When opened the application should load within 2 seconds.
- Queries to the server should result in a response within 3 seconds.
- When both devices have a strong internet connection, a sent message should be received on the recipient device within 5 seconds.
- When more intense processing is required (such as for initial setup), the user must be notified of the processing and it should not take more than 10 seconds.

Justification: This will ensure the application meets user expectations and will improve the users experience.

3.5 Scalability

The chosen architecture should allow for both vertical and horizontal scale-out so that the application would be able to easily scale to a user base of 50,000 users within a month and process up to 50,000 requests per second up to a maximum of 250,000 requests per second.**Justification:** This will ensure that the system can easily scale to meet higher demands.

3.6 Usability

Usability is one of the most important quality attributes. Computer literacy cannot be assumed as anyone could use this app however it can be assumed that users have used a messaging application before. Functionality should be as intuitive as possible and provide instructions where necessary. The system needs to be easy to use and understand by anyone who picks it up. Error messages should be self-explanatory and as much as possible of the input validation should be done on the client.

- users should not find any aspects of the system cumbersome, non-intuitive or frustrating,
- the use of colors should not be evasive and distracting,
- user interfaces should not be cluttered and hard to use.

Justification: This ensures the application will be able to service a wide ranges of users with varying abilities and technical know-how and will ensure that using the application is a pleasant experience.

3.7 Testability

All services offered by the system will be testable through unit tests and integration tests. This will consist of:

- automated unit tests testing components in isolation using mock objects, and
- automated integration tests where components are integrated within the actual environment.

Unit tests will test whether the system functions and classes yield correct, predictable results. Integration tests should test that all needed services from other subsystems are available and work together.

Acceptance testing will be used to determine whether the system meets the use cases and functional requirements described in this document. This will be done by verifying:

- that the service is provided if all pre-conditions are met (i.e. that no exception is raised except if one of the pre-conditions for the service is not met), and
- that all post-conditions hold true once the service has been provided.

Justification: This will ensure most errors produced when updating the code can be quickly identified and located. This will greatly improve the quality of the resultant software.

4 Architectural Design and Pattern

4.1 Architectural Diagram

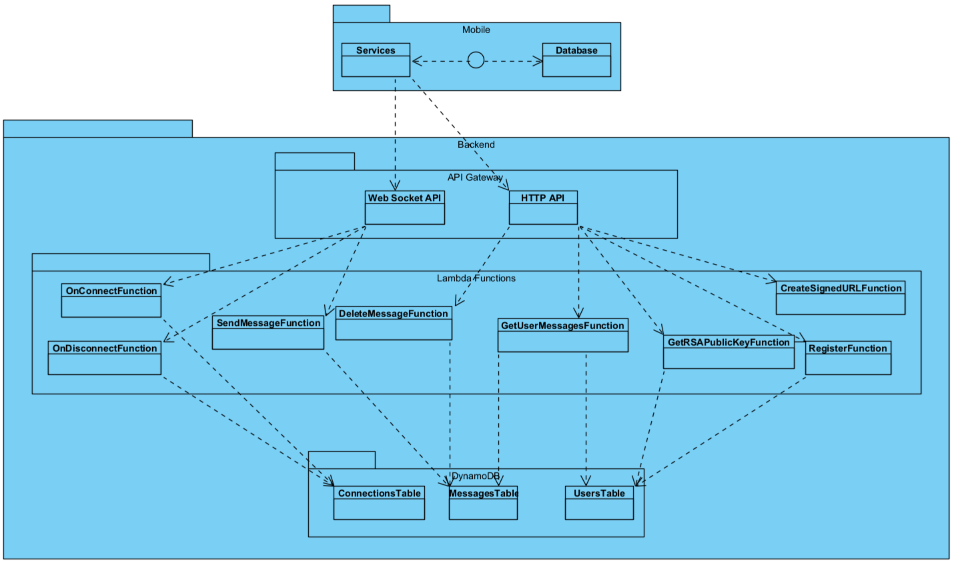

Figure 1: Diagram showing the Architecture Diagram for the Atbash system

4.2 Architectural Patterns

The architectural patterns we chose to use that would best fit the core concepts of Atbash being encryption focused are:

- 4-Tier: The Atbash system as a whole will be making use of 4-Tier Layered pattern. Within this pattern we will make use of a Model-View-Controller pattern, a Layered pattern and Microservices. The Mobile/client side contains a Presentation tier along with an Application tier and a Data tier. The Backend/server side contains an Application tier, which is made up of Microservices, and a Data tier. Communication between the client and server is done between the client-side application tier and server-side application tier. The Database layer allows for a central management area, where you can manage your users. The Backend layer provides protection of the database so that clients can't edit or read the data without the necessary permissions.

5 Architectural Constraints

The Atbash system contains a small number of constraints which had to be considered when choosing the architecture. The constraints consist of:

- The architecture must be available on IOS & Android.
- The architecture must be end to end encrypted.
- The architecture must be deploy-able on Amazon Web Service Servers.
- All libraries should be open-source where possible.

6 Technology Choices

Technological choices are a huge part of designing any form of application. It is vital to ensure that you choose the correct technology for all required systems to ensure that the application will not only run effectively but at maximum efficiency. The correct choice of technology will also aid in development of the product and allow the developers to create better and more productive code. The choices for the system as well as all their pros and cons are explained below with our final choice.

6.1 Mobile:

- Flutter
  - Overview - Flutter is a UI toolkit designed by google that allows for the development of natively compiled applications onto IOS & android while only using a single code base. Flutter is coded using Dart.
  - Pros:
    - Single code base
    - Native compilation
    - Fast deployment
    - Easy to use widget system
    - Hot reload allows for compiling changes in real time
    - Maintained and updated frequently by Google
  - Cons:

    - ∗ New language to learn (Dart)
    - ∗ App sizes are bigger than coding with native language
  - How technology fits - Flutter fits into our architectural constraints and requirements quite nicely as it allows us to code one front end application that can be compiled to work natively with both Android & IOS which greatly speeds up deployment time. It also has a lot of inbuilt UI widgets that make it somewhat easy to pick up and use efficiently which will be necessary if any of the developers have never used it before.

- Java & Swift (Native code for Android & IOS)
  - Overview - Java is very well known and is extensively used by many programmers including all the developers on the Atbash project. It is object orientated, very secure and robust which makes it a top choice to code in. Swift was developed by Apple as a native platform to code for IOS, it is very modern with many neat features and is very safe by design, accounting for and preventing a lot of minor inconveniences such as type safety and memory overflow.
  - Pros:
    - ∗ Native apps allow for the best performance possible
    - ∗ The space requirement is at its bare minimum when working natively
    - ∗ Native apps maintain a recognizable look to them compared to using third party software.
  - Cons:
    - ∗ Would have to learn and implement two separate languages
    - ∗ Would need to have two separate code bases
  - How technology fits - Java & Swift fit into our Architectural designs as they allow for Atbash to work on IOS & Android however they are also very time consuming and require two code bases to work on all mobile platforms.

- React Native
  - Overview - React Native is a open source application framework created by Facebook that is coded and used as a java script library. It allows you to code native applications while using react's framework.

  - Pros:
    - Is written in java script
    - One code base can be compiled natively
    - Fast refresh allows for changes to be seen immediately
    - Maintained by Facebook
  - Cons:
    - Lack of custom modules
    - Compatibility issues
    - Testing and debugging has been known to be hard
  - How technology fits - React Native is a great choice for our system as it allows us to use one code base and compile it natively to still get maximum performance. However it does not offer great testing features. However it does fit into the architectural patterns, designs and styles we have chosen.

