TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Robin Sulg 154829IABB

DEVELOPMENT AND ANALYSIS OF A SIGN-IN AND CUSTOMER VERIFICATION MOBILE APPLICATION

Bachelor's thesis

Supervisor: Enn Õunapuu
PhD

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Robin Sulg 154829IABB

SISSE LOGIMISE NING ISIKU TUVASTAMISE MOBIILIRAKENDUSE ARENDUS JA ANALÜÜS

Bakalaureusetöö

Juhendaja: Enn Õunapuu
PhD

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Robin Sulg

20.05.2018

Abstract

The following bachelor's thesis is about developing and analysing a sign-in and customer creation mobile application that solves the issues of a business that manages multiple online gambling sites. The complexities that need to be improved regard the account creation process and the sign-in process of those sites.

The main purpose of this thesis is to create at least a mobile application's prototype that is capable of resolving the mentioned complexities and to analyse the prototype in detail.

As a result of this thesis, a mobile application's prototype was created and analysed. The prototype demonstrates the solution for improving the issues of the gambling sites.

The outcome of the bachelor's thesis is considered successful as the developed prototype provides answers for the highlighted problems.

This thesis is written in English and is 41 pages long, including five chapters and 15 figures.

Annotatsioon

Sisse logimise ning isiku tuvastamise mobiilirakenduse arendus ja analüüs

Käesolev bakalaureusetöö keskendub hasartmänguleheküljesid haldava ettevõtte jaoks loodava sisse logimise ning isiku tuvastamise mobiilirakenduse arendusele ning analüüsile. Hasartmängulehtede kasutaja loomise ning sisse logimise protsessid on keerukad ning nende protsesside võimalikuks lahenduseks on vastava mobiilirakenduse arendamine.

Bakalaureusetöö eesmärgiks on luua vähemalt mobiilirakenduse prototüüp, mis on võimeline lahendama hasartmänguleheküljede kasutaja loomise ning sisse logimise protsesside murekohad. Samaväärseks eesmärgiks on ka arendatava mobiilirakenduse põhjalik analüüs.

Lõputöö tulemusena valmis mobiilirakenduse prototüüp ning selle põhjalik analüüs. Valminud prototüüp on suuteline lahendama hasartmänguleheküljede kasutaja loomise ning sisse logimise protsesside keerukused. Lahenduse demonstreerimiseks seati üles testkeskkond.

Toetudes bakalaureusetöö tulemustele peetakse lõputööd õnnestunuks, sest töö käigus valmisid seatud eesmärgid.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 41 leheküljel, viit peatükki ja 15 joonist.

List of abbreviations and terms

API	Application Program Interface
Backend	A program that handles server-side code and usually located in the server
CPU	Central Processing Unit
Endpoint	A unique URL that represents an object or collection of objects
GET	Method for the HTTP protocol to retrieve data
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
HTTPS	Extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network
iOS	iPhone Operating System
IP	Internet Protocol
JS	JavaScript
JSON	JavaScript Object Notation
KYC	Know Your Customer
NPM	Node Package Manager
PIN	Personal Identification Number
POST	Method for the HTTP protocol to insert data
RSA	Rivest-Shamir-Adleman Algorithm
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SIM	Subscriber Identity Module
SMS	Short Message Service
SQL	Structured Query Language
URL	Uniform Resource Locator
UUID	Universally Unique Identifier

Table of contents

List of figures

1 Introduction

Identifying customers holds great importance for institutions as it helps to prevent identify theft, financial fraud, money laundering and many other problems that affect institutions [1]. The gambling industry is one of the many industries where the customer verification process is needed and common [2]. One example of this are online gambling sites that allow their customers to transfer funds by using cryptocurrencies. Because of the nature and idea behind many of the cryptocurrencies, the data that is revealed when transferring funds is less than one would by choosing a more conventional way like bank transfers. To hold more information about the identity of their customers, the identity verification processes of the mentioned institutions may be more into detail and thus more time consuming.

1.1 The problem

In order to attract more people, the gambling sites and the businesses that manage them seek better alternatives to simplify the identity verification process. A business that is managing a large number of gambling sites has thought of a possible solution that would allow them to offer a faster and more comfortable method for registering new accounts. The problem for this business is that every gambling site they manage require the user to go through an identification process before the gambler can start using the services that they provide. The identification process among the managed gambling sites is similar and because of this, the user who is interested in playing on different sites has to go through a comparable process over and over again.

The gambling sites, that the business guides, mainly uses two different forms of identity verification. One of the forms of verification is based on identifying the customer by reading the data from the user's national identification documents. This process is time consuming and may take sometimes weeks to finish.

As a result of the repetitiveness of the identity verification processes that many of the gambling sites are using, the person who is going through the process more than once is most likely not satisfied with the user experience that comes with it. The business that manages the gambling sites wishes to make the identity verification process less

repetitive for their customers. The identity verification process is also referred as the customer verification process.

In addition, the business also wishes to provide its customers with a more comfortable and secure method to sign in to their accounts as the current methods require the gamblers to navigate to the gambling site and enter the customary username and password combination to log in.

1.2 The goal

The goal of this thesis is to develop and analyse a solution that is capable of offering the gambling sites a faster, safer and smarter alternative for the current account creation and sign-in processes.

The solution is a at least a prototype of a mobile application that:

- Simplifies the identity verification processes by reusing known data and by reducing the repetitive parts of the processes
- Provides a comfortable and secure method for signing in to gambling sites

This document will examine the tools and frameworks, logic, services and other technologies needed for developing this solution.

2 Methodology

In order to create a mobile application both pleasant for the users and understandable for the future developers, the approach needs to be systematic. Without doing so, both the quality and the user experience of the application will deteriorate. The possible result of this is that the future developments and business logic implementation for expanding the application gets harder.

2.1 Existing solutions

As the identity verification process is much needed and used in many industries there are mobile applications that essentially do just that. One example of this is the popular application used by Estonian banks – Smart ID [3], [4]. However, the business that manages the gambling sites wishes to have their own implementation of an application that in addition to the sign-in and identity verification functionalities would also provide other, business logic specific, qualities.

2.2 Goals and requirements

2.2.1 Functional requirements

- As a user I want to register an account
- As a user I want to log in to my account
- As a user I want to see a list of gambling sites that I can create accounts to
- As a user I want to create accounts to the listed gambling sites
- As a user I want to see a list of gambling sites that I can log in to
- As a user I want to log in to gambling sites where I have registered an account
- As a user I want to verify my identity by using the mobile application
- As a user I wish to create accounts to gambling sites from the mobile application
- As a user I want to log in to gambling sites from the site by using the application

2.2.2 Non-functional requirements

- The mobile application should support devices running both Android and iOS (iPhone Operating System)
- The application should be secure. Sensitive information should stay private

2.3 Development process methodology

For organising the development process the agile Kanban methodology was followed. This means that software was continuously delivered to the client and the requested changes were listened to and implemented in a timely manner [5].

Whilst developing the application the following principles were followed:

- The application should be as automated as possible
- Use dependencies in order to save on development time
- Use third party services to save on development time
- The design of the application should be responsive and support mobile phones in any size
- The design of the application should feel modern

2.4 Architecture of the solution

The functional requirements of the mobile application are complex. In order to provide the users with the described features and to give the mobile application a modern feel, we have resulted in using third party services.

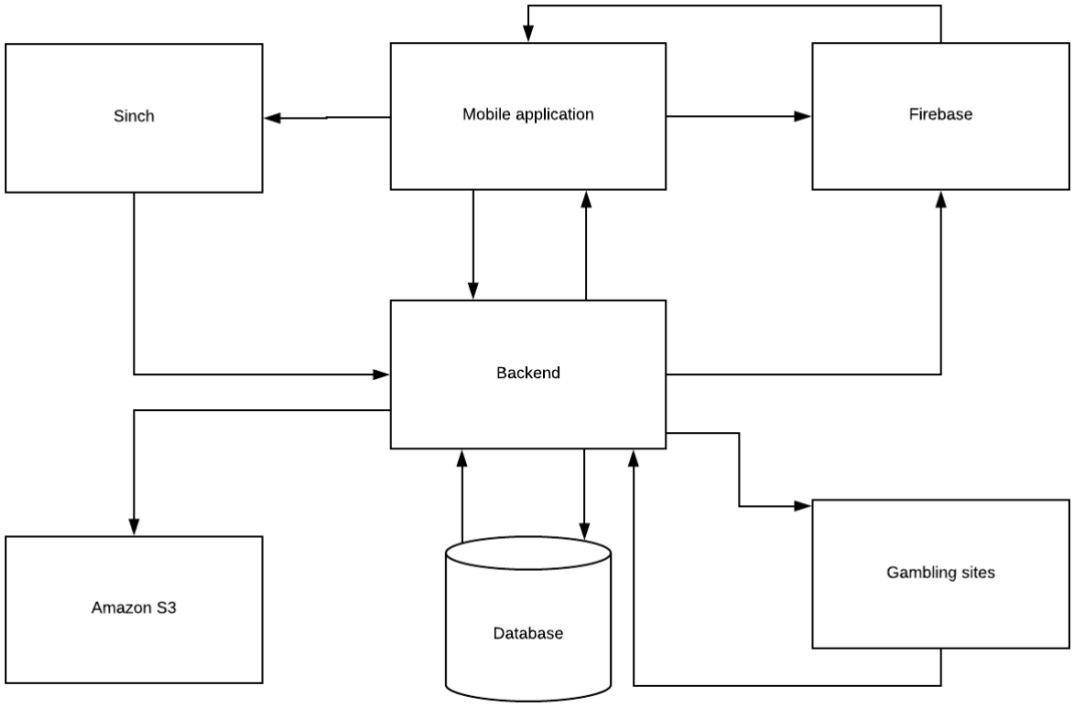


Figure 1. Architecture of the solution

Figure 1 illustrates the information flow of the solution designed for resolving the issues regarding identity verification and user sign-in.

