

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

5/25/2020

# Bachelor project group 20

Chestnut – Educational PKI Web  
App

Several thin, curved lines in shades of blue and grey originate from the bottom left and sweep upwards and to the right.

Konstantinos Pascal – s315567

Abozar Afzali – s315578



**Institutt for Informasjonsteknologi**

Postadresse: Postboks 4 St. Olavs plass, 0130 Oslo

Besøksadresse: Holbergs plass, Oslo

PROSJEKT NR.

TILGJENGELIGHET

Telefon: 22 45 32 00

# BACHELORPROSJEKT

HOVEDPROSJEKTETS TITTEL  Chestnut – Educational PKI Web App	DATO  25.05.2020
	ANTALL SIDER / BILAG  104
PROSJEKTDeltakere  Konstantinos Pascal – s315567  Abozar Afzali – s315578	INTERN VEILEDER  Thomas Sødning

OPPDRAUGSGIVER  Thomas Sødning	KONTAKTPERSON  Thomas Sødning
--------------------------------------	-------------------------------------

## SAMMENDRAG

This a project by OsloMet's associate professor, Thomas Sødning, that has been completed by Information technology students at OsloMet.

The goal of this project is to create a concept and develop the solution while focusing on the end-users' needs.

Chestnut aims to be a web application that assists students in learning the principles behind PKI.

It offers options for symmetric and asymmetric encryption as well as other utility tools.

3 STIKKORD
REST API
React, SPA
Web Application

# PROJECT PREFACE

This is the report for Chestnut, a web application developed by group 20 as Bachelor project 2020.

Originally, Chestnut was a JavaFX application that Thomas Sødning, who is an associate professor at OsloMet, had developed to assist him in teaching the cryptography concepts and principles to his students. The application had issues that could be solved by converting Chestnut to a modern web application.

When we were seeking projects to do for our bachelor, we were open to all opportunities, but a web application was more desirable. We had built a few web applications before, but we wanted to tackle a bigger project this time. We contacted Thomas about chestnut and scheduled a meeting.

We got to know Chestnut and its advantages and limitations. We also discussed the application's identity as an educational tool. The goal of the project was not only to capture the essence of Chestnut's identity but to emphasize it even more.

We agreed that a web application was the perfect answer to overcoming the limitations that the original Chestnut application had and gave us room for improving all aspects of it.

Special thanks go to Thomas Sødning, who gave us this amazing opportunity and helped us throughout the project, and everyone else that made this possible.

We would like to thank everyone who made this possible. Special thanks go to Thomas Sødning who not only gave us this opportunity but also motivated us and helped us to have a clearer line of thinking.

# Table of Contents

1	Project Presentation .....	7
1.1	Introduction .....	7
1.2	Background .....	8
1.2.1	Cryptography.....	8
1.2.2	Teaching cryptography and PKI .....	9
1.3	Issue .....	12
1.3.1	Portability .....	13
1.3.2	User experience, UI, and accessibility .....	13
1.4	Concept .....	13
1.4.1	Web application .....	14
1.4.2	REST API (Application Public Interface) .....	14
1.4.3	Existing solutions.....	14
1.5	Our solution .....	19
1.5.1	Front end – ReactJS .....	20
1.5.2	Back end – NodeJS .....	21
1.5.3	HATEOAS REST API .....	21
1.6	Conclusion.....	22
2	Process documentation .....	24
2.1	Introduction .....	24
2.2	Documentation tools .....	24
2.2.1	Trello .....	24
2.2.2	<a href="#">Visual Studio Code</a> .....	24
2.2.3	<a href="#">GitHub</a> .....	24
2.2.4	<a href="#">Jitsi Meet</a> .....	25
2.2.5	<a href="#">Adobe XD</a> .....	25
2.3	Software Development Methodology .....	25
2.3.1	Agile frameworks .....	25
2.3.2	Our choice .....	26
2.4	Kanban .....	26
2.4.1	Kanban board .....	26
2.4.2	Work in progress limit .....	28
2.5	Scrum .....	29

2.5.1	Scrum's daily standup meetings .....	29
2.5.2	Scrum Sprints .....	29
2.6	Development Process and methods.....	30
2.6.1	Pair programming .....	30
2.6.2	User-centered design .....	30
2.7	Project requirements .....	31
2.7.1	Functional requirement .....	32
2.7.2	Non-functional requirement (NFR) .....	33
2.8	Backlog.....	34
2.9	Project Phases .....	35
2.9.1	Planning Phase .....	35
2.9.2	Development phase .....	37
2.9.3	API Design .....	37
2.10	The big issue.....	40
2.11	Our solution .....	40
2.12	Frontend development .....	41
2.13	Documentation .....	41
2.14	Reflections .....	41
2.15	Conclusion.....	42
3	Backend documentation .....	44
3.1	Introduction .....	44
3.2	Technologies .....	44
3.2.1	Node.....	44
3.2.2	Express .....	44
3.2.3	express-hateoas-links.....	44
3.2.4	body-parser .....	44
3.2.5	mysql2.....	44
3.2.6	Sequelize .....	45
3.2.7	JSON Web Tokens (JWT).....	45
3.2.8	bcrypt .....	45
3.3	API.....	45
3.4	REST API .....	47
3.4.1	Richardson Maturity Model .....	49
3.5	REST API constraints.....	51
3.5.1	Uniform interface.....	51

3.5.2	Client-server .....	53
3.5.3	Stateless .....	53
3.5.4	Cache ability .....	54
3.5.5	Layered system .....	55
3.5.6	Code on demand .....	55
3.6	Our solution .....	55
3.6.1	Routes and controllers .....	56
3.6.2	HTTP status codes .....	58
3.7	Token middleware .....	59
3.8	Database models.....	60
3.9	API documentation .....	62
3.9.1	OpenAPI specification .....	62
3.9.2	Swagger.....	62
3.10	Conclusion.....	64
4	Frontend documentation.....	64
4.1	Describe what frontend is and what does it consist of?.....	65
4.2	Technology choices .....	65
4.2.1	Frontend framework .....	65
4.2.2	React .....	65
4.2.3	Creating new React app .....	67
4.2.4	Additional packages .....	68
4.2.5	CSS Framework .....	69
4.3	Design Choices .....	71
4.3.1	Color theory .....	71
4.3.2	Navigation Bar .....	73
4.3.3	Sub-navbars.....	75
4.3.4	Icons.....	75
4.3.5	Buttons.....	76
4.3.6	Tooltips .....	78
4.4	Chestnut Web Application .....	79
4.4.1	Landing Page .....	80
4.4.2	Homepage.....	82
4.4.3	Manage Keys .....	82
4.4.4	Search Users.....	88
4.4.5	Asymmetric Encryption .....	88

4.4.6	Symmetric Encryption .....	90
4.4.7	Utilities .....	92
4.5	Testing.....	95
4.5.1	Unit testing.....	95
4.5.2	Integration testing.....	96
5	Further development .....	98
5.1	Email verification .....	98
5.2	Generate different types of keys.....	98
5.3	Cipher methods for AES encryption .....	99
5.4	Conclusion.....	99
6	Bibliography .....	99
7	Table of figures .....	102

## 1 Project Presentation

### 1.1 Introduction

The presentation chapter of the thesis aims to give the reader an introduction to what this report and Chestnut are all about. We will not be going too much into detail or discuss any technicalities right now, as that is not the purpose of this chapter.

First, we will be discussing the background which prompted the development of Chestnut, the original JavaFX implementation of the application. We will also be looking into the issues that have arisen from that implementation and how we have arrived at our earliest concepts and solved those issues.



Moving on from the presented concept, we will be presenting an overview of our proposed solution and wrapping the chapter up with a conclusion discussing our thoughts and reflections on both the application and process.

## 1.2 Background

### 1.2.1 Cryptography

Cryptography deals with a set of methods that enable us to store and transmit information while safeguarding it from intruders. We can use cryptographic methods to keep information private and to communicate in a way such that only the intended recipient can read the message. (CommonLounge.com, 2018).

This definition is very broad as cryptography is a massive field with numerous applications both on the web and in the real world. The easiest way to understand cryptography is to think about it as a way of preserving the confidentiality, integrity, and availability of data. These 3 principles are also known as the CIA triad (Fruhlinger, 2020).



*Figure 1-1 Three principles known as the CIA triad*

(Henry, 2020)

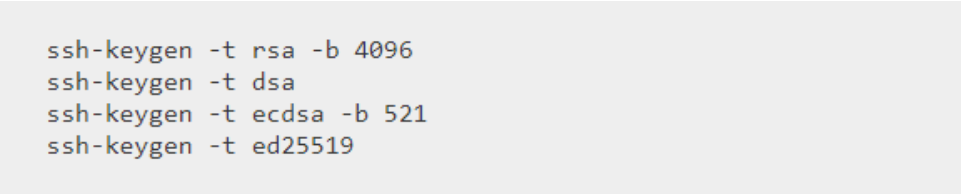
We have given this definition to show that while cryptography can be understood by most people on a surface level, it is a highly complex field with many moving parts and components. For students without prior experience or exposure, learning its concepts can be difficult, especially when the teaching methods themselves are not adequate.

### 1.2.2 Teaching cryptography and PKI

Thomas Sødning, our assignment giver and supervisor, is an associate professor at OsloMet at the Department of Library and Information Science that has had the chance to teach his students some of the fundamental principles behind PKI and cryptography.

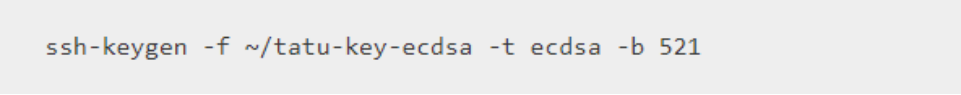
At first, he was using the integrated shell terminal that is present on UNIX based operating systems such as macOS and Linux. The terminal comes with various commands such as *ssh-keygen* that will generate keys and/or keypairs for the user.

Some examples of those commands:



```
ssh-keygen -t rsa -b 4096
ssh-keygen -t dsa
ssh-keygen -t ecdsa -b 521
ssh-keygen -t ed25519
```

*Figure 1-2 Figure of ssh-keygen commands*



```
ssh-keygen -f ~/tatu-key-ecdsa -t ecdsa -b 521
```

*Figure 1-3 Figure of ssh-keygen command with additional options*

The first image shows how the user may choose the type of key being generated by using the `-t` option and specify its size in bits with the `-b` option.

The second image makes use of another option, namely `-f`, which is the file location where the generated key/key pair will be sent to.

```

yourusername@ubuntu1804:~$ ssh-keygen -t rsa -b 4096 -C "your_email@domain.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/yourusername/.ssh/id_rsa):
Created directory '/home/yourusername/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/yourusername/.ssh/id_rsa.
Your public key has been saved in /home/yourusername/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:2zTTG82cFN19x69SJUc00VdNDQcd5HGVy4c8t6V6mjg your_email@domain.com
The key's randomart image is:
+---[RSA 4096]-----+
|           +%/|
|           +@/|
|          .+*X|
|          .==+|
|         S + o.==+|
|        + o.o.o. |
|       . . .o   |
|       E....   |
|       ..oo    |
+-----[SHA256]-----+

```

Figure 1-4 Result output from ssh-keygen command

This is that same command being used in an actual terminal, together with the resulting output of the command.

It is quite clear that an environment such as the terminal is inadequate for learning purposes. It is not user-friendly or intuitive at all and requires prior experience with it and knowledge of all the different commands. This level of experience and knowledge is something that most Library and Information Science students do not have, and as such the terminal is an extremely hard tool to use.

Thomas has echoed this exact sentiment to us, saying that many of his students were more preoccupied and distracted learning the fundamentals of the terminal rather than those of PKI.

Another issue with the terminal is that this specific shell is only available on UNIX like operating systems. Windows-based machines will not have the same shell, making it harder for both Thomas to teach and the students to follow along.

These issues have led to the abandoning of the command line and similar tools for teaching PKI.

To combat those issues, Thomas developed the first implementation of Chestnut using Java and JavaFX. JavaFX is a set of media packages and graphics for Java but most widely known as a platform for creating and delivering desktop applications.

This implementation can be found here: <https://gitlab.com/OsloMet-ABI/chestnut>

Some screenshots of Chestnut as a desktop application:

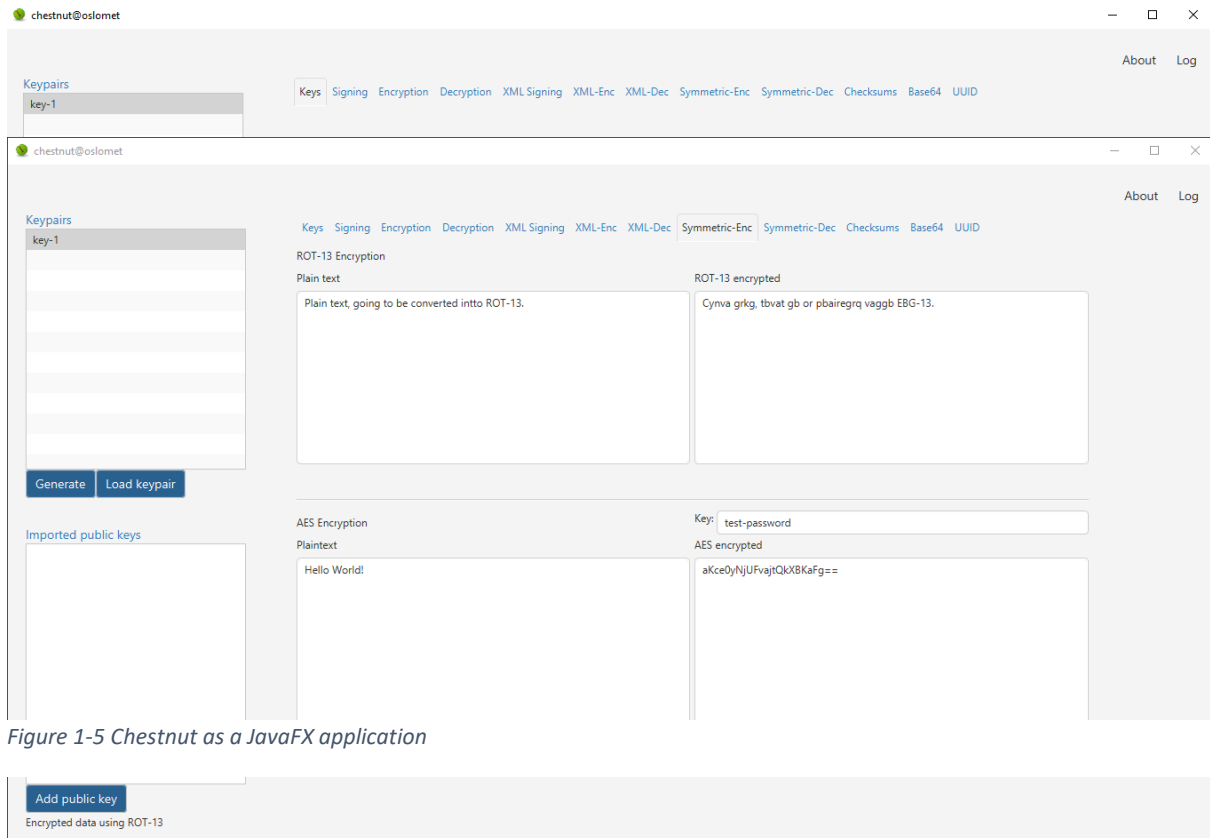


Figure 1-5 Chestnut as a JavaFX application

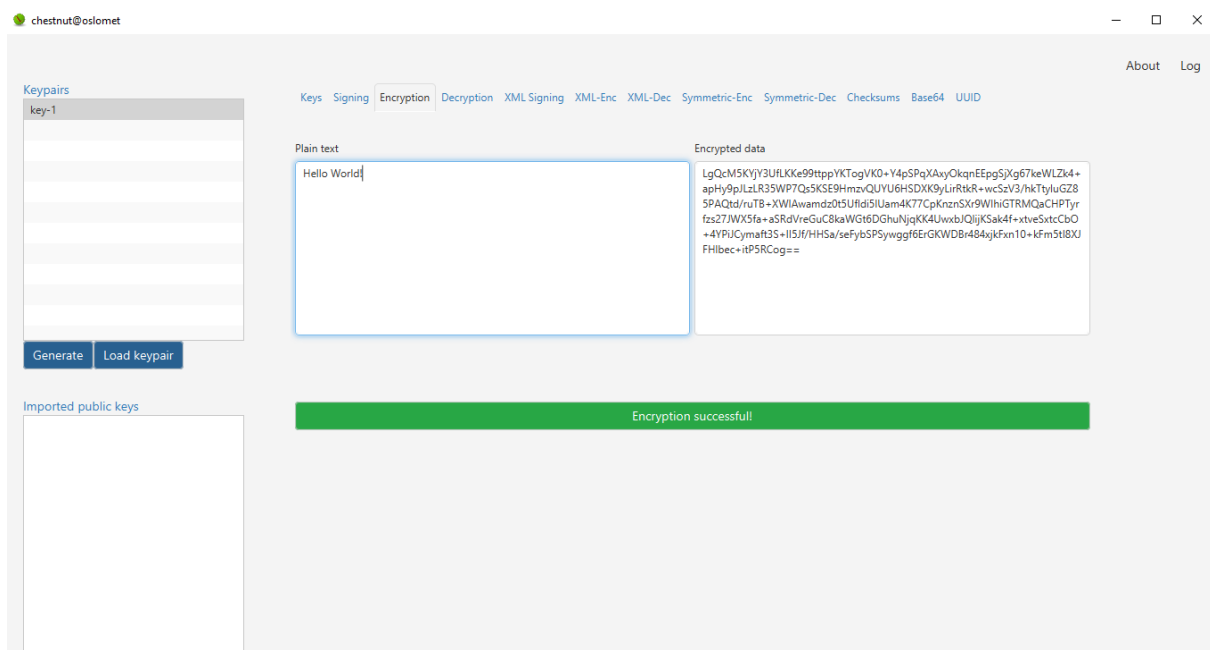


Figure 1-6 Chestnut as a JavaFX application (Symmetric Enc)