6.2 Web:

- Angular
  - Overview - Angular is a platform and framework that utilizes HTML and TypeScript to create single single page applications. It makes use of modules and components to simplify and expand the possibilities of working on a web page.
  - Pros:
    - Allows for dependency injection
    - Two way data binding, any changes made in the application or view get immediately reflected in the other.
    - Maintained by google
  - Cons:
    - Performance can be less than optimal
    - Known to have a steep learning curve
  - How technology fits - Angular would be an amazing framework to pick for the web development of Atbash. It allows for quick and easy coding with two way binding and is regularly updated and maintained by google allowing our system to be maintainable into the future.

- React
  - Overview - A JavaScript library developed by Facebook for the building of SPA for both web and mobile. React renders to the Document Object Model directly in the browser. It is used to develop front-end UI and can be used for mobile development or single web pages.
  - Pros:
    * Virtual DOM used
    * Easy to learn
    * Reuseable components
  - Cons:
    * poor documentation
    * Not a fully fledged framework as of yet
  - How technology fits - React is another great option for developing the web side of Atbash. However it is not fully implemented and designed as of yet and so it may be hard to work with and use to its full extent.
- VUE.js
  - Overview - VUE is a progressive framework for building user interfaces. The core library is only focused on the view layer. It allows you too extend HTML with html attributes called directives. It was created in 2014.
  - Pros:
    * Virtual DOM rendering
    * Reactive two way data-binding
    * Integration capabilities
    * Precise documentation
  - Cons:
    * Low amount of plugin's and extensions
    * Lacking support for large scale projects
    * Lack of experienced developers and community
  - How technology fits - VUE is a less optimal choice for Atbash as it only really targets the view layer of the web pages which means we would need to use other software to accommodate. Additionally it does not scale well.

6.3 Web Server:

- Spring Boot (Using Kotlin)
  - Overview - Spring is an application framework for the java platform. It was developed by Pivotal Team and can be used to create stand alone applications. Kotlin is a language developed by JetBrains that targets the JVM. This means that any Java code is compatible with Kotlin allowing the massive library support of java and the power of kotlin in one package.
  - Pros:
    - Reduced time on development
    - Provides default testing units
    - Allows integration of many Spring features like security and ORM
    - provides plug-ins
    - Allows for easy connection to databases
    - Kotlin is less buggy than java
    - Easily maintainable
    - Kotlin features more advanced than java, such as type safety
  - Cons:
    - A lot of things to understand before Spring can be used to its full potential
    - Kotlin can have a fluctuating compilation speed
  - How technology fits - Springboot is an amazing framework that would fit Atbash incredibly well. Atbash heavily relies on security and is very database-centric which Springboot has in built security ecosystems and ORM managers. Additionally Kotlin is an amazing language that offers a lot of flexibility and options compared to Java. This would allow for more rapid development.
- Express (Using JavaScript)
  - Overview - Express is a flexible and minimal Node.js framework that offers a robust set of features for development on web or mobile. It helps organize your application into a MVC architecture server side. JavaScript is a text-based programming language that can be used on both server and client side that allows for the creation of interactive web pages.

  - Pros:
    - Easy to learn
    - Well documented
    - Highly flexible
    - Large amount of packages available
    - JavaScript is easy to use and has been well documented and learned since 1995
    - JS is really quick as it is run client side
    - Able to be used on any web browser
    - JS is run client side, reducing server load
  - Cons:
    - Express is not huge on security
    - No easy testing
    - Javascript works differently on different browsers
    - Lack of debugging in JS
  - How technology fits - Express is highly flexible and offers a lot of versatility which would benefit the ever expanding design of Atbash however, the major component of Atbash is the security which Express is known to be relatively bad at and so the choice for it is unlikely.
- DJango (Using Python)
  - Overview - Django is a high level Python web framework. It promotes rapid development and deployment. It was created by Django Software Foundation in 2005 with the foal of easing the creation of complex, database driven websites.
  - Pros:
    - Rapid Development
    - Scalable
    - Secure
    - Uses python, a quick to learn and easy to use programming language
    - In built ORM.
    - Batteries included framework

  - Cons:
    - ∗ Uses python which is known to be slow
    - ∗ Not great for smaller websites
    - ∗ Monolithic design
  - How technology fits - DJango would suit our Architecture very well as it is very secure and scalable, meeting two requirements immediately. It also has an in built ORM and comes installed with many features, which is great for our database-centric structure. However it is known not to be great for small websites which Atbash will likely be, and it utilizes python which is known to be slow, and in a messaging application where security is king, that may not be acceptable.