2.5 Sinch

"Sinch is a cloud-based, mobile communications platform that makes adding Voice, Verification, Video, SMS, and IM into apps easier than ever. [6]"

Sinch plays a very important role in our application since it is providing us with the capability of verifying the user's phone number. The service that we are using is the Short Message Service (SMS) of Sinch. For a small fee Sinch is sending a 5-digit code to the user's phone number. This allows us to have some hope that the user about to create an account is more likely a human being with a non-malicious intent.

There are other companies that are offering the same service. The reason why we chose Sinch is due to the fact that it provides great documentation and is offering software development kits (SDK) that help with the development process.

2.6 Firebase Cloud Messaging

“Firebase Cloud Messaging (FCM) is a cross-platform messaging solution that lets you reliably deliver messages at no cost. [7]”

With the help of Firebase, we are able to send and receive push notifications. Push notifications are notifications that pop up on the mobile screen even when the application is idle.

By using push notifications, we are offering the user of our mobile application a comfortable way to initiate the sign-in process to various gambling sites and an assured method for receiving urgent messages from the application or messages about the gambling sites [8].

Using Firebase is a common practice amongst developers as they are providing their customers with an easy to use solution for incorporating the push notification service in to the project developed by the developer. The variety of documentations and SDKs that Firebase is offering is rich. The stated points are the reason why Firebase Cloud Messaging was picked for implementing the push notification capability for the mobile application.

2.7 Amazon and the Amazon S3 bucket

Amazon provides many web services. One of them being the S3 bucket service. This service allows us to store data like images in an environment where they are protected but yet easily accessible for users with the right credentials [9].

The application’s backend will also be deployed by using Amazon Web Services. More importantly the backend of the application is setup in Amazon’s EC2 instance by using Docker containers.

“Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. [10]”

2.8 Definition of entities

In order to give the reader of this document a better overview of the mobile application the main entities involved should be defined. The solution that we thought of required us to define three entities: customer, partner customer and partner site. With these three, we are able to create a mobile application that is capable of creating accounts to different sites, capable of signing in to them and capable of improving the repetitive nature of the national document based know your customer (KYC) polices of many gambling sites.

- Customer

  The customer is the registered user of the mobile application itself. A customer is created when the user has successfully gone through the customer creation process.

- Partner customer

  The partner customer can only be created once the Customer has been created. Partner customer is linked to a specific gambling site and refers to the customer.

- Partner site

  A third-party gambling site that has successfully managed the integration of the mobile application.

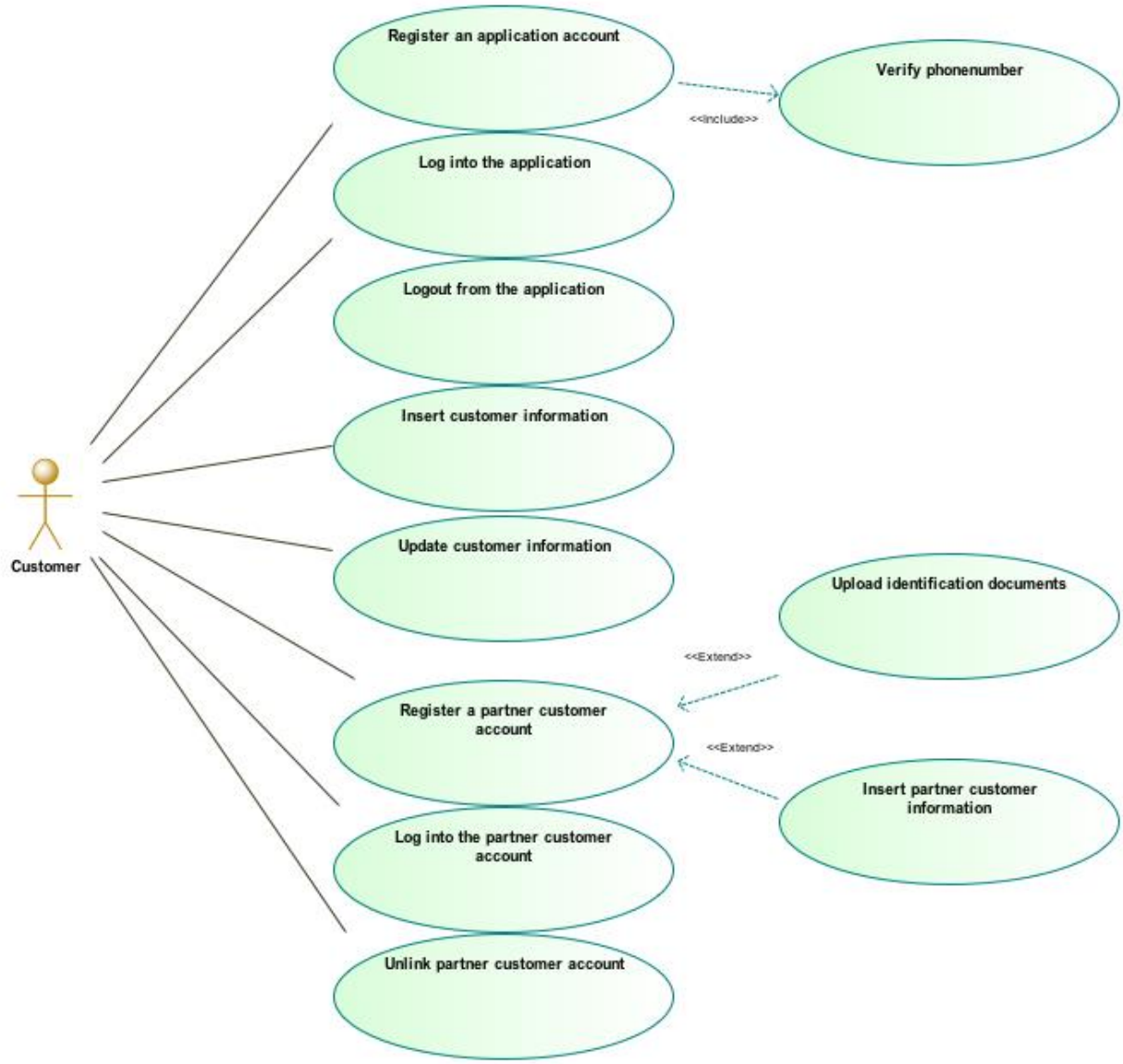


Figure 2. Use Cases of the Customer

Figure 2 is listing the use cases of the mobile application's customer entity.

2.9 Business logic requirements

Every solution needs to define logic requirements, otherwise there would be no guidelines on how the mobile application and the backend supporting it should act. The following are the business logic requirements:

- Every customer is bound to a single device where the application was installed and the customer was successfully created
- Every customer should be tied with a verified phone number

  Binding customers with a single verified phone number greatly decreases the amount of fake customer created within the application's space.
- Every customer should be tied to one document identity verification instance

  Once the customer has submitted his or her national identification documents the data gathered from those documents should stay immutable.
- Every customer can be bind to zero or many partner customers

  Since every gambling site listed in the application's lobby can have different requirements for creating an account, the application should be able to satisfy the KYC needs of every gambling site to a certain degree.
- Partner gambling sites should be able to specify the user information requirements for their gambling site

  When creating a partner customer, information specified by the partner gambling site should be displayed to the user.
- Data gathered from the document identity verification process (KYC) should override previously submitted user information.

  Once the document identity verification process has been confirmed, the customer should not be able to provide any further data about him or herself other than the data gathered from the Document KYC process. This rule only applies for data gathered from the documents.

2.10 Main development tools and frameworks

The tools, methods and frameworks used to build an application directly determines the outcome and the future of the project. With the current one, we have chosen widespread technologies which are supported by good documentation and community.

In order for the application to be successful it must have maintainers and by picking the following technologies that are moving upwards in the trend ladder, the shortage of future developers should not be an issue.

Another positive side by doing so is that the learning material and help from the community are easily available, which makes resolving problems a lot more efficient.

2.10.1 React Native

One of the non-functional requirements for the mobile application was that it should be available for both iOS and Android devices. This can be achieved by developing two separate native applications for Android and for iOS. That means that two applications must be written in different programming languages. In case of Android it would be Java and in case of the iPhone either Objective-C or Swift. However, lately there has come around a third alternative for writing native mobile applications and it is named React Native.

React Native is a framework created by Facebook that enables developers to build native mobile applications. React Native was born in the summer of 2013 as a Facebook's internal hackathon project and since then, the growth of the framework has been rapid. In 2015 React native was released to the public [11].