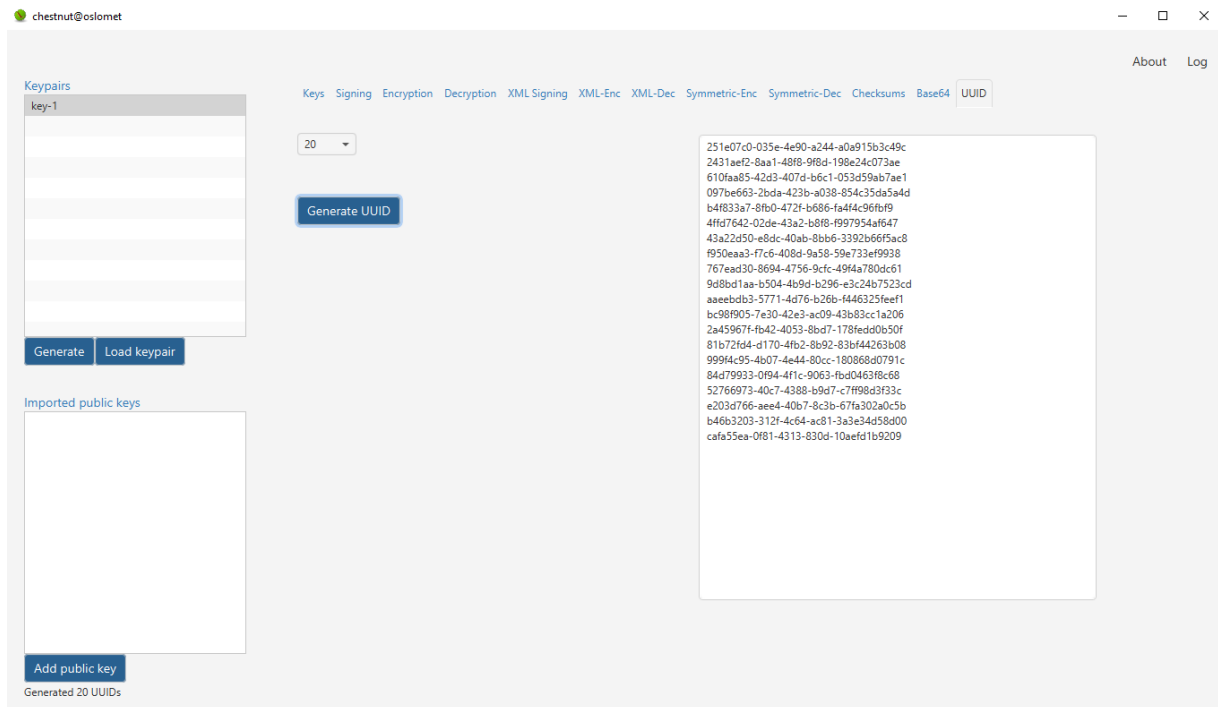


Figure 1-7 Chestnut as a JavaFX application (UUID)

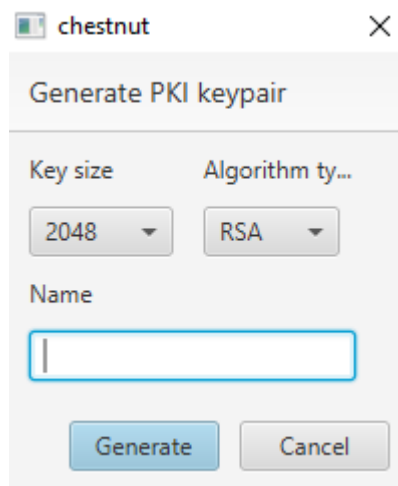


Figure 1-8 Chestnut as a JavaFX application (Generate keypair)

Chestnut as a JavaFX application was a huge step forward from the terminal. Not only is it more user-friendly because of the user interface, but it also allowed Thomas to only include what he thinks is necessary for his teaching.

### 1.3 Issue

While Chestnut as a desktop application was vastly superior to the teaching methods of the terminal, Thomas was faced with other issues. What those issues were and how we solved them, we are discussing in the following sections.

### 1.3.1 Portability

Portability means being able to move software from one machine platform to another. It refers to system software or application software that can be recompiled for a different platform or to software that is available for two or more different platforms (PCmag, 2020).

Chestnut as a desktop application meant that every student, or in this case, the user of the application had to manually download and configure Chestnut. While this is a process that in theory should be quick and easy, in practice, it was not.

As the application is Java-based, to compile the program, you would need to have JRE (Java Runtime Environment) installed on your machine as a prerequisite. Once again, Thomas and the students were facing issues with inconsistent behavior of the program across different environments.

### 1.3.2 User experience, UI, and accessibility

As defined by ISO (International Organization for Standardization), user experience (UX) is a person's perceptions and responses resulting from the use and/or anticipated use of a product, system, or service.

Also, according to ISO, User Interface (UI) is the collection of all components of an interactive system (software or hardware) that provide information and control for the user to accomplish specific tasks with the interactive system (ISO, 2010).

Chestnut as a desktop application does have all the necessary and required functionality and does the job it is supposed to. What it is missing is a more pleasant, accessible, and intuitive design for its GUI.

Both Chestnuts layout components and menu, which can be seen in the previous screenshots, are too small and non-resizable, making it both difficult to see and navigate. The navigation menu is also not as intuitive as it could be with too many options present at the same time.

Additionally, utilizing JavaFX as a desktop application meant students lacked portability on mobile and tablet platforms, thus limiting the reach of the application.

Taking all these issues into account, we created a concept of something that we think will not only satisfy all specifications, but also improve upon the User Interface, User Experience, and accessibility.

## 1.4 Concept

### 1.4.1 Web application

We knew at once that both portability and a more appealing design, with enhanced UI, UX, and accessibility, can be achieved by developing a web application. Web development is also the area we have the most experience in with programming and was therefore the perfect choice.

### 1.4.2 REST API (Application Public Interface)

In addition to the website, we were also expected to deliver an API backend for the application, making this a full-stack solution.

This API should be completely decoupled from the front end, making it possible for any other future developer to make use of it externally. This trait of the backend will also allow the easy substitution of our website with any other User Interface without the need of refactoring or change in server-side code.

A more detailed overview of all requirements, both functional and non-functional, and specifications are mentioned further down in the process documentation.

### 1.4.3 Existing solutions

After having settled on the idea of making Chestnut into a web application, we started looking into other, already existing, solutions on the web.

This was done because of two things:

- Seeing if anything similar is already implemented.
- If there are similar applications out there, how are they implemented?

We were successful in finding several solutions ranging from websites to software and online courses. While they are all trying to achieve the same thing, teaching cryptographic methods and concepts, they were all doing it in different ways.

It is also crucial for us to understand our target audience when developing Chestnut. Our target audience mainly consists of library and information science students without the advanced mathematical background that would be required to understand cryptographic concepts on a high level. That is something we have kept in the back of our heads when developing the web application and made sure to never include unnecessary, overly complex language and/or concepts.

#### 1.4.3.1 *Cryptoprograms web application*

Crypto Programs is a site to create and solve classical ciphers online. It can create 61 different classical cipher types and solve 45 cipher types including some used during World Wars 1 and 2 (Crypto Programs, 2019).

The screenshot shows a web application titled "Beaufort". At the top, there is a navigation menu with five items: "Cipher" (which is highlighted), "Description", "Background", "Security", and "About alphabets". Below the menu, the interface is divided into several sections. The "Plaintext:" section has a text input field containing "The quick brown fox jumps over the lazy dog.". A large downward-pointing arrow is positioned below the plaintext field. The "Encrypted text:" section has a text input field containing "Zxy bkliu xbgqe zfn vegpk djpt lro tcsq qwy.". Below this is the "Key:" section with a text input field containing "Secret Key". The "Options:" section contains four checkboxes: "filter whitespace characters", "group 5 characters", "filter non-alphabet characters", and "convert to first alphabet" (all are unchecked), and one checked checkbox: "filter key on alphabet characters". The "Alphabets:" section contains two text input fields for custom alphabets. The first field contains "ABCDEFGHIJKLMNOPQRSTUVWXYZ" and the second contains "abcdefghijklmnopqrstuvwxyz". Both fields have a three-dot menu icon to their right. At the bottom right of the "Alphabets:" section is a button labeled "Add alphabet".

*Figure 1-9 Cryptoprograms' web implementation of the Beaufort cipher*

This is an example of Cryptoprograms' Beaufort cipher implementation. It does a good job of teaching the user how something works interactively while having the clean looking user interface of a modern application.

They also divide their pages into clear subcategories and make use of a navigation menu at the top, intuitively where users would expect it to be.

Cryptoprograms' solution also has a description for each of its ciphers, allowing the user to read about it into more detail if they so desire:



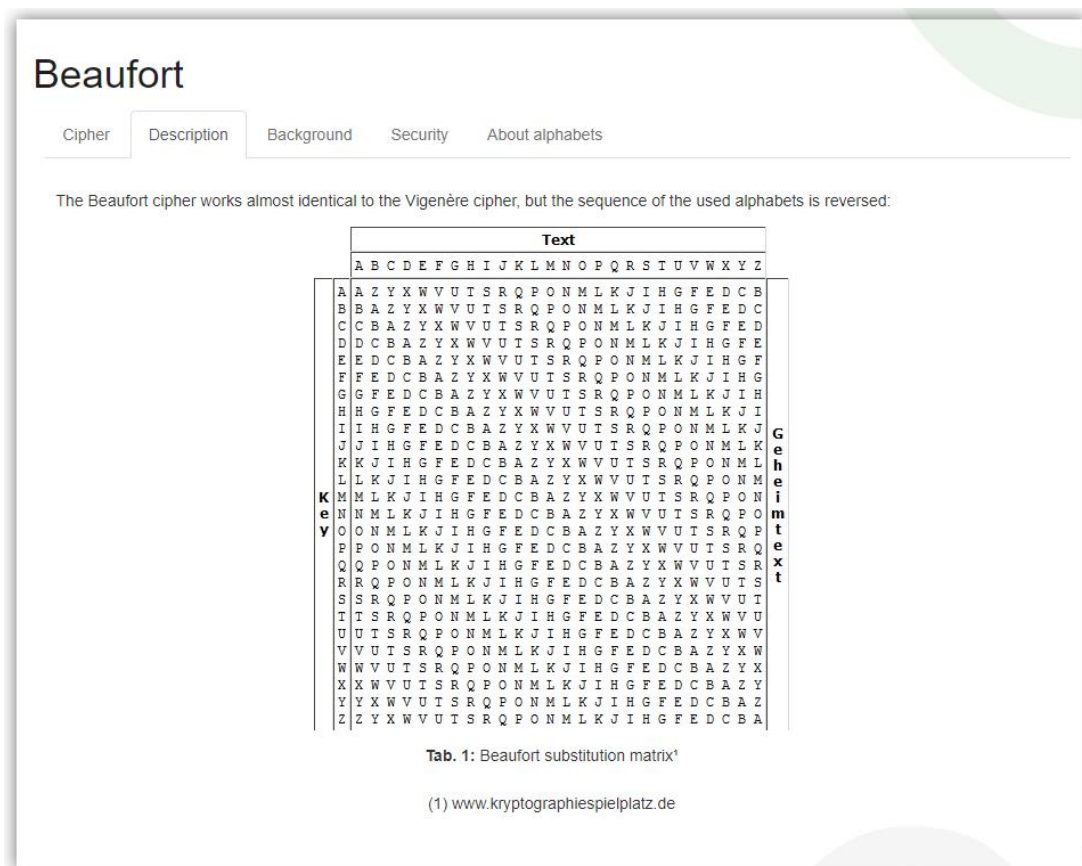


Figure 1-10 Cryptoprograms' Beaufort substitution matrix

There are many things this application is doing right, and we have taken some inspiration from it, especially when it comes to the design and layout. We also knew there are many things we can improve upon such as the actual functionalities.

Cryptoprograms only focuses on ciphers. While we are going to have a few cipher implementations in our application, what sets Chestnut apart will be the possibility for users to register themselves, create, manage, and save their keypairs. Those keypairs will be used and shared with other users to create an interactive environment showcasing how algorithms such as RSA and AES work in practice.

#### 1.4.3.2 CrypTool web application

CrypTool Online (CTO) runs in a browser and provides a variety of encryption and cryptanalysis methods including illustrated examples and tools like a password generator and password meter (CrypTool Online, 2019).

CrypTool is like Cryptoprograms in that it provides many of the same cipher functionalities. What it does differently is having password generators, password strength checkers, and step-by-step explanations with illustrations of how the RSA and AES algorithms work.

These can be seen in the following screenshots:

# RSA

Cipher
Security and References

This module demonstrates step-by-step encryption or decryption with the RSA method. The sender uses the public key of the recipient for encryption; the recipient uses his associated private key to decrypt.

## Prime factors

The security of RSA is based on the fact that it is easy to calculate the product  $n$  of two large primes  $p$  and  $q$ . However, it is very difficult to determine only from the product  $n$  the two primes that yield the product. This decomposition is also called the factorization of  $n$ .

As a starting point for RSA choose two primes  $p$  and  $q$ .

1st prime  $p$

2nd prime  $q$

For the algorithm to work, the two primes must be different.

For demonstration we start with small primes. To make the factorization difficult, the primes must be much larger. Currently, values of  $n$  with several thousand binary digits are used for secure communication.

$n = p \times q$  143 (8 Bit)

## Public key

The product  $n$  is also called module in the RSA method. The public key consists of the module  $n$  and an exponent  $e$ .

$e$

This  $e$  may even be pre-selected and the same for all participants.

Figure 1-12 CryptTool website RSA implementation

# AES (step-by-step)

Cipher
Description
Background
Security

Inspect the encryption of AES step by step. Tap on each byte to see the bytes it depends on.

Configuration
AES-256

AES Variants and Test Vectors

Number of Rounds: 14
- +

S-Box

Permutation

Chaining:
None CBC ECB

Initial Vector (CBC only)

Key
00010203 04050607 08090a0b 0c0d0e0f 10111213 14151617 18191a1b 1c1d1e1f

Expanded Key

Input
00112233 44556677 8899aabb ccddeeff

Encoding Rounds

Round 2

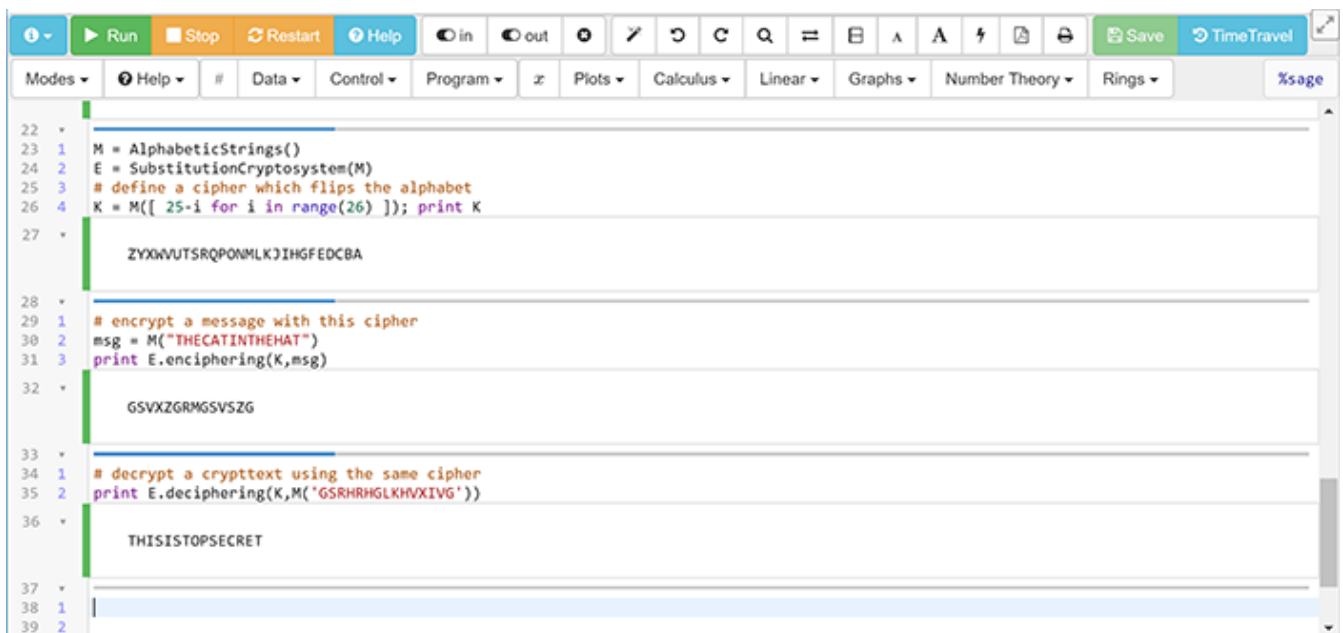
Figure 1-11 CryptTool website AES implementation

RSA and AES encryption/decryption are algorithms that ultimately would be implemented in Chestnut. They are part of the requirements and will be present in the application.

While this application is more complete than the first one, it is still missing the keys management aspect that Chestnut is looking to have.