6.4 Database:

- PostgreSql
  - Overview - PostgreSQL is a open source object-relational database emphasizing extensibility and SQL compliance. It was released in 1996 by PostgreSQL Global Development Group. It supports many data types and is compatible with many frameworks.
  - Pros:
    - ∗ Open Source
    - ∗ Highly expandable
    - ∗ Compliant with SQL standard
    - ∗ Can process complex data types
  - Cons:
    - ∗ Low reading speed
    - ∗ Very strict data storage
  - How technology fits - PostgreSQL is an amazing and well supported Object-Relational database and utilizes SQL. It is very scalable and can store complex data which may come in handy for the Atbash architecture. It is also open source which is great. Importantly, it can be mapped to by an ORM in many frameworks which means its easy to use and allows for quick develop-

ment. Additionally, access to the databases can be restricted which is great for security.

- Amazon DynamoDB
  - Overview - Amazon DynamoDB is a NoSQL database that supports key-value and document data structures. It was created by Amazon in 2012 and is included in its various Amazon Web Services.
  - Pros:
    - Fast performance
    - Continually supported by Amazon
    - Seamless scalability
    - Supports many data types
  - Cons:
    - Quite expensive if hosting a large database with many requests.
    - only deployable on AWS
    - Limited support in querying the database
  - How technology fits - DynamoDB is a non relational mapping database and so is easier to store and update on. It is also offered by Amazon Web Services which is our clients product and so it may be a great choice to make. However, it is not very expandable as it only works on AWS and querying the database can be trivial at times.
- Derbey
  - Overview - Derby is an open source relational database implemented in java. It was created int 2020 by the Apache Software Foundation.
  - Pros:
    - Can be embedded in any java framework
    - Supports JDBC and SQL
    - Easy to deploy and install
  - Cons:
    - Does not scale well
    - Is not very secure

  - How technology fits - Derbey is not a great choice for our architecture as it does not scale all that well and is not very secure which are two very important requirements of our system.

6.5 Final choices

- **Flutter** (Mobile): We ended up choosing flutter as our mobile technology simply because it is easy to use with an intuitive widget system. It has incredible documentation on how to use it effectively and is continuously updated and maintained by Google. Additionally, it offers in built testing and compiles natively to both Android and IOS, allowing us to have a single code base and get the best performance possible through native compilation. It also includes many security features and ORM features that fulfill the requirements of end to end encryption and our database-centric design. Overall it allows for a fast deployment time while allowing us to offer the app to all mobile devices and cater for the core requirements of the Atbash architecture.

- **Lambda Functions in Javascript** We chose to make use of lambda functions as they provide excellent horizontal scaling, allowing the server to meet the demands of a large number of users, while also reducing costs at times when load is low.

- **DynamoDB** (Database): We chose DynamoDB because of its ability to scale horizontally. If there is a large burst of users registering, or sending messages, the database can scale to meet their demands.

- **Signal Protocol**: The open source Signal protocol combines the Double Ratchet algorithm, prekeys, and a triple Elliptic-curve Diffie–Hellman (3-DH) handshake, and uses Curve25519, AES-256, and HMAC-SHA256 as primitives. Data security is ensured as it would take an exponential amount of time to crack the algorithms and access the data. (The AES-256 algorithm would take $2.29x10^{3\ 2}$ years to crack). HTTPS SSL encryption will be used for communication with the server.