By using React Native you are creating a native mobile application with JavaScript (JS). This means that you are actually developing an iOS and an Android application at once. Depending on the features of the application some code might still be implemented separately for Android and iOS but the bulk of the code is written in JS [12].

JavaScript and truly native code run in different threads that communicate with each other over a bridge. One of the threads is the main thread, which is a standard native thread that handles displaying elements to the user and processes user gestures. The

other thread is a specific React Native thread which is responsible for executing JS code and managing the logic of the application [13].

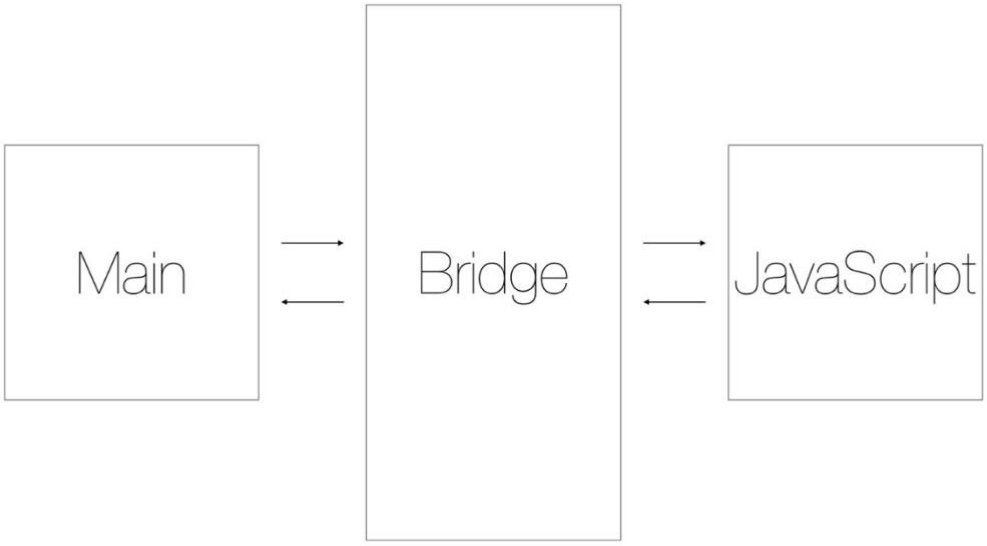


Figure 3. React Native Architecture
[13]

Figure 3 illustrates the architecture of React Native that allows the framework to function.

Because of the architecture of React Native, one downside to using is it is that the communication is not direct and thus more time consuming. This however can be bettered to a degree. React Native offers developers to implement their code on the native side. This means that if one process implemented in JS is considered slow, then the developer can choose to implement it on the native side instead where it might take less time to execute. This means that a process can be implemented in Objective-C and called out on the JavaScript side.

Another downside of React Native is that it is more CPU heavy than truly native applications.

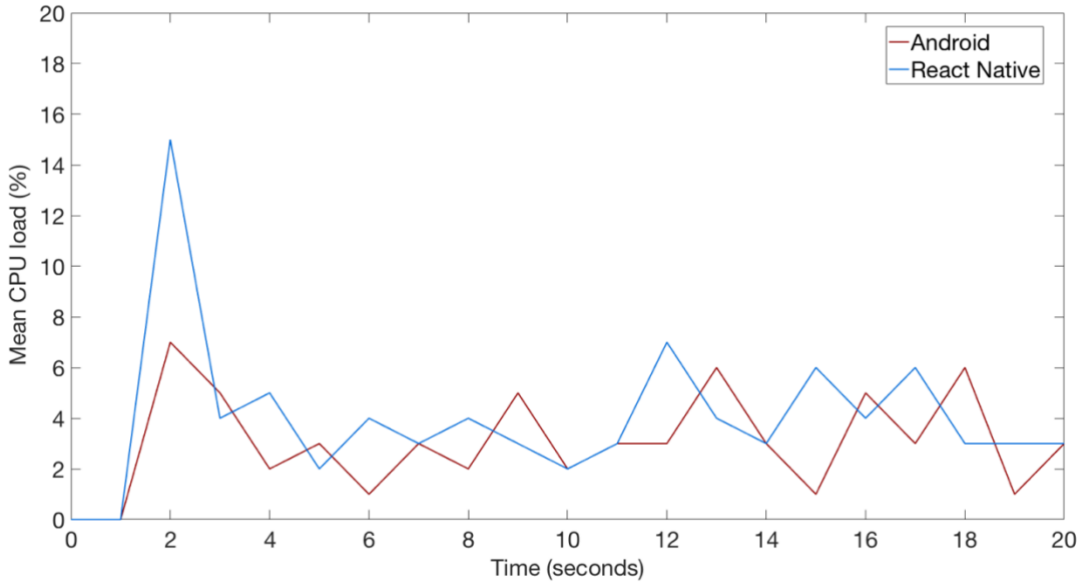


Figure 4. React Native CPU Usage Versus Android

[12]

Figure 4 is illustrating the result of a user interface test performed between an Android application and a React Native application. The test was touching a graph displayed on the application and then stretching it. The results indicate that React Native requires more CPU usage for performing even a simple task. Because of that, creating high performance mobile games and such is out of the question as they would demand too much resources. Since we are creating a sign-in and know your customer application that does not need to handle heavy CPU loads but rather act as a simple interface, then picking React Native is a reasonable choice as it allows us to save the time of developing two applications with a totally different code base.

2.10.2 The stack of the backend

For developing the backend, we are using Node.js as the backend run-time environment. A framework named Hapi for creating the server software and a framework called Sequelize to communicate with the database. The database system that we have chosen is PostgreSQL.

“Node.js is a JavaScript run-time built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. [14]“

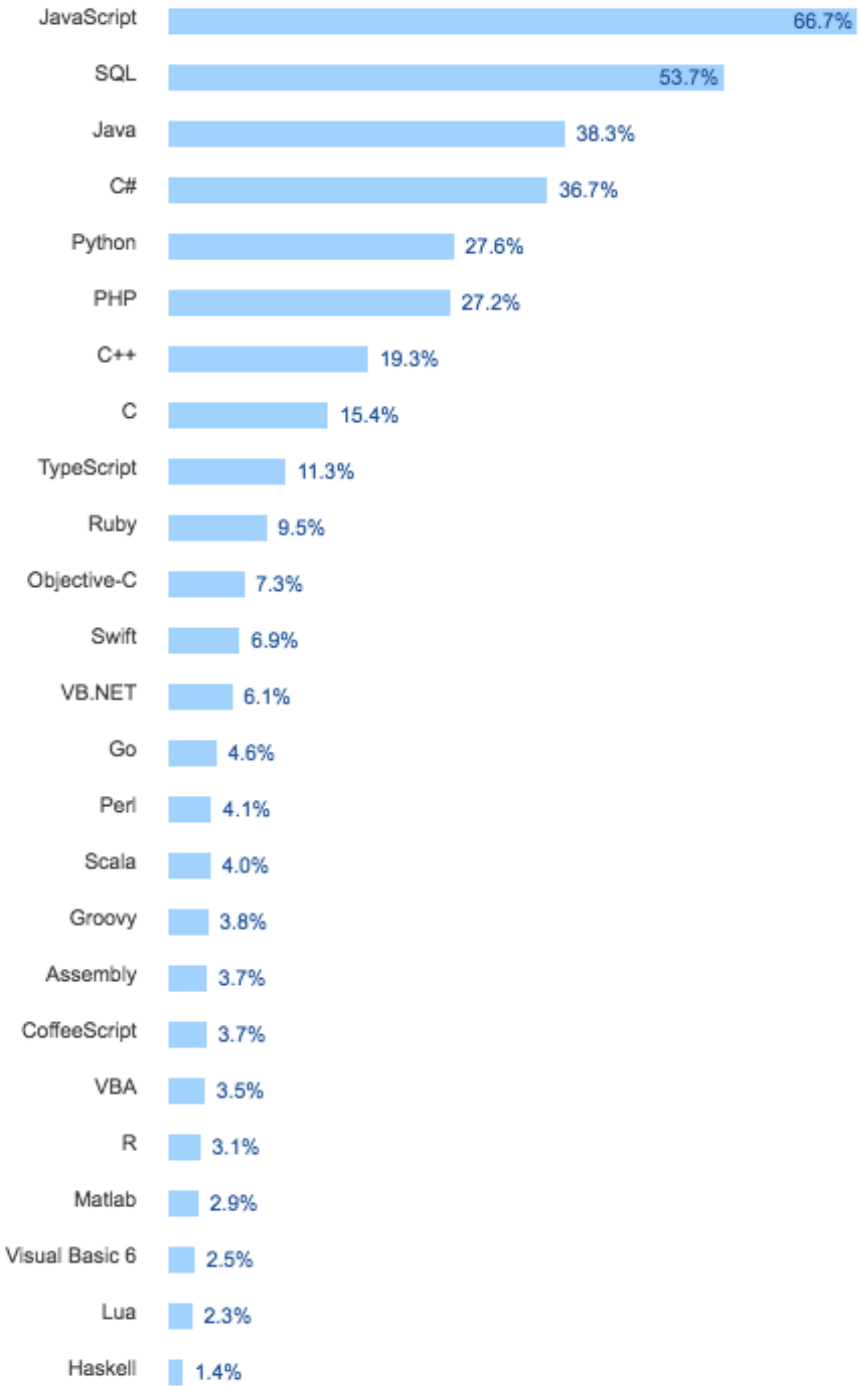
By using Node.js we have created an environment for ourselves where the code written in the backend and in the frontend (mobile application) are in the same programming language, JavaScript. This allows the developers to be more flexible as they can easily move between developing a feature for the frontend to solving a problem in the backend. As a result of this, the code base is more easily understandable for developers who are familiar with JavaScript since they are written in the same programming language.