#### 1.4.3.3 SageMath mathematical software

SageMath is a free open-source mathematics software system licensed under the GPL. It builds on top of many existing open-source packages such as NumPy, SciPy, and many more. It can be used both programmatically through a common, Python-based language or directly via interfaces or wrappers (SageMath, 2020).



```
22 1 M = AlphabeticStrings()
23 2 E = SubstitutionCryptosystem(M)
24 3 # define a cipher which flips the alphabet
25 4 K = M([ 25-i for i in range(26) ]); print K
26
27
28 1 # encrypt a message with this cipher
29 2 msg = M("THECATINTHEHAT")
30 3 print E.enciphering(K,msg)
31
32
33 1 # decrypt a crypttext using the same cipher
34 2 print E.deciphering(K,M('GSRHRHGLKHVXIVG'))
35
36
37
38
39
```

The screenshot shows the SageMath web interface. The top bar contains buttons for Run, Stop, Restart, and Help, along with input/output toggles and a toolbar with various mathematical symbols. Below the toolbar is a menu bar with categories like Modes, Help, Data, Control, Program, Plots, Calculus, Linear, Graphs, Number Theory, and Rings. The main area is a code editor with line numbers 22 to 39. The code defines an alphabetic strings object, a substitution cryptosystem, and a cipher that flips the alphabet. It then demonstrates encryption of "THECATINTHEHAT" to "GSVXZGRMGVSZG" and decryption of "GSRHRHGLKHVXIVG" back to "THISISTOPSECRET".

Figure 1-13 SageMath mathematical software interface

SageMath is a powerful open source tool but is far from what Chestnut is trying to be. It is not only a complex software that has a high difficulty for entry but is also primarily a mathematical software and not a cryptographic one.

It does have cryptographic capabilities, but it is not its primary focus and most of it would have to be achieved in Python, programmatically.

Using something like SageMath would not be a huge improvement over the shell terminal.

## 1.5 Our solution

For the solution subchapter, we will be giving an overview of the main technologies, frameworks, and libraries we have used for Chestnut. I will also be mentioning the core architectural patterns we have chosen to follow in the design of the API.

We will be going into detail about these technologies and express why we chose them over others in the third chapter of the report, the product documentation, under front end and backend, respectively.

### 1.5.1 Front end – ReactJS



React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components” (Intro to React, 2020).

React is currently the most popular front-end library for building a highly flexible and efficient single-page application (SPA) in the world. It trumps both of its main competitors Angular and Vue.

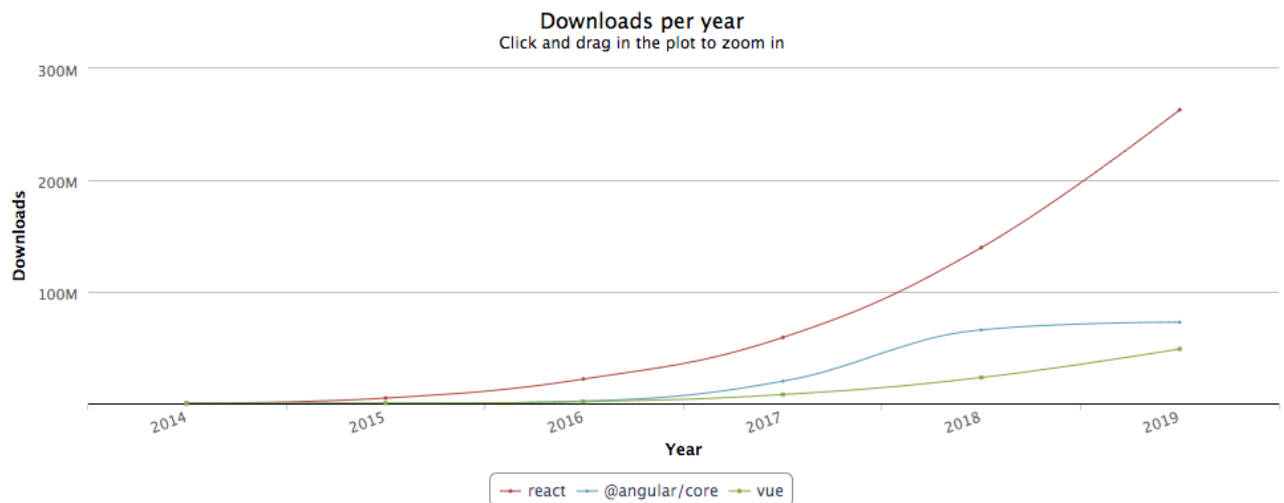


Figure 1-14 Downloads per year comparison for React, Angular, Vue

(tkrotff, 2019)

We not only chose React because of its popularity but also because of the experience we had with it before the project. Angular is known to be slightly more complex for beginners and Vue is a technology we have not used before.

### 1.5.2 Back end – NodeJS



When it comes to the server, we had multiple languages and technologies to choose from but ultimately went with NodeJS because we wanted to have 1 single language, JavaScript, being used across both front and back-end.

By doing this, we eliminate any hurdles that would be present from us not knowing or being familiar with a language or platform. The product owner had no technology requirements, so this decision was unproblematic.

### 1.5.3 HATEOAS REST API



The client and the server will not be communicating directly but through an API. We designed and implemented a REST API for the backend that facilitated communication between the client and the server. Additionally, we chose to make use of the HATEOAS architectural style.

With HATEOAS, a network application provides information dynamically using hypermedia to a client. A client communicating with a HATEOAS REST-API developed according to the level-3 principles needs little to no prior knowledge about how to interact with an application or server beyond a generic understanding of hypermedia (HATEOES, n.d.).

With this, our API can be considered truly RESTful.

This is a very important part of our application, so we will be going more into detail about REST and HATEOAS later in the report.

## 1.6 Conclusion

We believe that by bringing these technologies together and with our experience in web development, we have created a full-stack application that fulfills all of Thomas' requirements and specifications. As we will see later in the report, we also added certain features that were not part of the original description but were added to make the application better.

The product and development process has not been without its challenges, but we have managed to get past them and created a product that both we and the product owner are very satisfied with.

The next chapter is all about the product process.

# PROCESS DOCUMENTATION



## 2 Process documentation

### 2.1 Introduction

The process document describes how our group has worked with the project. We will go over structuring, and planning, our working methods, technologies, and the tools we made use of.

We intend this chapter to be readable and understandable to people who may not have a technology background or may just be technology hobbyists. Therefore, we try to explain concepts in a simple manner. The more in-depth description of the technologies we used is in chapter three, product documentation.

### 2.2 Documentation tools

During the project period, we used several tools to solve the tasks. In this chapter, we will go through the tools we used to communicate, plan, allocate tasks, and not least, develop the product. These tools are significant for understanding the work process and the way we have worked on this project.

#### 2.2.1 Trello

Trello is one of the most well-known web-based project management tools with nearly five million users. With the Kanban-style list-making feature, Kanban was our first choice for project management. We used Trello to plan and organize tasks throughout the entire project (Trello, 2020).

#### 2.2.2 Visual Studio Code

Visual Studio Code (VS Code) is an open-source and lightweight but powerful source code editor that comes with built-in support for JavaScript and Node.js. A productive aspect of the VS Code is the extensions it offers. Extensions let a user add languages, debuggers, and other tools to create their desired workflow (Visual Studio, 2020).

The alternative code editor that we considered using was IntelliJ IDEA which we had used in the past, but since one of our team members was using VS Code and our solution would make use of JavaScript and Node.js, it was an easy choice for everyone to use VS Code.

#### 2.2.3 GitHub

GitHub was our primary tool for version control. Using GitHub is a secure and effortless way to store and share files between team members. When one of the team members finished a task, he would push the changes to the GitHub repository (GitHub, 2019).

#### 2.2.4 Jitsi Meet

Jitsi Meet is an open source and fully encrypted video conferencing solution accessible via browser without the need to create accounts. (Jitsi Meet - Instant free videoconferencing, 2020).

We had intended to use Meet as an alternative to when meeting in person was not possible. But because of unforeseen circumstances, it became our only way of meeting to share and discuss the project. A key feature of Meet is the ability to share screen with others. We used this feature to share the progress with the product owner and get feedback.

#### 2.2.5 Adobe XD

Adobe XD is a powerful UI/UX design and collaboration tool (Adobe, 2020).

We used Adobe XD to create a design for how the website would look. This design was used as a starting point when we started the development.

### 2.3 Software Development Methodology

Software development methodology (SDM) or software development life cycle (SDLC) refers to a set of principles or rules that aim to ensure that development is completed on time and within a reasonable budget.

In 2001, 17 people came together and published the agile manifesto which comprises 12 principles. This manifesto is accessible on the agile manifesto's website. (Principles behind the Agile Manifesto, 2001).

We can describe the most modern and popular development methodologies as agile. Agile represents a philosophy for software development that emphasizes the iterative approach and customer satisfaction.

To explain it briefly, agile wants to move information between people quickly and accurately by placing people physically closer and trying to replace documents with talking in person and at whiteboards. The other core idea is to reduce the time between making a choice and seeing the consequences of it by working incrementally (Cockburn & Highsmith, 2001).

#### 2.3.1 Agile frameworks

Frameworks such as Scrum, Lean Software Development (LSD), and Kanban are distinct approaches to software development based on agile principles. Each framework has certain strengths and

weaknesses that a team should consider before starting to use them. Another common approach is to use several of these frameworks together to create an ultimate SDM that matches the needs of the team or the organization.

### 2.3.2 Our choice

The product owner did not require us to use a certain method or framework for this project. Instead, we did thorough research on the most popular agile frameworks to help us choose the ideal framework that not only would match the project requirements but our personalities and workflow. We concluded that Kanban was the best choice, but taking our experiences with Scrum into account, we integrated some of Scrum's principles into Kanban to make it more suitable for us. Scrum ban is a term for when Scrum and Kanban are combined.

Kanban encourages leadership from any position and focuses on personal competency. We believe this is excellent for our small team because we can self-organize, and each member can work independently.

Kanban is also flexible and welcomes changes that directly result from its incremental approach to development. In incremental software development, we design and implement an initial solution and then add a little more in each increment until the product is finished. The incremental approach is ideal for our project requirements.

## 2.4 Kanban

Kanban has three core principles: Visualization of the work, limiting the amount of work in progress (WIP) and maximizing efficiency.

The benefits of visualizing the work are that we can understand it better, easily show it to others, and keep everyone focused. Limiting the work in progress ensures the team does not undertake too much work at once, and ensures tasks are taken from the backlog after finishing previous tasks which leads to a better overall throughput.

### 2.4.1 Kanban board

A Kanban board allows us to visualize the work by creating Kanban cards. A Kanban card is the equivalent of a sticky note that someone would place on a whiteboard to represent a task. We needed to decide between two Kanban philosophies. The first one is by David Anderson, who took the Kanban method from the manufacturing process into software development. He defines five key elements that help manage the flow of work within a Kanban board. The five elements are visual signals, columns, WIP limit, commitment points, and delivery points. The second philosophy is personal-Kanban by Jim Benson and Tonianne DeMaria. Personal Kanban has two rules that are: Visualize the work and limit work in progress.

We used the second philosophy because it is more applicable to a small team with a simple workflow. The other key choice to make was whether to have one board for the entire project or create separate boards for distinct parts of the project. We created the first board for planning and design. It helped us keep track of the tasks we needed to do before starting to code. Some tasks

included in this board were writing the preliminary report, creating user stories, project requirements, and so on.

The other two boards that we created were for the frontend and the backend. Frontend includes the user interface and how the website looks while the backend is about creating the server and database to keep user data.

We used Trello as our digital Kanban board created a card for every task we needed to do. All team members could start working by pulling a card from the “Tasks” column on the left to the “In Progress” column. After finishing the task, the member would move the card to the “Ready for Review” section so another team member could review it. The reviewer could then place the card in the “Done” column to mark it as completed.

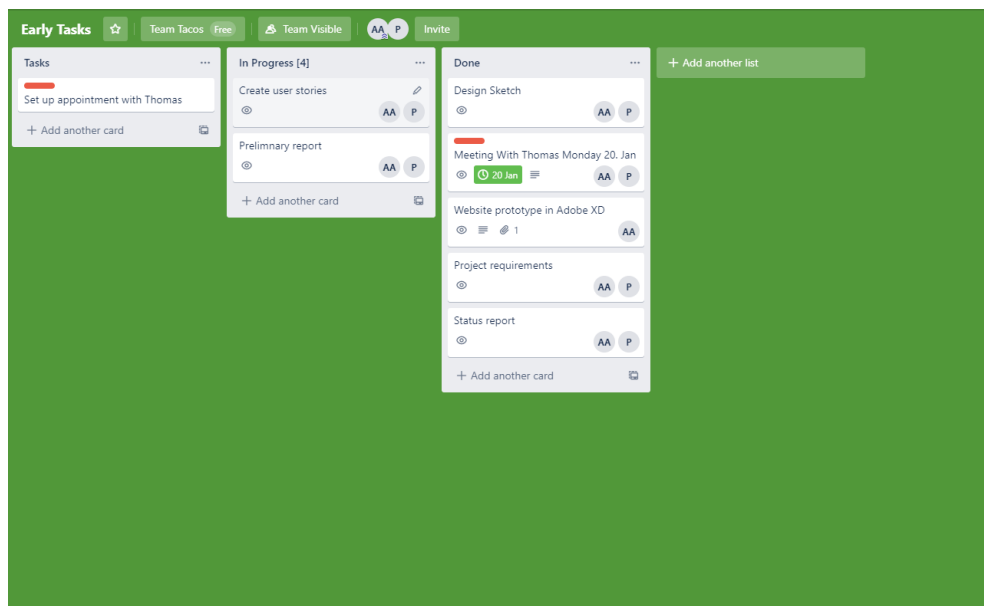


Figure 2-1 Trello being used as our digital Kanban board (Early tasks)

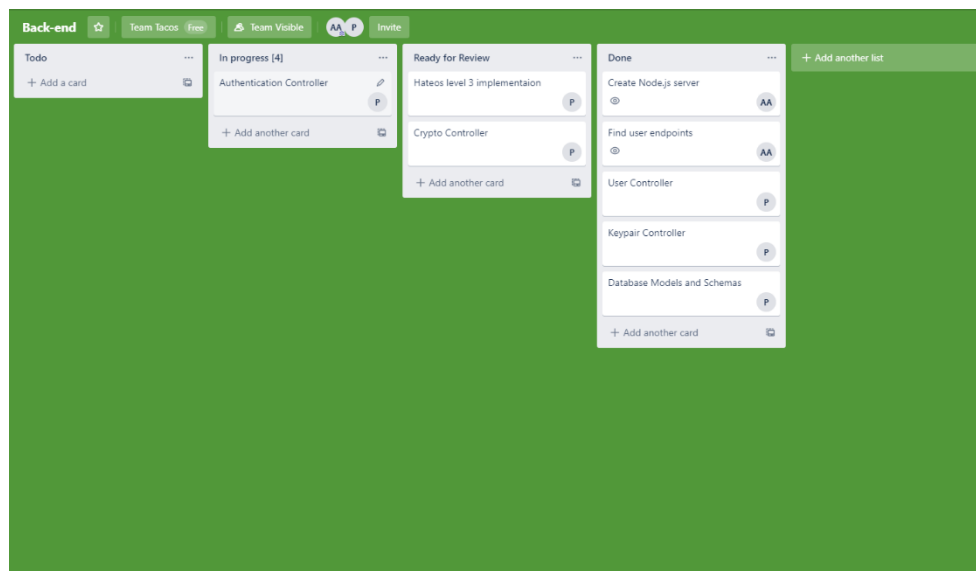


Figure 2-2 Trello being used as our digital Kanban board (Backend tasks)

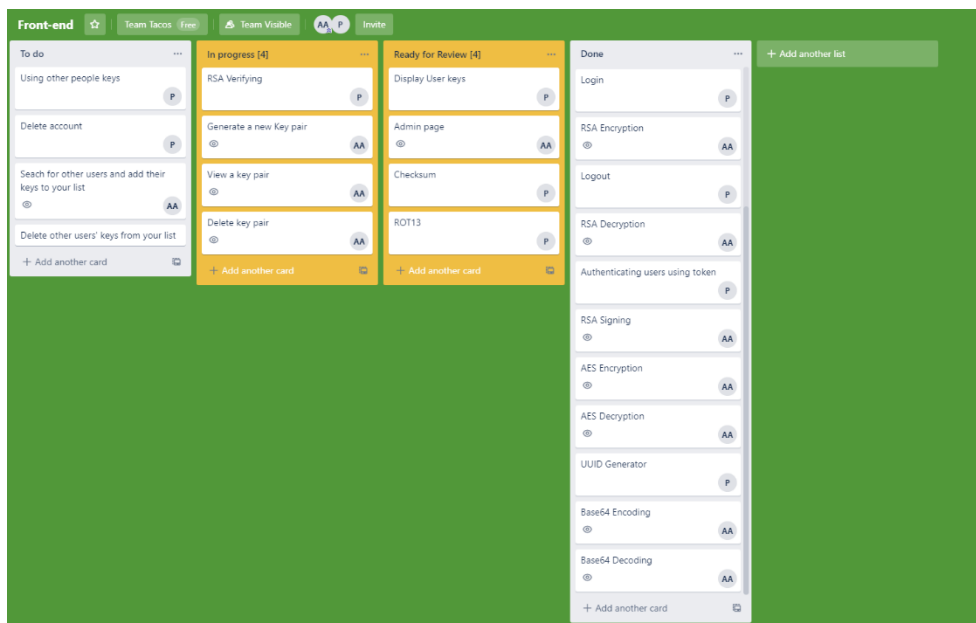


Figure 2-3 Trello being used as our digital Kanban board (Frontend tasks)

## 2.4.2 Work in progress limit

The idea behind limiting the number of concurrent tasks a team works on is that the team members can have a better focus on doing a few tasks at a time which results in shorter lead time which is the time between when a member starts a new task until he finishes the task.

When using the WIP limit we kept four concepts in mind. The first concept was to divide the workload of each task and create cards that would need nearly the same amount of work to complete. This would help us when we wanted to estimate how much time we needed to complete the project.

The second concept was to map the WIP limit to the team's skill. Since we had similar skill sets, we figured that a limit of two tasks per team member was ideal. The third concept was the importance of reducing idleness. Whenever a team member had downtime, he would try to help another team member with their task. By doing this, the team could have overall better productivity.

The last concept was the importance of understanding what the WIP limit is. We needed to remind ourselves that completing a task as quickly as possible is not the best option and we should take the quality of the product and codebase into consideration.

## 2.5 Scrum

The way Scrum works is that a team will have meetings where they discuss the project and split the work in sprints. During a sprint, the team holds short daily meetings to talk about how the sprint is going. After completing the sprint, the team reviews the work. The team will also hold a retrospective meeting where they discuss the last sprint and methods to improve the next one.

### 2.5.1 Scrum's daily standup meetings

In Scrum, teams have a short meeting in the morning where each member of the team answers three questions. What did I work on yesterday, what am I working on today and what challenges I am facing?

We believed that the addition of a daily standup meeting would assist us in having a better and more transparent communication during the project. However, it was essential to change the questions we asked, so they would reflect our development method, Kanban.

The first thing we did on every meeting was to check that our Kanban board is up to date. Then each team member would answer three questions. What is impeding us? By asking this question we would talk about the problematic items on the board and the potential solutions for them.

What is the flow like and what can we improve? Workflow is an important aspect of Kanban and we would constantly evaluate our work to see if we were having a good flow or not. We would then talk about how we could improve our overall work.

### 2.5.2 Scrum Sprints

In Scrum, we can break a complex project down into what we call sprints. A sprint is a time-boxed period, which cannot be more than a month. During this period, the Scrum team works to deliver parts of the project that are usable and potentially reusable.

We made use of sprints in our project, but not in the traditional sense where we planned out all the sprints from the beginning. Instead, after we completed the initial design of the solution, we met with the product owner every two weeks.

The first part of the meetings was retrospective; we would look at the work that we had done in the last two weeks and talk about it. In the latter part of the meetings, we would discuss the concepts we wanted to explore and the work that we needed to do for the next two weeks.

We did not use conventional Scrum sprints because they lack flexibility regarding adding or changing user stories during the sprint time and we wanted to include the product owner and discuss the sprints before starting them.

## 2.6 Development Process and methods

To develop the final product, we have made use of several methods in addition to Kanban and Scrum. In this section we will look at what these methods are, why we used them and how did they work in practice.

### 2.6.1 Pair programming

Pair programming is a technique in agile where two programmers work together. The first person who we refer to as the *driver* writes code while the other person whom we call navigator, observes the process (Pair Programming - Does it really work?, n.d.).

It is the navigator's responsibility to not only observe but to come up with ideas on how to improve the code and address the problems that may occur. The roles are switched frequently.

We used pair programming when we were working on critical parts of the application to help us hinder issues from arising. One of the challenging parts of the development was integrating all the different libraries and frameworks into React. It was easier and more instructive to work together.

### 2.6.2 User-centered design

User-centered design (UCD) or user-centered development (UDD) is an iterative process where the users and their needs are in the focus during different phases of design and development.

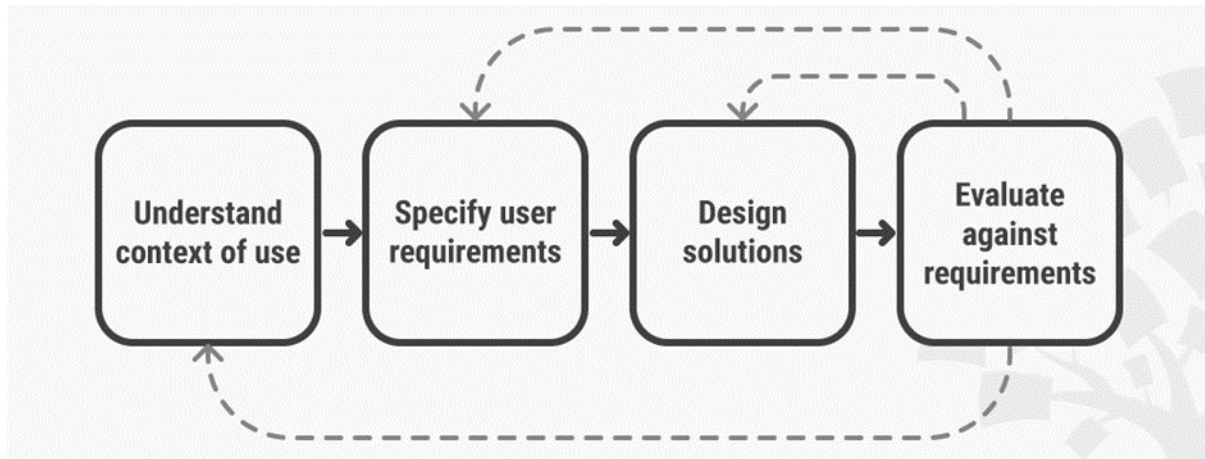


Figure 2-4 User centered design diagram

(What is User Centered Design?, n.d.)

The reason we focused on the primary users of our application from the beginning until the very end, is the fact that it was critical that the Chestnut as an educational tool, would not only be helpful but it should be easy and intuitive to use. Being students ourselves, we could feel how the students must be frustrated, and we focused on solving that.

## 2.7 Project requirements

The original Chestnut is a simple JavaFX application that students can use to explore the principles behind public key infrastructure (PKI). However, it comes with complications that make it hard to use.

Even though it was a JavaFX application, the developer of the original application reported multiple difficulties in terms of operating system portability. Mac users had difficulty installing the application and some of the GUI widgets did not function correctly. Some Windows users were unable to install the application and the installation process simply stopped without an error message or the application simply closed itself while it was in use. Sharing public keys via email was also problematic as the keys were sometimes autoformatted resulting in line breaks that broke the key.

The tool was frustrating both for the teacher as well as the students, so our job was to look at Chestnut as an inspiration and create a web application that lets students explore the concepts behind PKI. An additional requirement was that it could also be used to illustrate base64 encoding, checksums and to generate UUIDs. These are common themes that students in library and information science need to explore.



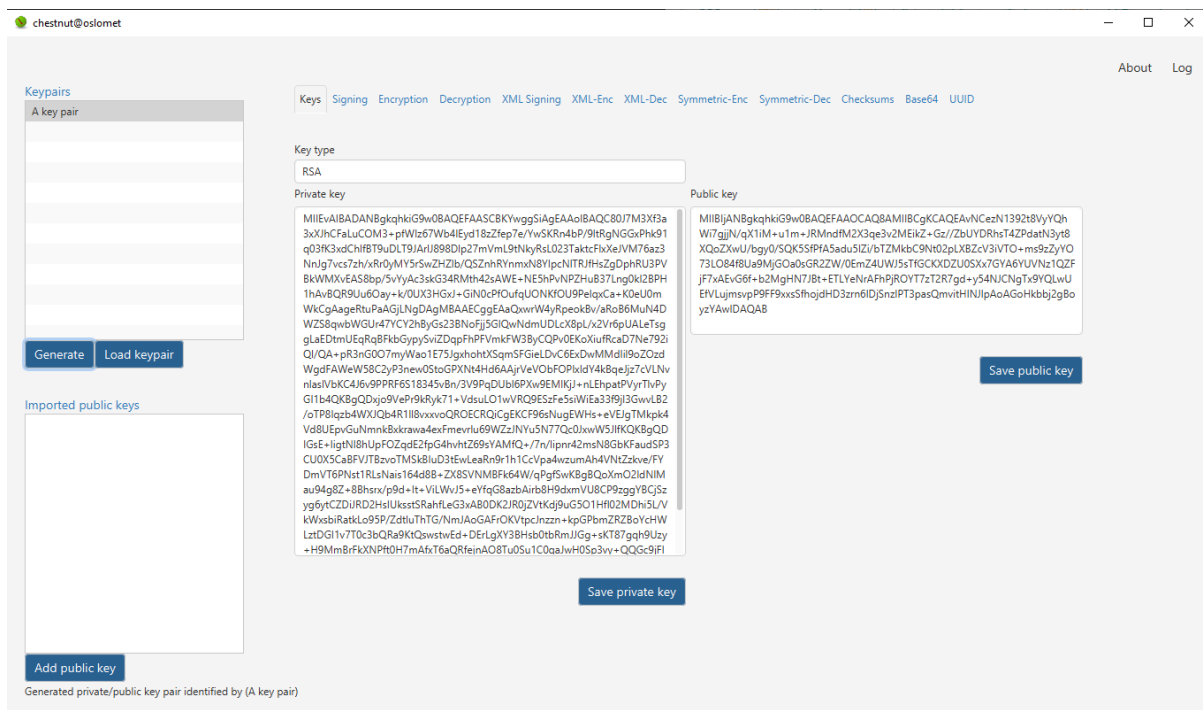


Figure 2-5 Chestnut as a JavaFX application

## 2.7.1 Functional requirement

A functional requirement (FR) is a description of a function a system must offer (What is a Functional Requirement?, 2020). Understanding functional requirements and specification are essential for building a valuable product.

We created a list of requirements by running the Chestnut application on our local machines and noting the different components of it. In a meeting with the project owner, we talked about the list and constructed the following requirements.

Authentication:

- A user should be able to register for a new account using their email address
- A user should be able to login using their username and password
- A user should be able to log out of the system.

Settings:

- A user should be able to delete their account

Keys:

- A user should be able to generate RSA key pairs with different lengths
- A user should be able to delete any of his key pairs

- A user should be able to search for other users' public keys and add them to a list
- A user should be able to remove other users' public keys from his list
- A user should be able to view the content of a key

Asymmetric encryption/decryption:

- A user should be able to encrypt data using a public RSA key
- A user should be able to decrypt data using the corresponding private RSA key
- A user should be able to sign data using a private RSA key
- A user should be able to verify a signature using the corresponding public RSA key

Symmetric encryption/decryption

- A user should be able to encrypt data with AES by providing a pass phrase
- A user should be able to decrypt data by providing the same pass phrase
- A user should be able to encrypt data with Rot-13
- A user should be able to decrypt Rot-13

Utilities:

- A user should be able to encode plain text to base64 format
- A user should be able to decode base64 format to plain text
- A user should be able to generate checksum for data inputs and files
- A user should be able to generate UUIDs

Demo:

- A user should be able to access a demo of the main application without logging in

Admin:

- An admin should be able to see all the registered users
- An admin should be able to search for any registered user
- An admin should be able to delete any user

## 2.7.2 Non-functional requirement (NFR)

We can think of non-functional requirements as quality attributes for a system. They describe criteria that can judge the operation of a system (What is Non-Functional Requirement? , 2020).

The non-functional requirements that the product owner stated are the following:

- Either React or Angular should be used as the primary framework for frontend
- The application should store as little information about the users as possible

- The application should follow the GDPR regulations
- The application should be user friendly

## 2.8 Backlog

A product backlog is a list of features, bug fixes and other activities that a team may deliver.

Product backlog is an important part of development because it gives the team members a starting point based on a priority sorted list. We can think of backlog as a “to do” list with the most important functionalities of the project. We created a user story for each project requirement and gave them a priority score of high, medium, and low based on the discussions we had.

Based on the system requirements, we made user stories.

User Story	Priority
As a user, I want to login to access the entire application	
As a user, I want to logout to preserve my security	
As a user, I want to delete my account to ensure my privacy	
As a user, I want to generate RSA key pairs to encrypt and decrypt data	
As a user, I want to delete my key pairs	
As a user, I want to encrypt and decrypt data using symmetric or asymmetric algorithms to understand them better	
As a user, I want to generate a checksum for input or a file to ensure its integrity	
As a user, I want to add other users' public key to a list so I can use it to encrypt data	
As a user, I want to sign data using a private key so others can make sure its genuine by verifying the signature by using public key	
As a user, I want to verify the signature with the public key to ensure its authenticity.	
As a user, I want to encode plain text into base64 format or vice versa because I want to learn more about encoding	
As a user, I want to create multiple UUID because I want to use them in another context	

As an admin, I want to have an overview of all registered users

As an admin, I want to delete users in case they are inactive

## 2.9 Project Phases

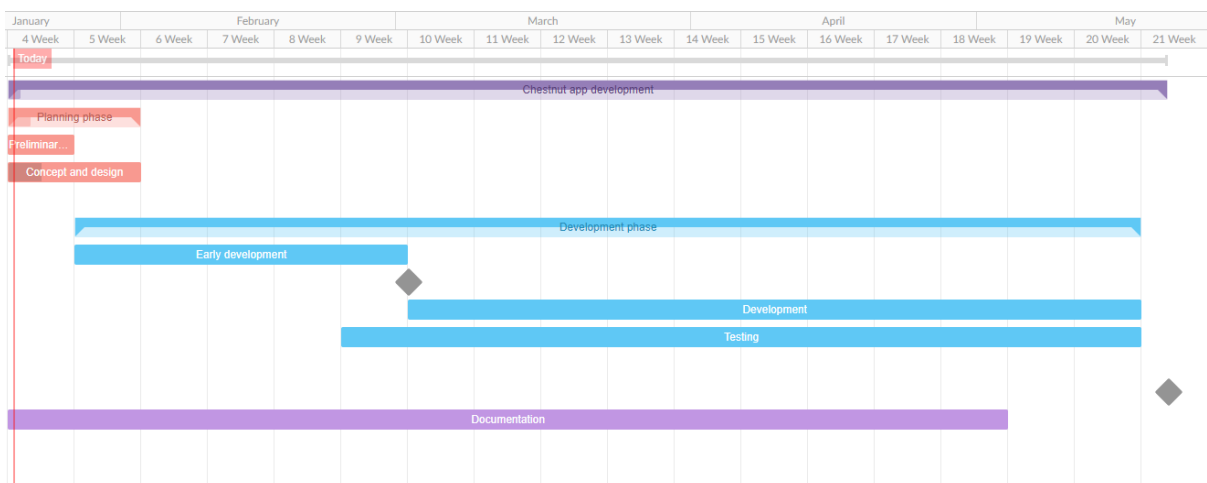


Figure 2-6 Project phases shown on the work timeline

The three distinct phases of our project were planning, developing, and writing documentation. We broke down each phase into one or several two weeklong sprints. We will go over the planning phase followed by the two development sprints we performed. Then, we will then talk about the adjustments we made to the sprints and our method of working when we had to work remotely.

### 2.9.1 Planning Phase

Planning was is the sprint 0. We got introduced to the issue, and we needed to come up with a plan that would help us deliver a neat solution timely. To compose such a plan, we needed to gain a deeper understanding of the problem we were solving and therefore we started by creating the product requirements. After creating the list of all the essential product requirements, we created the user stories to understand the expectations of the end-users of the product.

### 2.9.1.1 Design sketches

With project requirements and user stories in place, we felt that we had a good fundamental understanding of the issue and the solution we needed to develop. So we started sketching with pen and paper until we knew we had something that would work well.

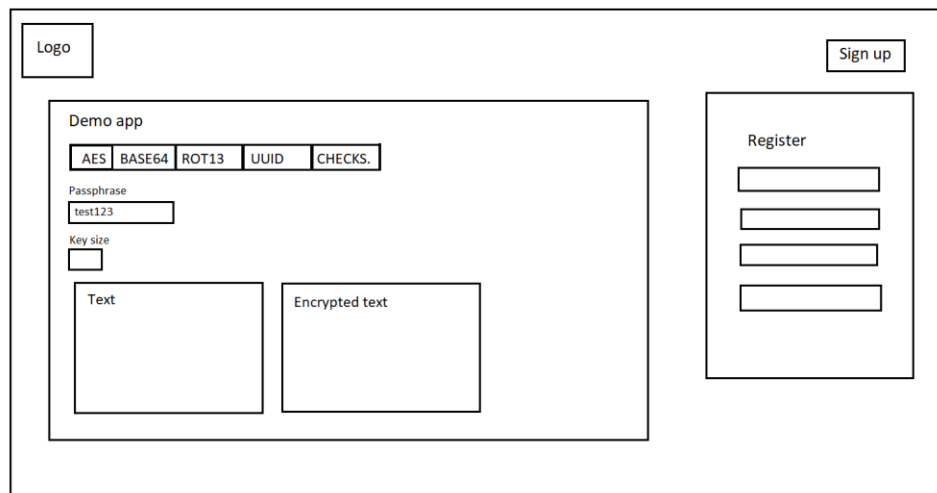


Figure 2-7 Early design sketch for the landing page and demo component

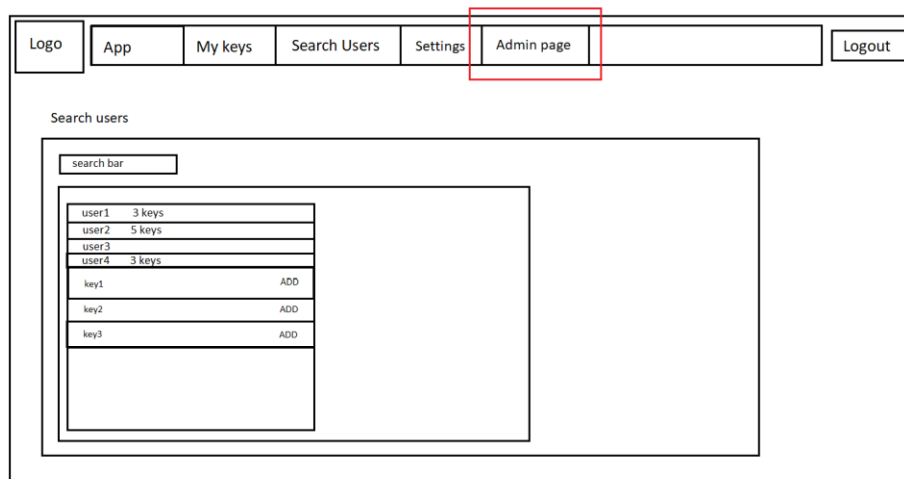


Figure 2-8 Early design sketch for website after authentication

### 2.9.1.2 Prototype

We used Adobe XD design and prototype functions to transform our sketches into an interactive prototype that showed the layout of the website, how a user could navigate to different pages and perform certain actions.