Another reason that supports the use of Node.js is that the backend does not need to perform CPU-heavy actions which would block incoming requests since Node.js only uses a single CPU core. Our backend acts more like a real-time application and for that, the event-driven, non-blocking Input and Output model of Node.js is excellent [15].

In addition, based on the survey conducted in 2017 by Stack Overflow, which is the largest online community for developers, JavaScript's yearly rise in popularity means that finding people with required knowledge for maintaining the project gets easier [16].

Programming Languages

% of This Category | % of All Respondents | % of Professional Developers



27,612 responses; select all that apply. Shown as a percentage of the respondents who chose at least one language, framework, database, or platform.

For the fifth year in a row, JavaScript was the most commonly used programming language. And once again, SQL takes second place, and Java third. However, the use of Python overtook PHP for the first time in five years.

Figure 5. Most Used Programming Languages of 2017
[16]

Hapi.js is an open source framework for Node.js that allows its users to build applications and services. The idea is to help the developer to focus on writing code instead of spending time on building the infrastructure. With the use of Hapi, we are creating the server and its routes that will later be used to execute business logic [17].

"Sequelize is a promise-based ORM for Node.js v4 and up. It supports the dialects PostgreSQL, MySQL, SQLite and MSSQL and features solid transaction support, relations, read replication and more. [18]"

Sequelize enables us to write queries in JavaScript that improves the readability of the code base for developers with weaker Structured Query Language (SQL) knowledge. In addition, Sequelize provides us with an easy method to connect to the database and allows us to perform raw queries for seriously complex scenarios.

2.10.3 Security and identity

Every application that handles customer data and enables its customers to log in to external or internal systems should be secure. In order to provide this, the first layer of security should be stating that in the production environment all communication should take place over HyperText Transfer Protocol Secure (HTTPS). By enabling HTTPS and configuring it correctly, the data sent back and forth is encrypted and thus offering greater security since the data is now less vulnerable to attacks like the "man-in-the-middle".

To add an extra layer of security and a signing functionality that our application needs to verify its customers, we have resulted in using cryptographic algorithms. The algorithms are used in the mobile application and in the backend.

The algorithm that we favoured was the Rivest–Shamir–Adleman (RSA) algorithm. The reason behind is that firstly, the algorithm provides us with encryption and decryption functionality and with a signing and signature verification functionality [19]. Secondly, RSA is one of the most used and known cryptographic algorithms of today. Due to its popularity, there are countless RSA libraries available and that means less lines of code and faster development.

An alternative route would have been to use different cryptographic algorithms. For example, using the Elliptic Curve Digital Signature Algorithm for signing and signature verification and the Elliptic Curve Integrated Encryption Scheme Algorithm to support the encryption and decryption functionality [19]. Those algorithms are not so well known and because of that libraries are harder to find or would require our own implementation.

2.11 Main processes

The main processes for the mobile application are:

- Creating a customer
- Verifying the customer identity and creating partner customers
- Signing in to partner sites

2.11.1 Overview of creating the customer

The prerequisites for creating a customer are:

- Phone number ownership
- Device ownership

It should be mentioned that both the phone number and the device should be unique. This means that there cannot exist more than one customer that is using a phone number or a device that is registered with the application. This way we can hopefully filter out fake accounts because the effort of creating a customer outweighs the value of owning a fake account.

The first step when creating a customer is verifying your phone number. For that, the user must enter his or her phone number. After this event, the user receives a Short Message Service (SMS) message containing the code required for proving the ownership of that phone number.

When the inserted phone number was successfully verified, the user is asked to choose a Personal Identification Number (PIN) and the customer is created. If the device supports fingerprint scanning and validation, the user is also asked if he or she wishes to enable logging in with the fingerprint option.

2.11.2 Verifying the customer identity and creating partner customers

If the customer entity was created, the user now has an option to create partner customer entities. Partner customers, as mentioned above, are entities linked to a specific gambling site and to the customer entity. One customer can own multiple partner customers.

There are essentially two verification standards. The first tier is input data, provided by the user. The second tier is data gathered from the uploaded national identification documents. Namely, the user has an option to upload his or her identification documents in order to verify his or her identity – this process is also named the Document KYC process. The identity verification process is usually done only once and may take some time to finish since the identity verification is not automated, but looked at by real human beings. Once the second-tier identification is done, it overrides the previous tier data if they contradict.

When a customer entity exists then the user of the application will be presented with a list of gambling sites. All of those sites might require some sort of data in order to create a partner customer entity linked to those sites. If the data asked by the gambling site is already linked to the customer, then all that the customer needs to do is to press a button since the data that is required is already known. If there is no data, the customer needs to provide it. When the user has submitted the required data to the gambling site it is then verified on the gambling site side and a response is sent back to the mobile application’s backend, determining if the partner customer creation process was successful or not.

2.11.3 Signing in to partner sites

If a partner customer exists and is verified, then the sign-in option will be enabled. The user of the application can now sign in to the gambling site from the application by simply pressing the login button or sign in from the gambling site by using the push notification service. This means that ideally every integrated gambling site will have the application's logo near the login section of their site that would hint the customers that such possibility exists.

3 Results

3.1 Implementation of the customer creation process

One could argue that the customer creation process is the most important process of our mobile application. The argument supporting this statement is that this process is the first experience that the user will have with the application. It is common knowledge that first expressions have a great influence on liking something or someone. Because of this, a lot of effort went into making the customer creation process as automated as possible and the application itself pleasing for the eye.

3.1.1 Determining the user's location and phone number prefix

Since one of the core ideas of this application is to offer great user experience, we want all of the processes to be as automated as possible. The first encounter of doing so was determining the user's location in order to decide the prefix of the user's phone number. The prefix is the code representing the country of where the phone number is used. In case of Estonia the value of the prefix is +372.

Figure 6. Welcome Page of the Mobile Application

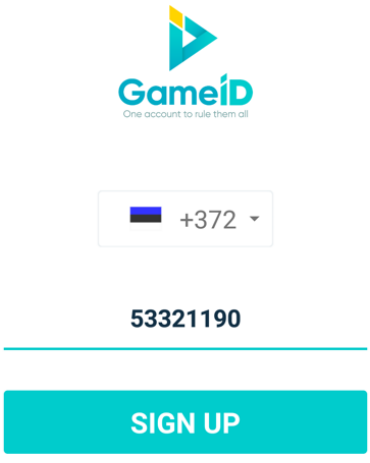


Figure 7. Phone Number Input

Establishing the user's location takes place during the transition of the scenes shown as Figure 6 and Figure 7

One way of determening the user's location is by using the Global Positioning System (GPS). This seemed like a good idea and was also implemented. Soon a downside was discovered to this approach. The problem was that using the GPS service in an application requires the owner of the device to grant extra permission which could be unacceptable for some users. This is especially relevant in this day and age where awarness of applications collecting data has risen.

The final implementation was determening the location based on the Internet Protocol (IP) address of the application's user. This can be done because we realised that for determining the prefix we do not need an accurate reading of the phone owner's location. The problem with this method is that the user must have internet access and should not use a Proxy server or a Virtual Private Network service since in that case the IP address might refer to a completely different geographic area rendering the effect useless.

To get the user's location, we are making an HTTPS GET request to a service capable of determening the location based on the IP address. This happens during the transistion to the next scene. The transition invokes a JavaScript method named preload.

The preload method calls out an HTTPS request and returns the payload of the response which is in JavaScript Object Notation (JSON) format. The payload contains latitude and longitude values and a country code. Those values are stored in state variables of React Native.

To find the prefix compliant with the country code, we are using a Node Package Manager (NPM) library called "libphonenumber-js". The library provides us with functions smart enough that are able to check for a correct phone number format for the majority of the world countries and also provides a function that takes in the country code as a parameter and returns the prefix in correlation with the country code provided. By using the last described function we are able to determine the correct prefix for the phone number [20].

If the operating system of a device is Android, then the preload function will also call out a native method that tries to read the phone number from the Subscriber Identity Module (SIM) card. This feature is only available for Android devices since it would violate Apple Inc. agreements. Apple is creating a container for each application that limits the access to read information from the SIM card [21].

3.1.2 Phone number verification

As mentioned in the overview section above, the user has to verify his or her phone number before he or she is allowed to create an account. To provide our users a service capable of doing so, we are using the help of Sinch. Sinch has a developed a special library for React Native that greatly simplifies the phone number verification process.