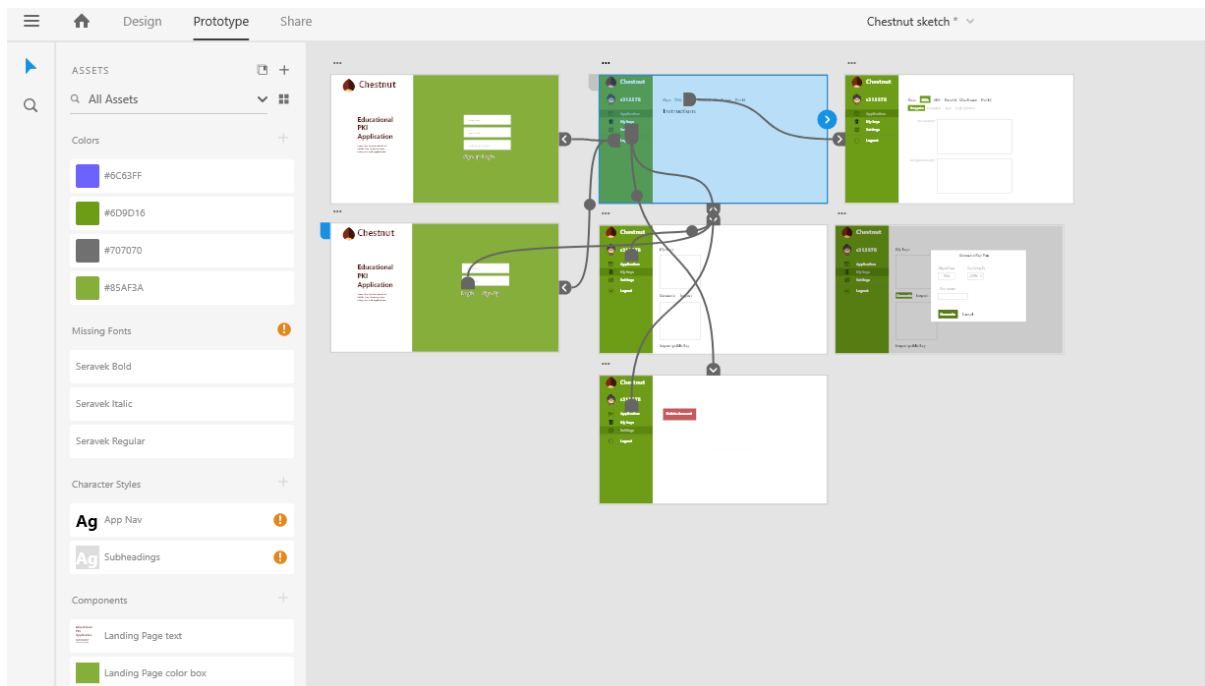


Figure 2-9 Early interactive prototype made in Adobe XD

## 2.9.2 Development phase

The development phase was substantial, and therefore, it was essential to break it down into several sprints. In a meeting with the product owner, we discussed how we could proceed with the development. The first option was to implement the frontend because we already had the prototype and we would implement the API and the backend later. The second option was to identify API endpoints and design our REST API before moving on to developing the frontend. The last option was to work on both parts simultaneously.

With input from the product owner, we concluded that the second option was the most beneficiary choice because not only did we not have enough resources to work on both parts of the application at the same time, designing and developing the API would allow us to have an even more clear vision of the website's layout and its functionalities that would also cause a smoother transition between the prototype we have to the completed website.

## 2.9.3 API Design

Before diving into the sprints in the development phase, we should mention that the backend is the backbone of the solution that does not directly solve any of the project requirement, but it lays the ground to fulfill the requirements when we would implement the frontend. Consequently, we do not provide a list of functional and nonfunctional requirements for each sprint. We will describe what topics each sprint covered and what we achieved after completing the sprint.

### 2.9.3.1 Sprint 1

The sprint number one consisted of researching and designing the REST API. We looked at the user stories that we had created previously and tried to figure out the API endpoints, which are basically all the places where the users need to interact with our server or the database. We used swagger to design, and Postman to test our REST API.

At the end of this sprint, we held a meeting with the product owner to discuss the REST API. We discovered that we had missed some endpoints we needed to implement before the API was complete. We also planned out the work for the next two weeks.

### 2.9.3.2 Sprint 2

The plan for sprint number two was to not only work on completing the API, but to work on the backend in its entirety.

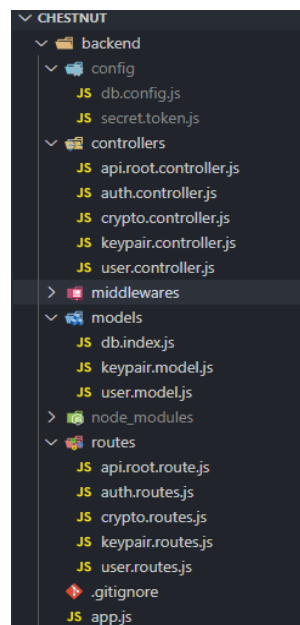


Figure 2-10 Figure showing file structure of Chestnuts backend directory

We thought about file structure before setting up the server. We wanted the file structure to be clear and easily understood. Therefore, we created folders that would hold several files that were similar.

---

**GET** ^

- GET** /api
- GET** /api/users
- GET** /api/users/{id}
- GET** /api/keys
- GET** /api/keys/users/me
- GET** /api/keys/users/{id}

**DELETE** ^

- DELETE** /api/users/{id}
- DELETE** /api/keys/{id}
- DELETE** /api/users/me

**POST** ^

- POST** /api/keys/new/users/me
- POST** /api/encrypt1
- POST** /api/decrypt
- POST** /api/login
- POST** /api/signup

**PATCH** ^

- PATCH** /api/users/{id}

*Figure 2-11 Overview of all Chestnut API endpoints*

According to the feedback from the product owner, we added the missing API endpoints and adjusted the rest of the endpoints. We will cover the API and how we secured it in depth in chapter three, product documentation.

We also created the database and the relevant models for the user and the keypair tables. We set up the database locally using MySQL Workbench.



Table	Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
keypairs	createdAt	datetime		NO			select,insert,update,references	
	KeypairID	int		NO			select,insert,update,references	auto_increment
	Length	int		NO			select,insert,update,references	
	Name	varchar(255)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
	PrivateKey	text		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
	PublicKey	text		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
	Type	varchar(255)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
	UserID	int		YES			select,insert,update,references	
users	Email	varchar(255)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
	ID	int		NO			select,insert,update,references	auto_increment
	IsAdmin	tinyint(1)	0	NO			select,insert,update,references	
	Password	varchar(255)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
	Username	varchar(255)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	

Figure 2-12 Figure of the chestnut database with keypairs and users tables

In the end, we connected the server, the API, and the database together to complete the backend. We performed tests using Postman to ensure everything worked as intended before moving on.

## 2.10 The big issue

By the end of the sprint 2, we found ourselves in a challenging, unforeseen situation which was the outbreak of the COVID19 virus. The university's campus was closed, and physical meetings were no longer possible. Our workflow got disrupted, and to get back on track, we needed to adapt and apply a new workflow.

Before the outbreak, our team was meeting daily to work on this project from around 12 to 6pm. By being close to each other, we had ensured that the information flow was fast and accurate. We could also discuss the tasks and work on a solution together.

## 2.11 Our solution

The first thing we did was to move away from having sprints and instead made use of Kanban pulling system. Each team member would pull one or two tasks depending on how much capacity they had, implement the feature, and incrementally add them to the product.

The next step we took to improve our workflow was to stop having daily standup meetings. Instead, we had online meetings only when we needed to decide on something or faced a challenge. The last step was to increase the number of meetings with the product owner from biweekly to weekly. We used Jitsi Meet for our online meetings.

We believed that by reducing the number of group meetings, we could focus on completing the tasks we had on our hands, while the increased number of meetings with the product owner was positive because we were getting fast and continues feedback which ensured us that the project was on the right path.

We are thankful to our supervisor who helped and supported us not only in this period but during the entire project.

## 2.12 Frontend development

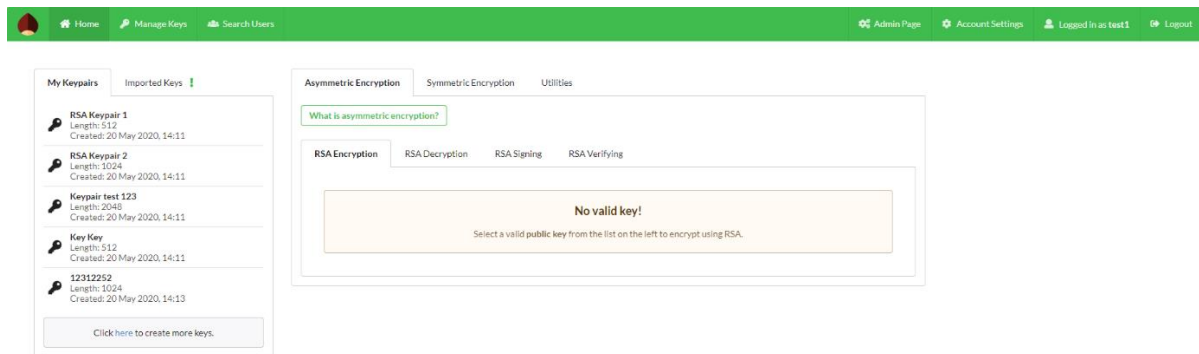


Figure 2-13 Chestnut as a web application

We started working on the frontend using our new workflow. We mainly followed the prototype we had created previously, but made necessary adjustments based on the knowledge and information that we had gained until this point.

## 2.13 Documentation

After the Chestnut website was up and running with all the required specifications, we had to complete the documentation. During the planning and developments phases, we completed some documents including project requirements and user stories. We took notes throughout the project to ensure we would forget nothing important. The writing process took place over the last three weeks.

We created a documentation folder in our project repository that includes all the documents. Our supervisor introduced us to a markup language called Markdown (Gruber, 2020). We used Markdown as our primary method of writing documents. The plugin, “Writage”, gave us the option to open, edit and export Markdown documents inside Office Word.

## 2.14 Reflections

We had to keep in mind that Chestnut is an educational tool that students use it. Although it was easy to look at the original Chestnut and figure out what technical functionalities we needed to include, we had to find creative ways to not only implement them but also improve them to become user friendly and intuitive to use.

After researching SDMs, we knew that Scrum ban would suit our needs. This method was working well until the virus outbreak, which surprised us. We needed to compose ourselves and weigh our

options once again. It took us some time to get rid of the confusion and figure out an alternative plan, but we overcame this obstacle and worked hard to deliver the product in the same time frame as before.

The development phase was interesting because there were always unique ways of implanting the same thing. We needed to be considerate of which libraries we used and what effect they had on the final product.

## 2.15 Conclusion

The process documentation gives insight about the way we came up with a plan and how we executed it during the entire project. This period was demanding and heavy as mistakes or misunderstandings could have had major consequences both on the group's work and on the project itself.

We used Scrum ban, which is a combination of agile frameworks, Scrum and Kanban. What influenced our choice and methodology was the fact that it was not essential to make an in-depth plan for the entire project. Agile approach covered project requirements and was also suitable for our group because we had used it in the past in other projects.

From the beginning we knew that we had to work consistently to deliver the product on time. We split the project into several two-week timeframes where we worked on certain aspects of the product each time. During each period, our team had daily meetings where we discussed and developed the product.

Under way, we faced some challenges, but we kept working as a team and supported each other. This helped us in overcoming the challenges and completing the project.

# PRODUCT DOCUMENTATION

## 3 Backend documentation

### 3.1 Introduction

In the computer world, backend refers to any part of a website or software program that users do not see. It contrasts with the frontend, which refers to a program's or website's user interface. In programming terminology, the backend is the data access layer while the frontend is the presentation layer (Backend Definition, 2020).

In this part of the chapter, I will be explaining the technologies we have used for the backend, how Chestnuts backend is structured, go into further detail on our REST API and explain some of our design choices there.

### 3.2 Technologies

#### 3.2.1 Node

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a web browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting. (Node.js, 2020).

#### 3.2.2 Express

Express.js, or simply Express, is a web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js. (Express, 2017).

#### 3.2.3 express-hateoas-links

Extends express's `res.json` method to accept an array of HATEOAS links to be appended to the output JSON object. This is crucial to achieve the type of REST architecture we want to achieve with our API. More about this will come later when talking about the API. (express-hateoas-links, 2019).

#### 3.2.4 body-parser

As of the time of developing Chestnut, Express version is 4.17.1 and to be able to handle incoming HTTP POST requests in Express.js version 4 and above, you need to install the middleware module called `body-parser`. (body-parser, 2020).

`body-parser` extracts the entire body portion of an incoming request stream and exposes it on `req.body`. As developers, we can thereafter take this body and do the necessary operations upon it depending on the request.

#### 3.2.5 mysql2

Since we are using MySQL as our database solution, the `mysql2` package is needed.

mysql2 is a client for NodeJS with focus on performance which makes writing code for a MySQL database easier. (mysql2, 2020).

### 3.2.6 Sequelize

Sequelize is a promise based Node.js ORM for Postgres, MySQL, MariaDB, SQLite and Microsoft SQL Server. It features solid transaction support, relations, eager and lazy loading, read replication and more. (Sequelize ORM, 2020).

Using Sequelize, we can write database models representing tables, make queries to the database and handle any errors that may occur. All that in JavaScript language syntax.

### 3.2.7 JSON Web Tokens (JWT)

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. (Introduction to JSON Web Tokens, 2020).

### 3.2.8 bcrypt

bcrypt is a simple, lightweight library that helps you hash passwords in JavaScript. We have used this to hash every incoming password within the POST request before it is stored in the database. (bcrypt, 2020).

## 3.3 API

API stands for **A**pplication **P**rogramming **I**nterface and is a common term for how different software or code interacts with each other. These can be web-based APIs, where a client, in our case a web browser, will send requests and receive its data inside a response from the server.

The returned response may come in many forms but for Chestnuts API, we have chosen to communicate in JSON (**J**avaScript **O**bject **N**otation) format.

```

1 {
2   "status": "200 - OK",
3   "encryptedText": "GGXf0ftkrJK3lnjLbzWuntHpXIz0N84au/2/p7zIj8Fk5g/TtJUX55jLnktABbpUHz20jI",
4   "links": [
5     {
6       "self": {
7         "method": "POST",
8         "description": "Encrypt string using the provided public key.",
9         "href": "http://localhost:8080/api/encrypt"
10      }
11    },
12    {
13      "method": "POST",
14      "description": "Decrypt cipher string using the provided private key.",
15      "href": "http://localhost:8080/api/decrypt"
16    }
17  ]
18 }

```

Figure 3-1 Chestnut API response in JSON format

Having an API also makes the backend more secure. With an API in place, clients do not communicate with the backend directly but rather through this interface. This also means that, sensitive code within the backend is both hidden and unreachable.

Examples of such sensitive information within the code can be credentials to the database connection, JWT passphrases or even the source code as vulnerabilities would be found much easier if it were to be public.

The figure below shows how a browser communicates with an API rather than a server directly.

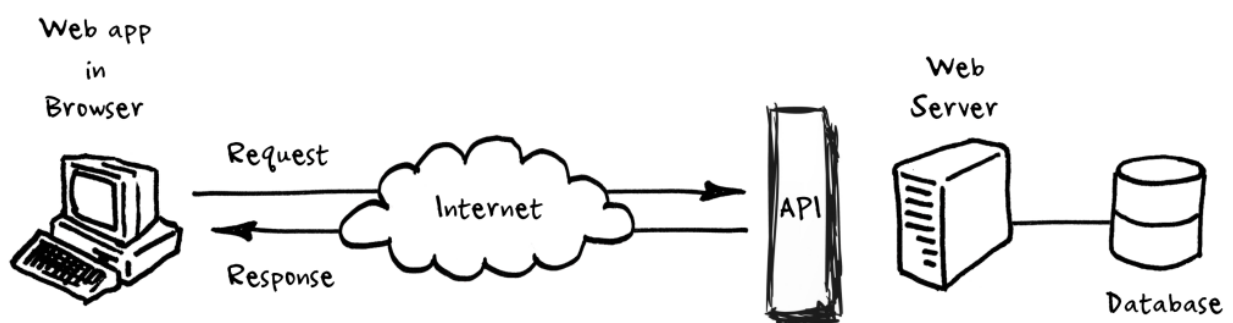


Figure 3-2 Communication between a client and an API

(Drummond, 2013)

A core characteristic of the Chestnut API is that it is also RESTful and that it adheres to the REST set of architectural constraints and characteristics. What that means, we will be clarifying in the next section of the chapter.

### 3.4 REST API

REST stands for **R**epresentational **S**tate **T**ransfer and has over time become one of the most important architectural styles when building APIs in modern web development (HATEOAS and REST: The hypermedia principle, 2018). The REST architectural style was invented by Professor Roy Fielding in his dissertation in the year 2000 (Fielding, 2000).

Communication between client and REST APIs is done through the HTTP protocol and with the use of HTTP verbs.