When the phone number with the correct prefix is determined, the user is presented with an option to press the "Sign up" button. This action triggers an event that calls a function that will make a POST request to our backend. The backend is then responsible for checking if the number in the payload of the request is already used by another user of the application. If it is, then an error response is sent back and the user has to use another number. However, if the number is free, the backend is marking it with a state of *verifying*.

When the response sent back to the mobile application from our backend was marked as *verifying*, then the mobile application will trigger Sinch's library function that will tell Sinch to send out a verification code via the SMS to the number specified as the input parameter of that library function. The user is now responsible for reading the 5-digit code from the received text message and needs to insert it into the application. This event calls out another library function that sends the verification code to Sinch to get verified. Sinch now determines if the entered code was valid and makes a POST request to a Uniform Resource Locator (URL) specified in the Sinch's admin panel. The URL points to our backend's endpoint that decides based on Sinch's message if the phone number verification was successful or not. If it was successful, the phone number's state in the database is marked as *verified* and the user can continue with the customer creation process.

In case the user's device is running on Android then the phone number verification process will be automatic. Sinch has implemented the first mentioned library function in a way that it is capable of reading the content of the SMS message and automatically sends out a request to their backend with the sent code in the message. The process is shown in Figure 8.

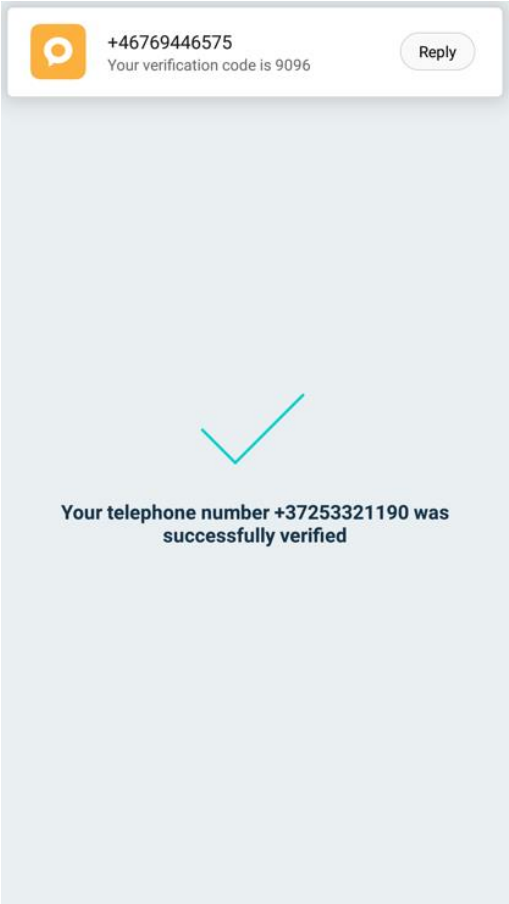


Figure 8. Successful Phone Number Verification

3.1.3 Generating the RSA key pair for the customer

To provide the user with the functionalities described in the methodology section, we need to generate a RSA key pair as this key pair is used to encrypt and decrypt sensitive messages exchanged between the backend and the customer. The key pair is generated after the user has provided the PIN (Figure 9) that is later used to perform some of the described functionalities.

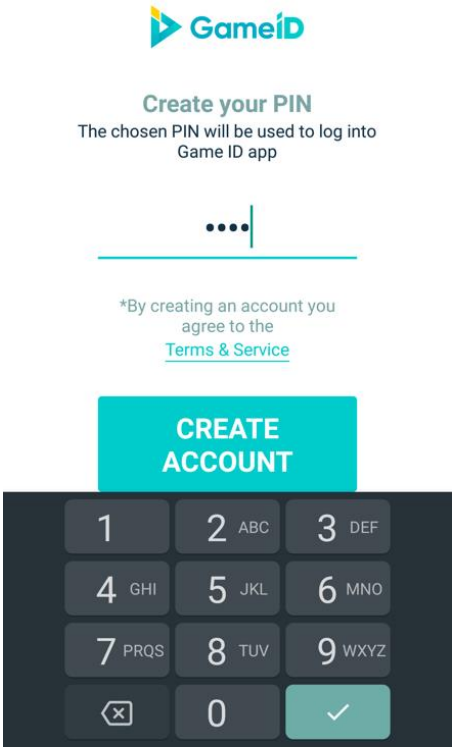


Figure 9. PIN Selection

For generating the key pair, we are using a NPM library called node-forge. The library provides us with functions that allow us to perform actions possible with the RSA cryptosystem. In addition, it also provides us with hashing algorithms that need to be used to perform some of the possible RSA functions [22].

Choosing the correct RSA key size holds great importance. The correct RSA key size determines the security and toughness of many functions performed with the RSA keys. The key size will also influence the length of data that can be encrypted.

Based on the recommendations of IBM the medium-strength RSA key size starts from 1024 bits. Key size of 2048 bits is considered high-strength and 4096 very high-strength keys [23].

For the RSA key pair of the customer entity we have chosen 2048 bits. According to publicly available literature this should give us a strong and secure enough encryption. A key 2048 bits long allows us to encrypt data up to 256 bytes [23].

3.1.4 Creating the customer

Once the phone number and PIN code are available, we are ready to create the customer. This process consists of three API calls to the backend. The first sub-process is creating customer credentials. This process is generating the RSA key pair that was analysed above. When the key pair is generated the signature is created by using the RSA private key of the customer.

To create the customer's signature, we are using the library function of node-forge. This function is provided with the public key that was received when the RSA key pair was generated. The public key will act as the data that is being hashed. RSA signature generation is essentially encrypting the hash with the private key. Before doing so the data is hashed with the Secure Hash Algorithm 256 (SHA-256) algorithm [19].

When the customer's signature is created, the public key of the customer is sent to the backend via a POST request. The endpoint that receives the data checks by using the node-forge library signature verification function if the public key is indeed the right public key. If the signature is verified by the function then the sent public key and other gathered data are inserted into the database. The public key is later used to encrypt sensitive messages directed to the customer.

The second sub-process consists of transferring the PIN and other data that was gathered from the process into the database. Before doing so, the PIN code is hashed by using the SHA-256 algorithm as it is considered bad practice to send such information as plain text. For the salt of the hashing function we are using the Universally Unique Identifier (UUID) version 4. This means that the data what is hashed is PIN plus the generated UUID value. Because of the uniqueness factor of UUID, it is likely that every customer's PIN has a unique salt, making it harder for the hacker to determine the value of the PIN as the hacker cannot use the same salt for multiple customers.

This data is then put into a container on the mobile application's side and encrypted with the backend's public RSA key. The encrypted data is then sent to the backend via a POST request. The backend will then decrypt the container with the backend's private key and store the sent data in the database.

The third sub-process is storing the customer's RSA keys. The general idea of the RSA private and public key is that the private key should be known only to the owner of the key pair. This way we can actually assume that the signature belongs to this specific person and not to someone else. Because of this, we have decided to store the private key and the public key in to the storage of the device. This is a security risk as the user might not be knowledgeable enough to avoid exposing the private key to parties with a malicious intent.

3.2 Verifying the customer's identity

Every gambling site might have different requirements in terms on the data they need for creating an account for their site. To enable gambling sites to configure this data, we have implemented the partner customer logic. Whenever the user registers to a new gambling site he or she is creating a partner customer entity. Every partner customer entity is linked to a certain set of data that is determined by the gambling site and often overlaps. For example, a customer might have two partner customers entities linked to it. Data linked with the first one contains: email, name, sex etc. and the other one might contain: name, sex, promotion code. The important part is to notice that the value of name for the first partner customer might not equal the value of the second partner customer's name field. This rule changes when the customer has been verified via the Document KYC process which sets the final values for gathered information fields.

As the Document KYC process is the highest form of verification, it is assumed that the data gathered from that process is superior to the data inserted by the user. That is why the user is not allowed to insert values for fields that have become known from the Document KYC process.

3.2.1 Document Know You Customer process

The Document KYC process is initiated when the customer is trying to create a partner customer account to a partner gambling site that has asked us to verify the user on that level.

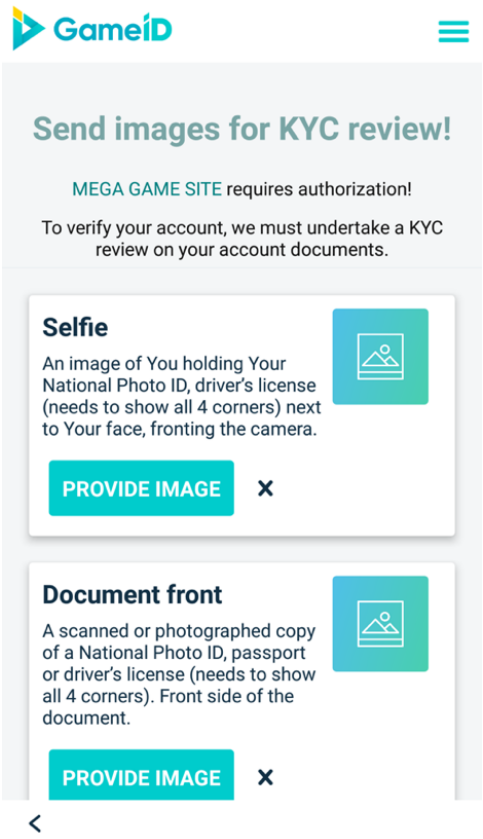


Figure 10. Start of the Identity Verification Process

In order to successfully complete the Document KYC process, the user is required to upload three images for review. These images are: an image of the customer holding an official document next to his or her face, an image of the document's back and an image of the document's front. To provide these images the user has two options. The first option is picking the pictures out from the phone's storage via the gallery application and the second one is to take them by using the device's camera.