These requests from the client are made to a URI, a Uniform Resource Identifier. The URI is the address where the request is being made. URIs are also commonly referred to as endpoints. This is combined with an HTTP verb to achieve the desired action on a collection or resource on the server.

Collections are a collection of many individual resources while a resource is a single individual item. In Chestnuts case, we have collections in the form of keys and users. A resource can for example be a single user with a specific ID.

Below are examples of endpoints that the Chestnut API provides next to the HTTP verb they are associated with.



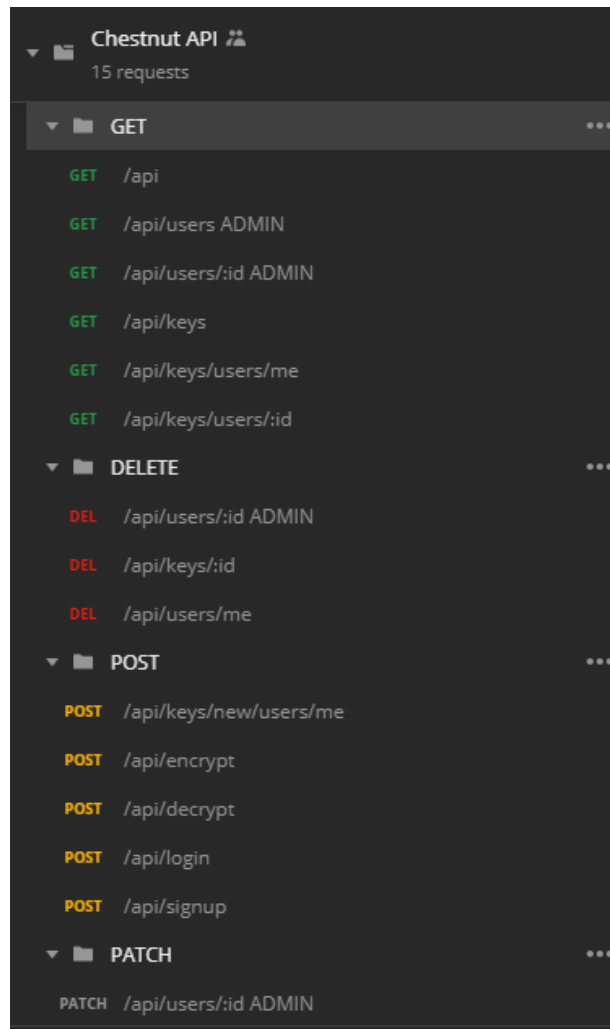


Figure 3-3 Overview of all the API endpoints, image taken from Postman

As seen in the figure above, Chestnut is using 4 of the most common HTTP verbs. Those are GET, POST, DELETE, and PATCH. These actions can perform the four basic CRUD operations of creating, reading, updating, and deleting resources.

Below is a figure of HTTP verbs side by side with their corresponding CRUD verb.

## HTTP Verbs and CRUD

HTTP Verb	Operation	URL Example
GET	Read	GET /api/project/1
POST	Create	POST /api/project
DELETE	Delete	DELETE /api/project/1
PUT	Update	PUT /api/project/1
PATCH	Update	PATCH /api/project/1

Figure 3-4 HTTP verbs compared to CRUD operations

(Bogdan, 2016)

Because REST has gained massive importance in the recent past, so has its popularity among businesses and developers. Many web developers have been attempting to implement REST APIs as best as they can, but it is often the case that this has resulted in APIs that are not truly RESTful.

For an API to be truly REST compliant, it must follow certain core principles defined by Roy Fielding in his dissertation. One of these constraints that are often overlooked is the implementation of HATEOAS.

### 3.4.1 Richardson Maturity Model

The Richardson Maturity Model is a way to categorize and grade APIs according to the official restraints of REST. The more the API adheres to these constraints, the higher its level will be. The model has 4 levels (0-3) where level 3 indicates a truly RESTful API. (What is the Richardson Maturity Model?, 2016).

Below is a figure of the 4 different maturity levels of an API.



Figure 3-5 Figure showing Richardson's maturity levels for REST APIs

(Fowler, 2010)

Level 3 APIs make use of **Hypermedia as the Engine of Application State**, which is what HATEOAS stands for. Hypermedia is a broader term in computer science and can refer to many media types such as graphics, video, audio, plain text, and hyperlinks. In the context of a web API and Chestnut, hypermedia refers to hypermedia links.

Most common web APIs will either be level 2, also known as REST-like APIs, or level 3, a truly RESTful API. This would be the correct way of categorizing APIs but a very common mistake for developers is to consider a level 2 API, a truly RESTful API.

This common mistake, often due to ignorance or negligence, has made Roy Fielding vocal on numerous occasions, expressing his frustration on how his work from his 2000 dissertation is often misinterpreted.

Below is a quote from Roy Fielding himself, from his personal blog:

*I am getting frustrated by the number of people calling any HTTP-based interface a REST API. [...]*

*What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. [...]*

(REST APIs must be hypertext-driven, 2008)

### 3.5 REST API constraints

REST defines 6 total characteristics which make any web service, a true RESTful API. Those characteristics are a uniform interface, client-server separation, stateless behavior, cache ability, layered system, and code on demand. (REST Architectural Constraints, 2018).

#### 3.5.1 Uniform interface

This is the feature that differentiates REST APIs from non-REST APIs. What this means is that there should be a uniform way of interacting with your API no matter the device or type of application that is making the requests.

Uniform interface is further categorized into 4 guidelines:

- **Resource-Based**

Individual resources and collections can easily be identified with requests.

For example: `/api/keys` or `/api/users/1`.

- **Manipulation of Resources Through Representation**

The client has a complete representation of a resource with all the necessary data to further manipulate that resource on the server, given he has the permission to do so.

For example: Usually user will get his key IDs too when requesting for a list of owned keys. That ID can then be used to delete the key.

- **Self-descriptive Messages:**

Each request from the client contains enough information for the server to be able to analyze and process the request correctly.

For example: This can be achieved by setting required fields in the body of the requests. Such as username being required when creating an account.

- **Hypermedia as the Engine of Application State (HATEOAS):**

The REST API includes links in each HTTP response, usually appended at the bottom. These links can be used by the client to discover other resources on the server more easily.

Below is a figure of a response from Chestnuts API, when requesting to GET /api/keys/users/me.

```
{
  "status": "200 - OK",
  "keypairs": [
    {
      "KeypairID": 5,
      "Name": "Test 1",
      "Length": 1024,
      "PublicKey": "-----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCczBoWKFArXhZK5tQ
      "PrivateKey": "-----BEGIN RSA PRIVATE KEY----- MIICXAIBAAKBgQCczBoWKFArXhZK5tQUjZb2BWLdHncffSI0H
      "createdAt": "2020-05-24T15:28:11.000Z"
    },
    {
      "KeypairID": 6,
      "Name": "Keypair",
      "Length": 1024,
      "PublicKey": "-----BEGIN PUBLIC KEY----- MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC20w1rt8j5ghaxHV7
      "PrivateKey": "-----BEGIN RSA PRIVATE KEY----- MIICXgIBAAKBgQC20w1rt8j5ghaxHV7Lh2A6gx6LqIS+7TK6b
      "createdAt": "2020-05-24T15:28:11.000Z"
    }
  ],
  "links": [
    {
      "self": {
        "method": "GET",
        "description": "Get all keys, public and private of currently logged in user.",
        "href": "http://localhost:8080/api/keys/users/me"
      }
    },
    {
      "method": "POST",
      "description": "Generate a new keypair for currently logged in user.",
      "href": "http://localhost:8080/api/keys/new/users/me"
    },
    {
      "deleteKeyByID": [
        {
          "method": "DELETE",
          "description": "Delete keypair with id 5",
          "href": "http://localhost:8080/api/keys/5"
        },
        {
          "method": "DELETE",
          "description": "Delete keypair with id 6",
          "href": "http://localhost:8080/api/keys/6"
        }
      ]
    }
  ]
}
```

Figure 3-6 Chestnut API response to GET /api/keys/users/me

The user making this request will receive his keys and in addition to them, Chestnut API also provides a links array with other related actions on the resource. These can be for example creating (POST) a new key to the collection or deleting (DELETE) an existing key resource from the collection.

The links that are a constraint of REST are represented as values to the href field in the JSON response.

### 3.5.2 Client-server

REST applications should have a client-server architecture meaning that both the client and server can function and evolve independently of each other.

The client's job is to only make requests and represent the results of those requests in UI form for the user to see and interact with. The server is the one that holds the resources internally and responds to requests accordingly. It should not be concerned with how the User Interface or website looks on the client and neither the user state on that website.

Chestnut follow this constraint by having implementations of both backend and frontend working separately on 2 different servers.

This is showcased in the following figure, with the API running on port 8080 while the frontend implementation is running on port 3000.

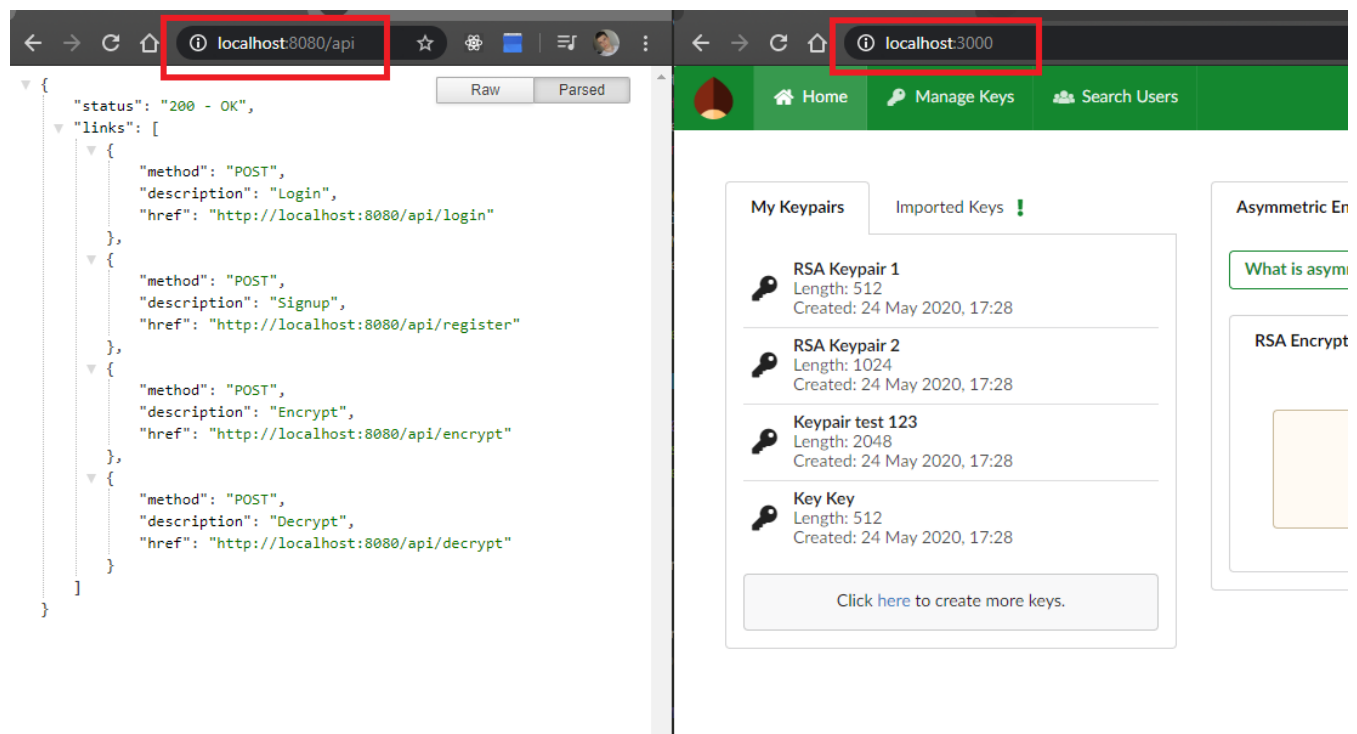


Figure 3-7 Figure showcasing both the backend and the frontend running simultaneously on different servers

### 3.5.3 Stateless

Professor Roy Fielding has also worked on HTTP and this can be seen in this constraint (Roy T. Fielding, 2012). This constraint implies that all interaction between the client and the REST API

should be stateless. The backend will not store anything about any of the requests and treat every single request as a new one.

Every single request needs to contain all the information necessary for the request to be serviced, including authorization details to access endpoints that require authentication and/or authorization.

Below is a figure showing a GET request to Chestnut. This request has a token attached to the Authorization field of the header that will be used to safely identify the user making the request.



Figure 3-8 Authorization token within the header portion of a GET request to Chestnut

This is a token that is only issued to a user upon successfully logging on the website.

### 3.5.4 Cache ability

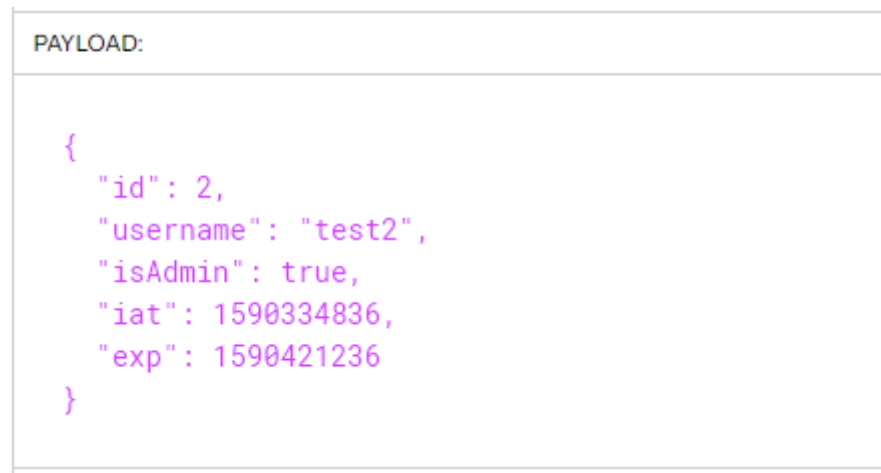
A cache is a component of software that will store data so that future requests will be served faster.

Whenever applicable and possible, caching of data is important. Caching brings performance improvements for the client and improves scalability because the load on the server is reduced.

The REST API must also specify whether the data it sends back is cacheable or not. If it is, it must send additional information the client will use to cache the data and know when it should update the data by making another request. These additional fields can be fields such as creation and expiration date.

When logging in Chestnut, the users receive a token they store locally. This token is cached in the browsers local storage so that users will stay logged in until the token is either deleted manually, the user logs out or the token expires.

Below is a figure of a token's payload issued by Chestnut:



*Figure 3-9 Authorization token payload contents*

In addition to some useful piece of information that will be used on the website, the API also sends back the fields “iat” and “exp”.

The “iat” field stands for the issue date while “exp” is the expiration date. By default, our tokens are valid for 12h.

### 3.5.5 Layered system

Between the client that makes the request for a resource and the server that sends back the response, there may be several additional servers in the middle. These servers are also known as layers and will provide different functionality. (Avraham, 2017).

Examples of such functionality are a security layer, a caching layer, or a load-balancing layer.

### 3.5.6 Code on demand

This constraint of the REST API architecture is entirely optional. Sometimes, instead of only sending static representations of resources in the form of JSON or XML, the API may return executable code in the form of a script. (REST Architectural Constraints, 2018).

## 3.6 Our solution

The Chestnut REST API adheres to all these constraints meaning we were successful in creating a truly RESTful API. The only exception to this is that Chestnut does not currently consist of multiple layered systems.



These systems can easily be implemented further down the line by either adding new functionalities or by separating entire portions of the current solution such as the data access layer.

### 3.6.1 Routes and controllers

The routes are the specific endpoints we have mentioned earlier, they are the address at which the client can make its requests.

Controllers are one form of middleware. This means that they are a method that will run after a request has been received, and before the response is sent back. Every route in Chestnut has a separate controller method that will validate the request, do the operations the user is asking for and send back the correct response.

Below is a figure showing how our backend is structured with separate folders for both routes and controllers.

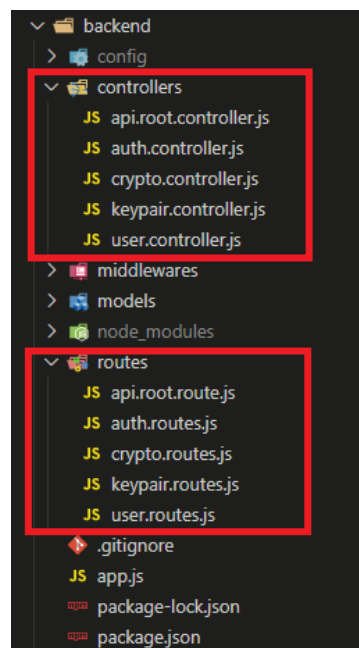


Figure 3-10 Controller and route folders in the backend directory

The next two figures show an example of a controller and its corresponding route in the routes folder. This route is responsible for deleting the currently logged in user.

```

// Delete currently logged in user
exports.delete = async (req, res, next) => {
  const userID = res.locals.decodedData.id;

  const url = req.protocol + '://' + req.headers.host;

  deleteKeys = await Keypair.destroy({ where: { UserID: userID } });
  deleteUser = await User.destroy({ where: { ID: userID } });

  // Check if user exists
  if (deleteUser == 0) {
    return res.status(404).json({
      status: '404 - Not Found',
      message: 'User was not found.',
    });
    // Delete user and his keys
  } else if (deleteKeys && deleteUser) {
    return res.status(200).json(
      {
        status: '200 - OK',
        message: 'User and associated keys deleted successfully.',
      },
      [
        {
          self: {
            method: 'DELETE',
            description: 'Delete currently logged in user.',
            href: url + '/api/users/me',
          },
        },
      ],
    );
  }
};

```

Figure 3-11 Controller that handles deletion of currently logged in user

```

module.exports = (app) => {
  const User = require('../controllers/user.controller');
  const router = require('express').Router();
  const jwtMiddleware = require('../middlewares/check.token');

  // Retrieve a single users info by id
  router.get('/:id', jwtMiddleware.checkToken, User.getUser);

  // Get all users and their info, admin only
  router.get('/', jwtMiddleware.checkToken, User.getAllUsers);

  // Delete currently logged in user
  router.delete('/me', jwtMiddleware.checkToken, User.delete);

  // Delete user by id, admin only
  router.delete('/:id', jwtMiddleware.checkToken, User.deleteByID);

  // Update an users info by id, admin only
  router.patch('/:id', jwtMiddleware.checkToken, User.updateUser);

  app.use('/api/users', router);
};

```

Figure 3-12 Delete currently logged in user route

Incoming requests from the client are also validated in the controller in addition to the client. It is not as common for the controller to detect issues in the requests as Chestnuts website is implemented in such way as to reduce these errors.

Below are a few figures with examples of the REST API returning errors to the user. The first is a POST request to the login route in which the user enters a wrong password for an existing user. Second figure is an error to the POST encryption route in which the user does not provide the API with all required fields.

```
1  {  
2    "status": "422 - Unprocessable Entity",  
3    "message": "Invalid Password."  
4  }
```

Figure 3-13 Figure showing a 422 status code response

```
1  {  
2    "status": "400 - Bad Request",  
3    "message": "Both fields are required and must be filled (text, publicKey)."  
4  }
```

Figure 3-14 Figure showing a 400 status code response

Status codes are an important part of both successful and failed requests, so we have spent a good amount of time ensuring that all our responses are given back with a fitting status code and message.

### 3.6.2 HTTP status codes

HTTP response status codes indicate whether a specific request has been successful or not. These status codes are categorized into:

- Informational responses (100-199)
- Successful responses (200-299)
- Redirects (300-399)
- Client errors (400-499)
- Server errors (500-599)

Within Chestnuts API, we are making use of the following codes:

HTTP status code	Status message	Description
200	OK	The request has succeeded.
201	Created	The request has succeeded, and a new resource has been created as a result.
400	Bad Request	The server could not understand the request due to invalid syntax.
401	Unauthorized	Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.
403	Forbidden	The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource.
404	Not Found	The server cannot find the requested resource.
422	Unprocessable Entity	The request was well-formed but was unable to be followed due to semantic errors.
500	Internal Server Error	The server has encountered a situation it does not know how to handle.

(HTTP response status codes, 2020)

### 3.7 Token middleware

We have talked about controllers being a form of middleware. We will now show an actual middleware we have implemented in our code. This middleware is responsible for checking if an authorization token has been attached to the header part of a request.

If the token is found in the header, the middleware will then check for the authenticity of that token and if the token belongs to the user making the request.

The middleware will run for every request made on the REST API.

Below is a figure showing the middleware method:

```

// Function that will check for valid token
const checkToken = (req, res, next) => {
  // Store token in variable
  let requestToken =
    req.headers["x-access-token"] || req.headers["authorization"];

  // Check if token was provided in request header
  if (!requestToken) {
    return res.status(403).json({
      status: "403 - Forbidden",
      message: "Provide a token to access this route.",
    });
  }

  // Remove Bearer from string if there is one
  if (requestToken.startsWith("Bearer ")) {
    requestToken = requestToken.slice(7, requestToken.length);
  }

  // If token is provided, verify it
  if (requestToken) {
    jwt.verify(requestToken, config.secret, (err, decodedData) => {
      if (err) {
        return res.status(401).json({
          status: "401 - Unauthorized",
          message: "Invalid token was provided.",
        });
      } else {
        res.locals.decodedData = decodedData;
        req.userData = { id: decodedData.id };
        next();
      }
    });
  }
};

```

Figure 3-15 Middleware function checking for authorization token

This method will return an error to the user if it finds missing or bad token or allow the request to be processed by its controller by passing it along with next().

### 3.8 Database models

By using Sequelize.js, we were able to easily write database models in JavaScript. These models are the blueprint for creating new table entries in the MySQL database. In our application we have users and keys.

Sequelize also allows to easily create an instance of a connection to a database, with your own credentials. This is shown in the figure below:

```
const dbConfig = require('../config/db.config.js');

const Sequelize = require('sequelize');

const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
  host: dbConfig.HOST,
  dialect: dbConfig.dialect,
  logging: false,
  define: {
    timestamps: false,
  },
  pool: {
    max: dbConfig.pool.max,
    min: dbConfig.pool.min,
    acquire: dbConfig.pool.acquire,
    idle: dbConfig.pool.idle,
  },
});
```

Figure 3-16 Connecting to the database using Sequelize

Based on the models we wrote, Sequelize will also generate new users and keys based on the following blueprints:

#### User

ID	Integer, auto increment, unique, Primary Key
Username	String, unique
Password	String
Email	String, unique
isAdmin	Boolean, false by default

#### Keypair

KeypairID	Integer, auto increment, unique, Primary Key
Name	String
Type	String, currently only allowing "RSA"
Length	Integer, currently only allowing 512, 1024 or 2048
PublicKey	Text
PrivateKey	Text
createdAt	Date in UTC format

## 3.9 API documentation

In this report, we have only partially documented the API with many smaller details such as exact behavior for every single route missing. Therefore, we have decided to document Chestnut API on Swagger, using the OpenAPI specification.

This will give the chance for anyone to explore the API in an alternate and much more approachable way using Swaggers website.

The API has been published and is available for viewing to anyone at <https://app.swaggerhub.com/apis-docs/konstapascal/chestnut-api/1.0#/>

### 3.9.1 OpenAPI specification

OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI file allows you to describe your entire API, including available endpoints, operation parameters input and output for each operation, authentication methods and other information. (Swagger documentation, 2020).

The specification can be written in YAML or JSON with the format being easy to learn and readable to both humans and machines.

### 3.9.2 Swagger

Swagger is a website consisting of open-source tools built around the OpenAPI specification. These tools can help you design, build, document, and consume REST APIs. We have written our documentation using the 3.0 version of the OpenAPI standard using the Swagger editor. The documentation code is written in YAML. (Swagger documentation, 2020).

Below is a figure showing the Swagger editor with YAML code for the documentation.

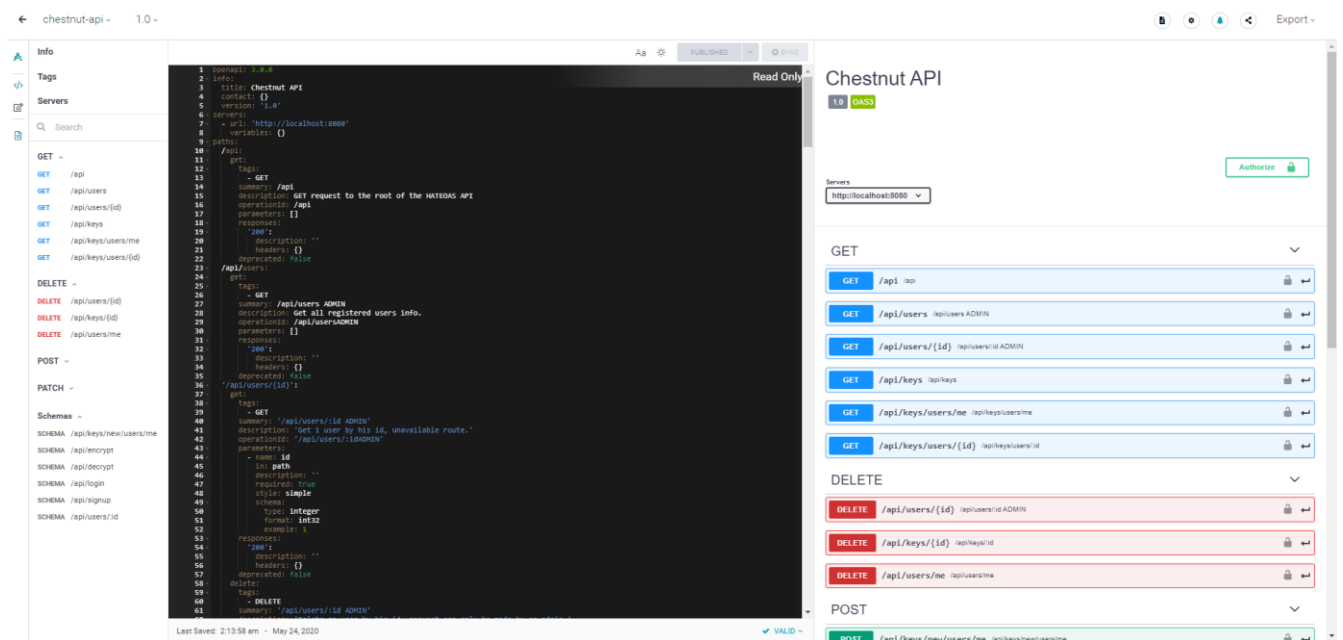


Figure 3-17 Figure showing the Swagger Editor

The end result of this documentation is a published API that be reached and viewed at <https://app.swaggerhub.com/apis-docs/konstapascal/chestnut-api/1.0#/>

Swagger creates an interactive page of your REST API in which the user is presented with an overview of all the endpoints. These endpoints are expandable to explore each and one of them into further detail.

Below is an image of all our endpoints taken from Swaggers autogenerated publishing page:



GET		▼
GET	/api /api	🔒
GET	/api/users /api/users ADMIN	🔒
GET	/api/users/{id} /api/users/id ADMIN	🔒
GET	/api/keys /api/keys	🔒
GET	/api/keys/users/me /api/keys/users/me	🔒
GET	/api/keys/users/{id} /api/keys/users/id	🔒
DELETE		▼
DELETE	/api/users/{id} /api/users/id ADMIN	🔒
DELETE	/api/keys/{id} /api/keys/id	🔒
DELETE	/api/users/me /api/users/me	🔒
POST		▼
POST	/api/keys/new/users/me /api/keys/new/users/me	🔒
POST	/api/encrypt1 /api/encrypt	🔒
POST	/api/decrypt /api/decrypt	🔒
POST	/api/login /api/login	🔒
POST	/api/signup /api/signup	🔒
PATCH		▼
PATCH	/api/users/{id} /api/users/id ADMIN	🔒

Figure 3-18 Overview of all endpoints taken from the published page of Chestnut API on Swagger

### 3.10 Conclusion

Despite facing some difficulties during development and having to learn new concepts such as HATEOAS, we are very satisfied with the result. We know that our assignment giver is also satisfied as having HATEOAS implemented was one of his optional requirements.

We managed to build a truly RESTful API that adheres to all the REST architectural constraints and have it function completely independent of any front-end implementation. This way, the REST API is very flexible by being able to serve not only any front-end implementation but also multiple at the same time.

## 4 Frontend documentation

Designing and developing the frontend of our application was what we spent the most time on. It was important for us to have a design that captures and improves the original chestnuts functionalities. In this section we will cover what frontend means in web-development, our technology choices, and how we designed and developed the website.

## 4.1 Describe what frontend is and what does it consist of?

“Frontend web development is the practice of converting data to a graphical interface, using HTML, CSS, and JavaScript, so that users can view and interact with that data. ” (Front-end web development, 2020)

Users directly interact with unique aspects of a frontend program, including inputs, buttons, links, and other features. When designing and developing the frontend, it is essential to take accessibility and users’ ease of use into consideration.

## 4.2 Technology choices

From the start of the project, we needed to make important decisions. The question was which frontend- and CSS-frameworks were most suitable for us. We will go over the choices we made and the reasoning behind them.

### 4.2.1 Frontend framework

The project requirements stated that we needed to use either AngularJS or React as our frontend framework. After evaluating the options, we chose React. What influenced our decision was the fact that we had some prior experience working with React and us not wanting to learn a new framework in the small timeframe we had.

### 4.2.2 React

React is not truly a framework because it does not provide default state management, routing, and data fetching which are important aspects of other frameworks. Instead, it has left these things to the developer community. (facebook/react, 2020)

As a result, the community has developed many libraries that complement React. These extra libraries transform React from a UI library to a complete framework.

React also has some distinct features like React components, virtual DOM, and JSX. Because grasping these features is essential to understanding how our user interface works, we will explain them briefly.

#### 4.2.2.1 *React components*

In React, we divide the user interface into smaller, logical parts. Each part is what we call a component. The idea behind components is that instead of having two separate files for markup and logic, React separates concerns by coupling them both in one unit which we call a component. (Introducing JSX, 2020).

In the provided figure, the navigation bar at the top, the key pair list on the left, the text boxes in the middle, and the generate a key pair form are all different components that we have marked with red squares. Depending on preference, these components can be divided into even smaller components.

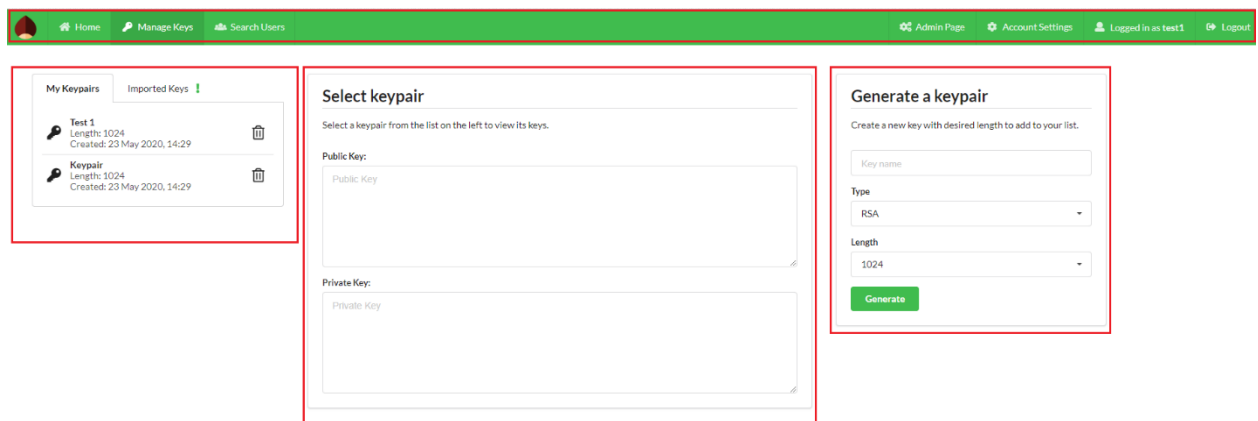


Figure 4-1 Chestnut manage keys components layout

The “My Keypairs” contains two components that we have marked. The first one is the container which holds all the item lists while the second component is a list item. This is a logical approach because we need one container which can hold many items.

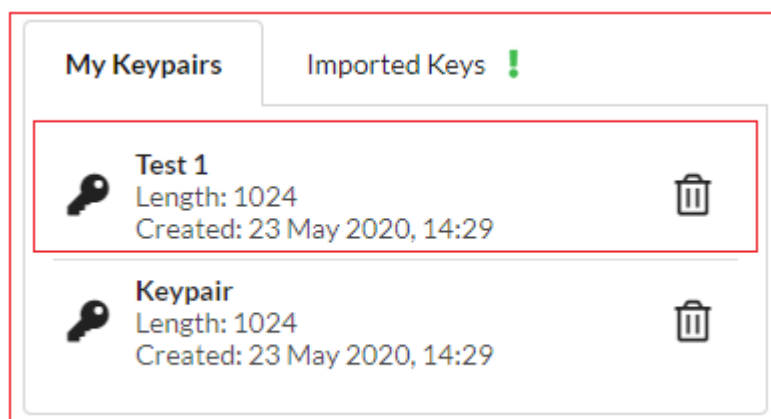


Figure 4-2 Chestnut Keys List component layout

#### 4.2.2.2 JSX

JSX stands for JavaScript XML. JSX allows us to write HTML in React because it converts the HTML element into React elements. (React JSX, n.d.)

When JSX is compiled into JavaScript. It can run faster than if it was directly written in JavaScript. JSX also offers a class-based system that developers can use. (JSX - a faster, safer, easier JavaScript, 2013).

#### 4.2.3 Creating new React app

Before we can create a new React application, we need to have Node installed on our computer. The Node run-time environment lets us execute programs written in JavaScript. The Node package manager (npm) allows us to install and update packages that we can use in our app to be more efficient. (Patel, 2018).