The NPM package that is providing us the gallery browsing and the camera function is named react-native-image-crop-picker. When the images are photographed, we are storing them inside the device by using another NPM package called react-native-simple-store. When all of the three images have been provided, the user is asked to send them out for review.

The Document KYC process consists of three API calls to the backend. First of them being a POST request to a specific endpoint of the backend. This endpoint is responsible for creating a Document KYC instance linked to the customer who initiated it. This is mainly done to keep track of the process. The states representing the process status are: *init*, *pending* and *confirmed*. The first API call will mark the state as *init*. This is used to avoid the scenario, where the user has taken two pictures out of three and then exits the application as this would restart the process.

The second API call is responsible for sending the images one by one using a POST request with the content type marked as multipart/form-data. The backend's endpoint receiving this data is then uploading the sent images to the Amazon S3 bucket and storing the image name and the image extension into the database.

The third and last API call is updating the state of the Document KYC process linked to the customer. If all of the three images were uploaded successfully, then the state of the process is marked as *pending*.

The final step is completed by a human being. The images uploaded by the customer are displayed in the system's administration panel. The administrator will then review the documents and decide whether the verification process was successful or a failure. Depending on the outcome, the administrator is then updating the state of the Document KYC verification instance.

3.3 Transactions

In order to manage creating accounts and to support other use cases of the partner customer entity, we have implemented the transaction logic. Every transaction can be with a following type:

- App_register
- Site_register
- App_login
- Site_login

- Unlink

The type of the transaction instance refers to the action that the user, in control of the partner customer entity, is performing. An illustrative example of this is creating an account to a gambling site from the application. This means that a transaction with the type of *app_register* is created. Each party that plays a role in performing this action has to sign the transaction instance to prove that the action is wanted by everyone. When every party has signed the transaction and the signatures are validated, the transaction is marked as *confirmed*. The usual parties involved are: The platform, gambling site (partner site) and the user (partner customer).

3.4 Creating the partner customer entity

When all of the information requirements set by the gambling site are satisfied, the process of creating the partner customer entity can be started.

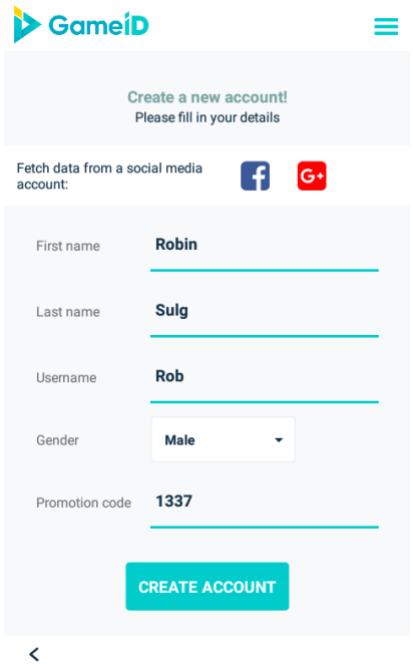


Figure 11. User Information Input

The user interface for the gambler to provide the required information is shown in Figure 11. Once the user has submitted the required data, it is then sent to the backend. The backend is creating the transaction instance with the type of *app_register*.

While creating the transaction, the backend is also adding a signature behalf of the platform and sends out a notification to the partner site asking for its signature. When

this request is successfully resolved, a new RSA key pair, linked to the partner customer, is generated and stored into the memory of the device. This newly generated key pair is now the certificate that proves the ownership of the created partner customer entity. With the generated key pair, the user is sending the partner customer's signature to the backend which adds it to the transaction. In order to complete the partner customer creation process, the partner site (gambling site) has to make a POST request to the backend with its signature. If all of the three signatures are validated, the partner customer entity is linked with the partner site (gambling site).

3.5 Signing in to gambling sites

By creating the partner customer, the user is now able to log in to the gambling site that the partner customer entity was linked with. As described in the functional requirements, the user is allowed to sign in by using two different methods.

3.5.1 Singing in from the mobile application

The sign-in process from the application is using the same transaction logic as it was described above. By initiating this process, a transaction instance is created with the type of *app_login*. The state: *app_login* refers to the method how the sign in process was initiated. In this case it was from the mobile application, hence the name of the type.

The mobile application displays the user a list of gambling sites (Figure 12) that have been integrated with the platform. If there is an active partner customer linked to any of those sites, the user is presented with an option to log in to the specified gambling site.

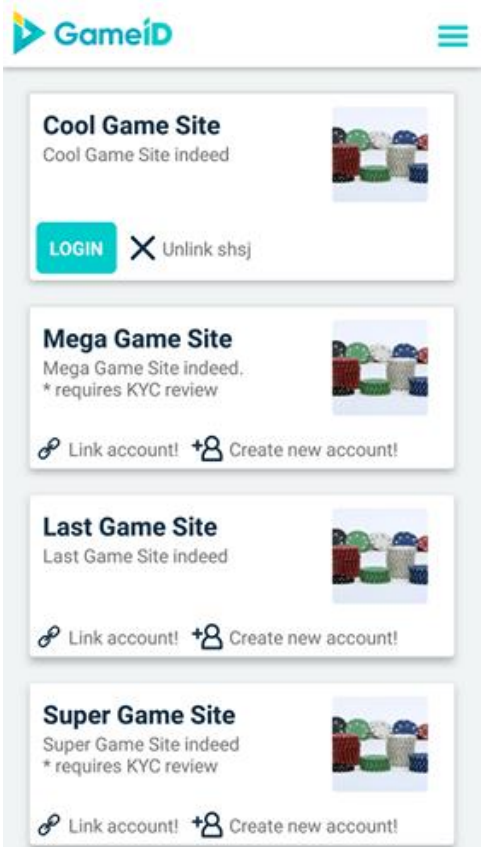


Figure 12. Lobby of the Mobile Application

When the sign-in process is initiated, a request is sent out to the backend that creates a transaction instance into the database with the type of *app_login* and adds the signature of the platform to it. When the response from this action is sent back to the mobile application, the partner customer's signature is added to the transaction. After the previous request is resolved, the URL specified by the gambling site is opened from the application. This event lets the gambling site know that the partner customer wishes to sign the transaction linked with the sign-in (login) event. The gambling site is now responsible for making a POST request to the platform's backend to a specific endpoint with the right credentials. This action will add the gambling site signature to the transaction and if the status of the transaction instance is *confirmed*, the user is logged in

3.5.2 Signing in from the gambling site

The second method for signing in is initiated from the gambling site by specifying the username and the phone number bound to the customer entity. The gambling site is then responsible for making a POST request to the backend with the inserted data. The endpoint that the request was done against is then finding the partner customer entity based on the phone number and the username. With the found data, the backend is creating a transaction, signing it and asking Firebase to send a push notification to the mobile phone. The push notification then triggers a new scene that is presented for the user. The scene is illustrated by Figure 13.

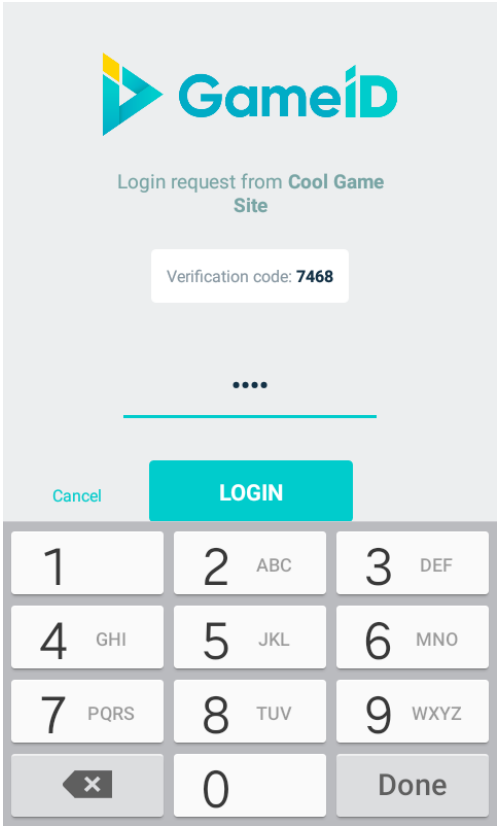


Figure 13. Site Login Pin Confirmation

The mobile application asks the user to enter his or her PIN. If the entered PIN was correct, it triggers a function that makes a POST request to the backend and adds the signature of the partner customer to the transaction. Meanwhile the gambling site is using a technique called polling to validate that the transaction was signed both by the platform and by the partner customer. If the signatures are valid, the gambling site (partner site) will add its signature to the transaction by making another POST request to the platform's backend and the user is allowed to sign in.

4 Analysis

After the mobile application was developed some details occurred that could be carried out better or would require a precise approach to appear credible. Because of this, the following chapter is dedicated for analysing the selected features, models and properties of the mobile application.

4.1 Analysis of the application's architecture

As it was mentioned in the Methodology chapter (2.4), the mobile application and the backend are dependent on third party services. Naturally, having a high level of dependency of something or someone is a bad property to have. Using third party services for the development of our mobile application however was reasonable as the services they provide require a lot of effort and resources to implement. Another reason supporting the made choice is that the most important third-party services that we are using have been developed by major corporations and widely used amongst developers. Because of this, the likelihood of them being deprecated is low.

A more important problem, that one should draw focus on is the cross communication of parties illustrated in Figure 1. The mobile application is interacting with three entities: Firebase, Sinch and the solution's backend. Ideally, one would want the mobile application to communicate with the lowest number of APIs as keeping this number low decreases the complexity of the application for the developer and also proves to be more secure as the application is exchanging data with a verified harmless source.

4.1.1 Sinch

The very first implementation of the SMS service was done on the backend's side. The mobile application was only responsible for providing the backend with the phone number and the verification code sent by Sinch. The backend then would handle all communications with Sinch and forwarded the mobile application information on how it should act. However, to offer the Android users a more comfortable way to verify their

phone number, the phone number verification process was implemented by using the Sinch's SDK for React Native.

With the second implementation, the verification process is far less understandable for new developers as the information flow is scattered. The SMS verification process is initiated by using the Sinch's SDK on the mobile application's side. When Sinch receives the initiation message, it notifies our backend with the result of the phone number verification process. The last implementation is far more complex than the first one as the process now relies on Sinch to make a POST request to our backend. Sinch is able to do this by reading the URL marked in their admin panel [24].

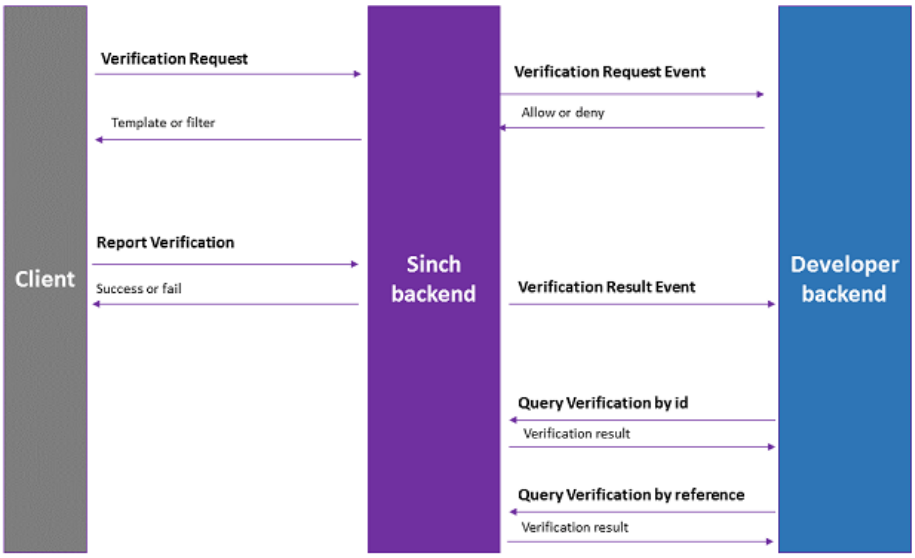


Figure 14. Sinch Phone Number Verification Process

Without proper documentation of the process it is very hard for the developer to understand the process flow solely on looking at the program code. The programmer has to deduce this information by reading code from multiple files and endpoints.

The last implementation was still favoured because Sinch is claiming that by implementing the phone number verification process in the described manner has better

security. The guidelines for the phone number verification process are illustrated in Figure 14 [24].

4.1.2 Firebase Cloud Messaging

Firebase Cloud Messaging (Google Firebase) is another third service party that is communicating directly with the mobile application. Because of the manner that the Firebase documentation asks the developer to implement the push notification service, the device needs to register itself as illustrated in Figure 15:

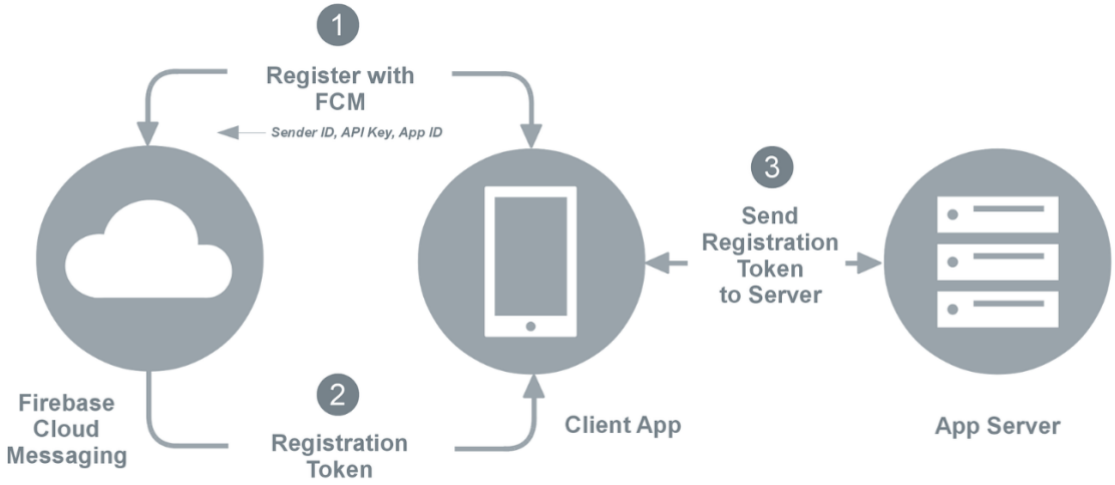


Figure 15. Google Firebase Registration Process

[7]

Google Firebase has implemented a common platform that enables developers to use push notifications more easily as there are certain differences for sending push notifications for iOS devices versus Android devices. Because of the nature of Firebase, the mobile application needs to directly communicate with the service. The device needs to exchange the device token with Firebase. The device token is a special code used to identify devices. Based on that token the service is capable of knowing to who does it need to send the push notification. The device token is also stored in the backend of our application. The token needs to be stored in order to initiate the push notification service from the backend. Doing so allows the backend to ask Firebase to send out a push notification to a specific customer that is using the partner customer entity to use the site login option or to notify the customer with a message.

When the user initiates the sign in process from the gambling site, the flow is in the following order:

1. The user provides his or her username and phone number
2. The backend finds the customer entity based on the inserted values
3. The backend is requesting Firebase to send a push notification to the device and provides the device token that the device has used to register with Firebase
4. Firebase sends the push notification to the mobile application
5. The logic for acting on the push notification is handled in the mobile application

This process cannot be implemented in a manner where the information flow is solely between the backend. As it was mentioned above, the device needs to communicate with Google Firebase in order to exchange the device token and allow the push notification service to function [25].

4.2 Partner customer creation process

One of reasons why this application was developed was to offer the clients of the business managing the gambling sites an easier and a faster method for creating accounts.

By using the partner customer entity, we are able to store the data provided by the user and reuse it. This offers great value when the user is creating accounts to gambling sites with overlapping customer information requirements. The application also supports retrieving user data from the social media platforms: Facebook and Google.

Since the mobile application is not yet used by real working gambling sites, the Figure 11 illustrates the simplification of a customer creation process by using a demo gambling site.

The values of the fields “first name”, “last name” and “gender” are already known to the application since the customer has previously gone through a customer creation process, where he or she needed to provide values for those fields. The customer is now creating

an account to another site that also requires the same data. The application then fills those values automatically and the customer is left to provide only the value of the username and the optional field “promotion code”. This is done by defining the most common values that the gambling sites share. The list of the common and shared values is: first name, last name, gender, email, birthdate, national document id, identification code and country. When the values of the list are provided, for example from a previous partner customer creation process (n - 1), then upon initiating a new partner customer process (n) the mobile application is capable of automatically filling the known values. When the customer provides different values during the new process, then for the next customer creation process (n + 1) the values of the previous one is used (n).

There is an exception to this rule that was mentioned above. When the common values have been gathered from a Document KYC instance then the values of those fields cannot be changed anymore.

One could see how reusing the previously provided data can potentially increase the registration speed and requires less effort from the user. Without this application, the user should create the account via the browser and provide the commonly asked values separately for every gambling site that he or she wishes to create an account to.

A bigger difference in terms of comfortability and time consumption is noticeable when the gambling site requires identity verification based on national documents. It is true that the user is not gaining much when going through this process for the first time but the difference comes into play when the gambler has already completed the process and now wishes to create another account to a different gambling site that too requires the same level of verification. As the integrated gambling sites trust the platform, the about to be created account is linked with the existing Document KYC instance and with that, the gambling site is considering the newly created account to be verified on the Document KYC level. Because of this trust between gambling sites and the application, the gambler does not need to submit new identification documents and possibly saves many hours of waiting for his identity to get verified.

4.3 Partner customer sign-in process

The mobile application allows its users to use it for signing in to their accounts from the mobile phone or from the site. The first option is signing in from the application which requires the user to just a press of a button.

The second option for signing in is a bit more complex as it requires the user to initiate the sign-in process from the gambling site and confirm it by entering the PIN into the mobile application as shown in Figure 13.

These two sign-in methods enable the user to sign in to a number of accounts by using the same PIN which is considered to be a more comfortable method than the customary username and password combination. Instead of trying to remember the difficult password for each site, the gambler can rely on remembering only one.

The provided login methods are also safer since the user is:

1. Less likely to end up on a phishing site
2. Not inserting sensitive data into text inputs
3. Signing in from a safe environment (the mobile app.)
4. Using RSA cryptography to verify that the login transaction was correct and successful

4.4 Credibility of the transaction logic

The transaction logic plays a key role in the security and credibility status of the application. As one can see, the transaction logic is used in two of the three main processes. The two processes hold greater importance to the application than any other process combined, since they are involving third parties as a result of the transaction logic. As it was explained in the implementation segment (3.3), each transaction is signed by a party and in order for a successful transaction, all three parties must have signed it. Once all of the three signatures are verified by the backend then the transaction is marked as confirmed, meaning that all of the signatures are valid and the transaction has gone through.

4.4.1 Signature creation for the transaction logic

The signature creation for the transaction logic is a bit different from the customer entity signature creation. For the customer entity, we hashed the entity's public key and then signed it using the RSA library sign function.

The main difference lies in what data is being hashed. The reason why the data that is hashed cannot be the entity's public key is that the hash value would be the same each time and not linked to the transaction in any way. Because of this, one could argue that the signature is not related to the transaction and thus meaningless. In order to avoid those kind of situations, the data that is hashed in a serialized form of the transaction's identifier, the transaction's type, the partner customer's identifier and a 4-digit challenge that is created for each transaction. If this data is signed by an entity then the data clearly points to a certain transaction and can be considered as a credible signature since the data that the signature is based on, contains clear connection to the transaction.

The signature is validated by comparing the hash. Firstly, the serialized data is hashed in the same manner as it was when it was signed. To obtain the hash that it is compared, the signature must first be decrypted using the author's public key [19].

For the entities: partner site, partner customer and the system, the signature creation algorithm follows the following steps when signing transactions:

1. The RSA key pair is generated
2. The transaction data about to be signed is serialized in a certain order
3. The transaction data is hashed using the SHA-256 algorithm
4. The created hash is signed with the RSA private key
5. The value from previous step is encoded into base 64
6. The encoded value is added to the transaction in the database as the signature of the entity

Naturally each entity follows the described actions on their side and transmits the data to the backend of the system for it to be validated and stored.

4.4.2 Signature verification for the transaction logic

The verification of the signatures is done by the backend of the system. The process follows the following steps:

1. The signers public key is retrieved from the database
2. The transaction data is serialized in the same manner as it was done for signing it
3. The transaction data is hashed with the SHA-256 algorithm
4. The base 64 encoded value of the signature is decoded
5. The signature is verified by using the signer's public key

4.4.3 Credibility

The transaction logic provides us with a credible method to implement the functions that the user needs. The credibility of this logic lies in the RSA cryptosystem. The RSA cryptosystem is widely used in many of today's applications.

The transaction logic's integrity is based on the RSA signature and its signature verification capabilities. The RSA signature is considered secure if the key size is at least 2048 bits long [23].

As mentioned above, the requirement for a transaction to be successful is when all three signatures are added and validated. The signature for the transaction is valid since the hash value that was verified with the signer's public key was created from the same data that the hash was created when signing the transaction. If the signature was created from a different value then the hash would also differ and thus the signature verification would fail [19].

Forging a signature is extremely hard and unlikely since the forger has to find the correct transaction details to hash and has to know the signer's private key or try to

deduce it from the public key which requires extreme amount of computing power if even possible. Even finding the public key is a difficult task for the attacker, since it is stored safe in the database which has its own security measures that need to be broken. It should also be mentioned that the attacker also has the possibility to try and obtain all of the three private keys as each one of them is representing a party that signed the transaction.

There are many other ways how an attacker might break the system but this should be in another form than trying to forge the signatures of a transaction and is not in the scope of this thesis.

5 Summary

The following bachelor's thesis was focused on developing and analysing a sign-in and a customer verification mobile application.

The purpose of this thesis was to develop and analyse at least a mobile application's prototype that would satisfy the needs of online gambling sites with similar user information requirements. In order to reach this purpose, the mobile application had to simplify the customer creation process and the sign-in process of those gambling sites.

As a result of this theses a mobile application's prototype was developed, capable of carrying out the features set by the functional and non-functional requirements at the beginning of the thesis. The main processes and logic that allow the mobile application to meet the set requirements where described and analysed. To demonstrate the results of the thesis a demonstration environment was set up.

In addition, the reader of this thesis is presented with an overview and analysis of the core tools and frameworks, third party services and security of the application. All of them used and needed for developing the mobile application's prototype.

The purpose of this thesis was achieved as the mobile application's prototype was successfully developed and analysed in detail.

References

[1] H. Bagdasarian, “Know Your Customer,” [Online]. Available: http://www.identity-theft-awareness.com/know-your-customer.html. [Accessed May 2018].

[2] Gambling Commision, “You need to know your customers - remote casinos,” [Online]. Available: http://www.gamblingcommission.gov.uk/for-gambling-businesses/Compliance/General-compliance/AML/How-to-comply/You-need-to-know-your-customers-remote-casinos.aspx. [Accessed May 2017].

[3] Smart-ID, “Smart-ID,” [Online]. Available: https://www.smart-id.com/about-smart-id/. [Accessed May 2018].

[4] A. Vahtla, “Estonian Public Broadcasting,” 31 January 2018. [Online]. Available: https://news.err.ee/679048/estonia-s-smart-id-more-popular-in-latvia-lithuania. [Accessed May 2018].

[5] C. Suscheck, “A Productivity Comparison of Kanban and Scrum,” 5 October 2011. [Online]. Available: https://www.agileconnection.com/article/productivity-comparison-kanban-and-scrum. [Accessed May 2018].

[6] Sinch, “Who we are,” [Online]. Available: https://www.sinch.com/about-us/company/. [Accessed May 2018].

[7] Firebase, “Firebase Cloud Messaging,” [Online]. Available: https://firebase.google.com/docs/cloud-messaging/. [Accessed May 2018].

[8] Developers Google, “Introduction to Push Notifications,” 9 Aprill 2018. [Online]. Available: https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications. [Accessed May 2018].

[9] Amazon Web Services, “Working with Amazon S3 Buckets,” [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html. [Accessed May 2018].

[10] https://aws.amazon.com/ec2/ , “Amazon EC2,” [Online]. Available: https://aws.amazon.com/ec2/ . [Accessed May 2018].

[11] Southem, 3 October 2016. [Online]. Available: https://medium.com/react-native-development/a-brief-history-of-react-native-aae11f4ca39. [Accessed May 2018].

[12] N. Hansson, “Effects on performance and usability for cross-platform application development using React Native,” 2016.

[13] B. Evkoski, “React Native: What it is and how it works,” 12 June 2017. [Online]. Available: https://wetalkit.xyz/react-native-what-it-is-and-how-it-works-e2182d008f5e. [Accessed May 2018].

[14] Node.js, “Node,” [Online]. Available: https://nodejs.org/en/ . [Accessed May 2018].

[15] J. Rachowicz, “When, How And Why Use Node.js as Your Backend,” 23 February 2017. [Online]. Available: https://www.netguru.co/blog/use-node-js-backend. [Accessed May 2018].

[16] Stack Overflow, “Developer Survey Results 2017,” [Online]. Available: https://insights.stackoverflow.com/survey/2017. [Accessed May 2018].

[17] Hapi.js, “A rich framework for building applications and services,” [Online]. Available: https://hapijs.com/ . [Accessed May 2018].

[18] Sequelize, “Sequelize,” [Online]. Available: http://docs.sequelizejs.com/ .

[19] B. A. Nicholas Jansma, “Performance Comparison of Elliptic Curve and RSA Digital Signatures,” 2004.

[20] Telefónica I+D, “Libphonenumber-js,” [Online]. Available: https://github.com/telefonicaid/libphonenumber-js. [Accessed May 2018].

[21] University of Southern Calefornia, “ITP 342 Mobile App Development,” University of Southern Calefornia, [Online]. Available: http://www-bcf.usc.edu/~trinagre/itp342-20153/lectures/ITP342_DataPersistence.pdf. [Accessed May 2018].

[22] Digital Bazaar, Inc., “A native implementation of TLS in Javascript and tools to write crypto-based and network-heavy webapps,” [Online]. Available: https://github.com/digitalbazaar/forge. [Accessed May 2018].

[23] IBM Corporation, “Size considerations for public and private keys,” [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.icha700/keysizec.htm. [Accessed May 2018].

[24] Sinch, “Verification,” [Online]. Available: https://www.sinch.com/docs/verification/. [Accessed May 2018].

[25] C. D. B. U. Mark McLemore, “Firebase Cloud Messaging,” 3 January 2018. [Online]. Available: https://docs.microsoft.com/en-us/xamarin/android/data-cloud/google-messaging/firebase-cloud-messaging.