We create a React application using the VS Code terminal by typing the following commands.

```
aboz@DESKTOP-CTAIP50 MINGW64 ~/Documents
$ mkdir chestnut2020spring

aboz@DESKTOP-CTAIP50 MINGW64 ~/Documents
$ cd chestnut2020spring/

aboz@DESKTOP-CTAIP50 MINGW64 ~/Documents/chestnut2020spring
$ npx create-react-app chestnut
[#####.....] / extract:lodash: sill extract lodash@4.17.15 extracted to C:\Users\aboz\AppData\Roaming\npm-cache\npx\20616\node_modules\staging\lodash-83c0e996 (955ms)
```

*Figure 4-3 Creating a folder named chestnut2020spring and creating template of a react app*

Firstly, we create an empty folder that we name “chestnut2020spring”. Inside this empty folder, we create our React app that we call “chestnut” and npm will automatically install all the necessary packages that we would need to run this app. A successful React app creation gives the following message.

```
Success! Created chestnut at C:\Users\aboza\Documents\chestnut2020spring\chestnut
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd chestnut
  npm start

Happy hacking!
```

Figure 4-4 Terminal output after create-react-app has finished downloading all files

4.2.4 Additional packages

We have added several packages throughout the development of the frontend. We can find the installed packages by writing “npm ls --depth=0” in the terminal.

```
Konstantinos@DESKTOP-DU5NUVM MINGW64 /c/vsc_repos/chestnut/
$ npm ls --depth=0
frontv2@0.1.0 C:\vsc_repos\chestnut\frontend
+-- @tippy.js/react@3.1.1
+-- axios@0.19.2
+-- jwt-decode@2.2.0
+-- moment@2.24.0
+-- react@16.13.1
+-- react-dom@16.13.1
+-- react-router-dom@5.1.2
+-- react-scripts@3.4.1
+-- semantic-ui-css@2.4.1
+-- semantic-ui-react@0.88.2
+-- uuid@8.0.0
```

Figure 4-5 Overview of installed packages for the frontend part of Chestnut

Package	Description	Source

68

Tippy	Tippy.js is a tooltip solution for the web. We used tippy to add explanations for cryptography terms on the website.	<a href="https://github.com/atomiks/tippyjs-react">https://github.com/atomiks/tippyjs-react</a>
Axios	Axios is a library used to send HTTP requests from node.js or XMLHttpRequests from the browser. Axios supports the ES6 Promise API.	<a href="https://github.com/axios/axios">https://github.com/axios/axios</a>
Jwt-decode	Jwt-decode is a small browser library that helps to decode JWTs token which is Base64Url encoded.	<a href="http://github.com/auth0/jwt-decode">http://github.com/auth0/jwt-decode</a>
Moment	Moment is a lightweight date library for parsing and validating dates. We use moment to add date information to a user-generated key pair.	<a href="https://github.com/moment/moment">https://github.com/moment/moment</a>
React-dom	React and React-dom are installed when we initially create a new React app. React-dom renders components on the website.	<a href="https://www.npmjs.com/package/react-dom">https://www.npmjs.com/package/react-dom</a>
React-router	React-router adds the core functionality of routing in React.	<a href="https://github.com/ReactTraining/react-router">https://github.com/ReactTraining/react-router</a>
Semantic-UI-React	Semantic-UI-react is a CSS framework that we used to add style and responsiveness to our components.	<a href="https://github.com/Semantic-Org/Semantic-UI-React">https://github.com/Semantic-Org/Semantic-UI-React</a>
UUID	UUID is a lightweight library that we used to generate UUIDs.	<a href="https://github.com/tc39/proposal-uuid">https://github.com/tc39/proposal-uuid</a>
Node-forge	Node-forge is an implementation of cryptographic tools in JavaScript. We used this library to implement all the cryptography tools on the Chestnut website.	<a href="https://github.com/digitalbazaar/forge">https://github.com/digitalbazaar/forge</a>

#### 4.2.5 CSS Framework

CSS (cascading style sheets) is a language that describes how an HTML document should be styled and displayed (CSS Tutorial, n.d.). When developers start on a new project, they have two choices. They can either start from a blank document or use a CSS framework which can quickly create a user interface that can be edited throughout the project.

We can say that a CSS framework is a collection of prescribed stylesheets that we can use without having to write them ourselves. We investigated several popular frameworks that we could use with our project. Eventually, we had to choose between Bootstrap and Semantic-UI. We had used these frameworks before and had a good understanding of how each of them worked. We knew that

either of these two could work well within our project and we also did not need to spend time figuring out a new framework.

In the end, we decided to use Semantic-UI. Firstly, Semantic-UI has officially integrated React into Semantic-UI which is called Semantic-UI-React. This React integration comes with some features like declarative API, augmentation, and sub-components that make using this framework quite intuitive. Secondly, it is a well-documented framework. And lastly, it gives us the ability to develop a responsive solution without doing too much extra work.

We have used these features throughout the project and therefore will give a quick explanation with a code example for each one of them. It is important to note that while we have used the features that we are going to cover, some of the code examples are taken directly from the Semantic-UI documentation for explanation purposes.

The word declarative in programming means that we write some code that describes what we want, but we do not necessarily give a step by step instruction. The example below shows how we declared a button in JSX and the corresponding rendered HTML.

```
<Form.Button
  type='button'
  style={{
    backgroundColor: '#14872f',
  }}
  onClick={generateUuid}
>
```

Figure 4-6 Declaring a button in JSX

```
<button
  type='button'
  class='ui button'
  style='background-color: rgb(20, 135, 47);'
>
  <p style='color: rgb(255, 255, 255);'>Generate</p>
</button>
```

Figure 4-7 Corresponding button rendered in HTML from the previous JSX button

Augmentation lets us render one component as another component. This feature was especially useful when we worked with “MenuLinks” because it removes the need of adding nested components.

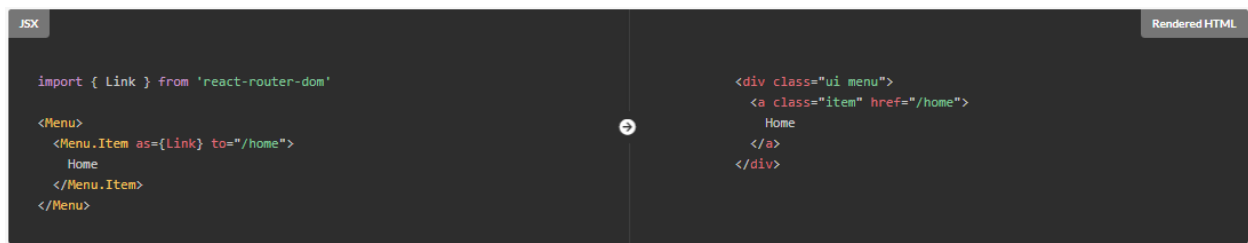


Figure 4-8 Making use of augmentation to declare a menu item as a link

A menu item declared as a link that points to the “home route.”

Sub-components let us quickly create customized components.



Figure 4-9 Subcomponent showcase in JSX

## 4.3 Design Choices

From the very beginning, we had a user-centered approach to designing the UI/UX. Since our primary users are students who may or may not be familiar with cryptography concepts, we designed and developed Chestnut to not only look good but be user friendly. In addition, we emphasize on keeping the features as simple as possible to not overwhelm the users.

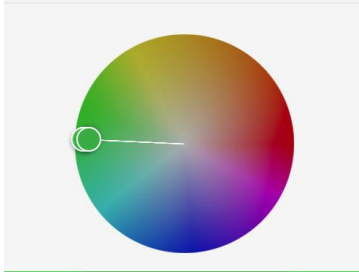
In this section, we will cover different parts of the Chestnut application and the thought process behind our choices.

### 4.3.1 Color theory

Color theory in web design refers to how developers and web designer can choose colors which work well together. The colored circle that we see in the figure below is called an RGB color wheel. To select a good color scheme, a designer can choose between commonly accepted structures such as triadic, compound, or analogous.

We used Adobe Color web application to explore and decide what color scheme we wanted to use.

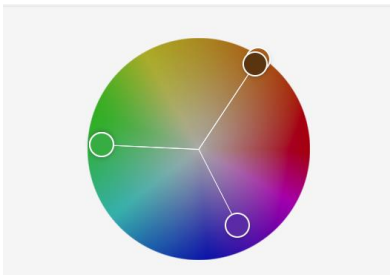




*Figure 4-10 Analogous color wheel*



*Figure 4-11 Compound color wheel*



*Figure 4-12 Triadic color wheel*

Analogous, compound, triadic color wheel for the RGB color #16AB39 from left to right

The main reason we wanted to use green as our primary color was because it reminded us of chestnut trees. We examined the different methods provided by Adobe Color to explore different schemes for green. We ended up using the triadic method because it provided us not only with green, but also brown which is the color of a chestnut.

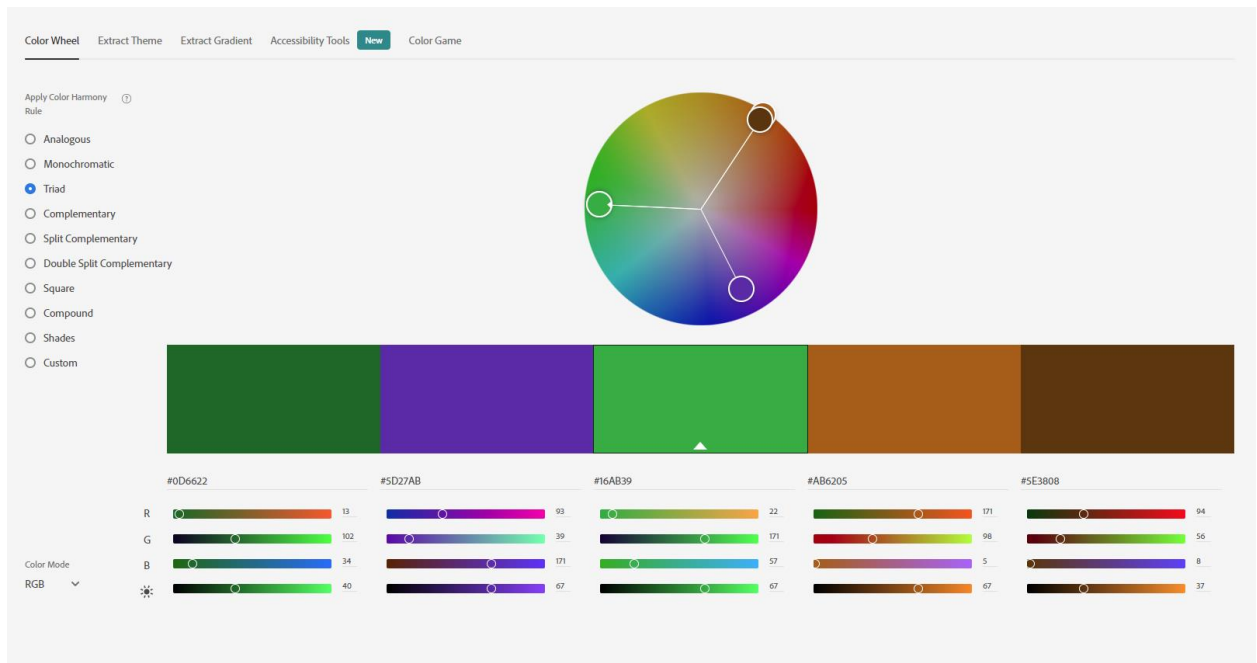


Figure 4-13 Color picker in Adobe Color

### 4.3.2 Navigation Bar

The navigation bar or the navbar is one of the most important elements of a website. Navbars let users navigate through the different sections of a website and therefore, is an essential part of every UI/UX design process.



Figure 4-14 The whole navbar of Chesnut web app



Figure 4-15 Zoomed in on the left side of the navbar



Figure 4-16 Zoomed in on the right side of the navbar being logged in as admin

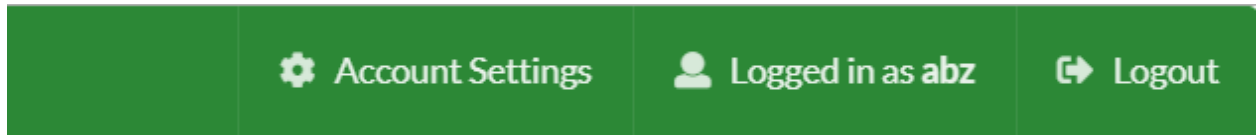


Figure 4-17 Zoomed in on the right side of the navbar being logged in as normal user

Our design focuses on simplicity, legibility, and clarity. To achieve simplicity and clarity, we use a solid color for the background, and we create navbar links only for the distinct parts of the application. Also, we present the users with additional sub-navbars where it is necessary.

We provide users a visual orientation to help them know where they are on the website all the time. We do this by highlighting the currently selected item in the navbar.

Another important factor in a good navbar design is having high contrast between the background- and the font color. High contrast is essential to attaining an easily readable navbar. The Web Content Accessibility Guidelines (WCAG) states that small texts should at least have a contrast ratio of 4.5:1 while large texts should have a ratio of 3:1.

We checked our navbar's contrast by using the "tanaguru contrast finder" which is an online tool to make sure the colors we use work together. The contrast ratio between the colors we use is 4.63 which means they are valid colors to use.

Foreground Color :

For each color (red, green and blue), enter a number between 0 et 255.

Red :  Green :  Blue :

The color should be between #000000 and #FFFFFF

Hexadecimal :

Background Color :

For each color (red, green and blue), enter a number between 0 et 255.

Red :  Green :  Blue :

The color should be between #000000 and #FFFFFF

Hexadecimal :

Minimum ratio :

In the international reglementation established by the [WCAG](#), [the success criteria 1.4.3](#) requires a minimum contrast ratio of 4.5:1 (and 3:1 for enlarged text).

This minimum contrast ratio is also required by the French regulation, established by the [RGAA 3.0 2016](#), in [the criteria 3.3 et 3.4](#).

Component to edit :

☒ Edit the foreground color

☐ Edit the background color

Gimme :

☒ valid colors and *very close* to initial color

☐ a *range* of valid colors

**Check and find contrast**

**Valid Contrast : 4.62924**

The choosen colors are already good

Figure 4-18 Checking the contrast between white text on our shade of green

### 4.3.3 Sub-navbars

We have added additional navbars to the cryptography section of the application to make navigating between different components easier and more intuitive. First, we have divided all the cryptography components into three meaningful categories. Asymmetric encryption, symmetric encryption, and utilities.

After selecting a category, the user can already perform an action or choose another action from the newly presented sub-navbar in each category.

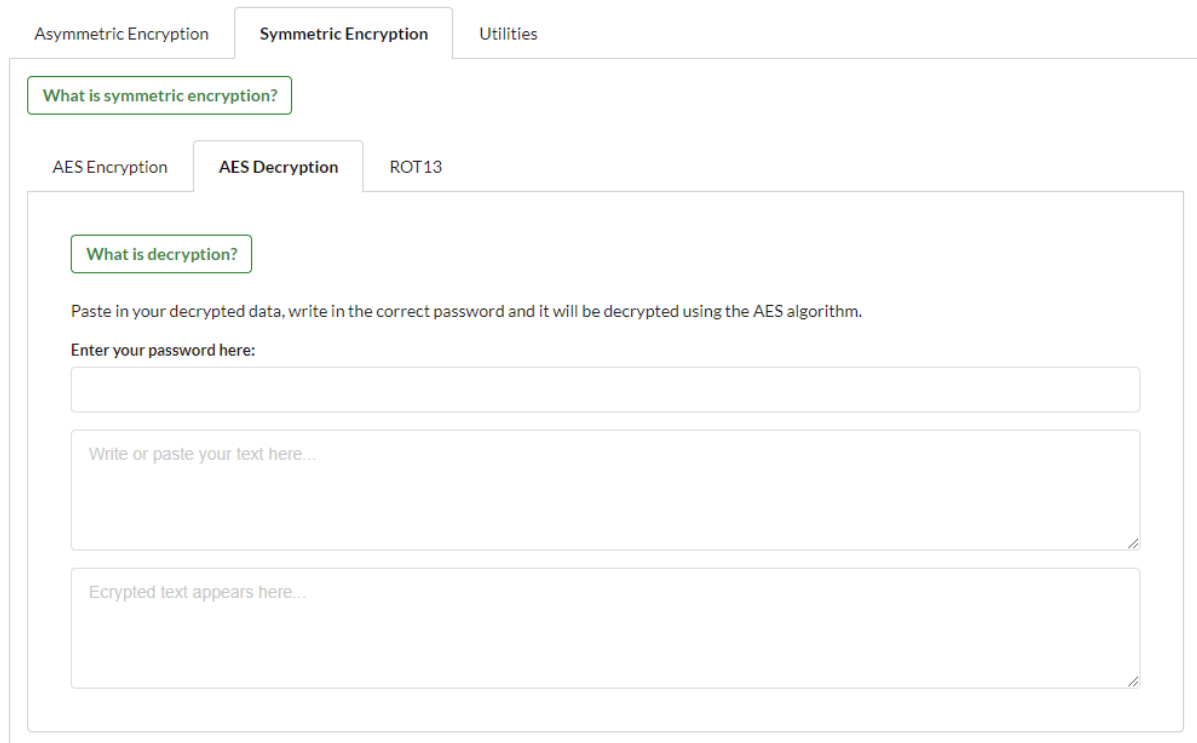
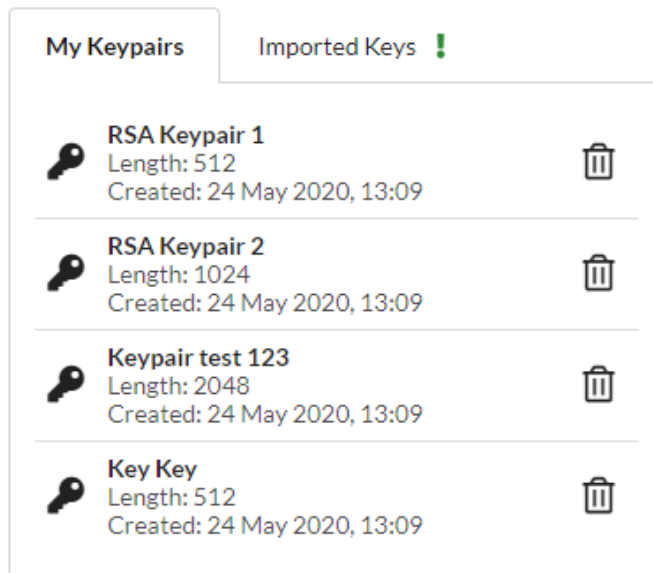


Figure 4-19 Figure of sub-navbars on the home page of the application

### 4.3.4 Icons

Icons help draw the attention of the users as well as increasing readability (MERTZ, 2012). We use icons for different purposes in the Chestnut website. In the navigation bar, icons help with readability and to add more context to what each navbar link. We also use icons instead of buttons in some places.

The trash icon is often associated with deleting or throwing away something. We make use of this information to create a clean looking component.

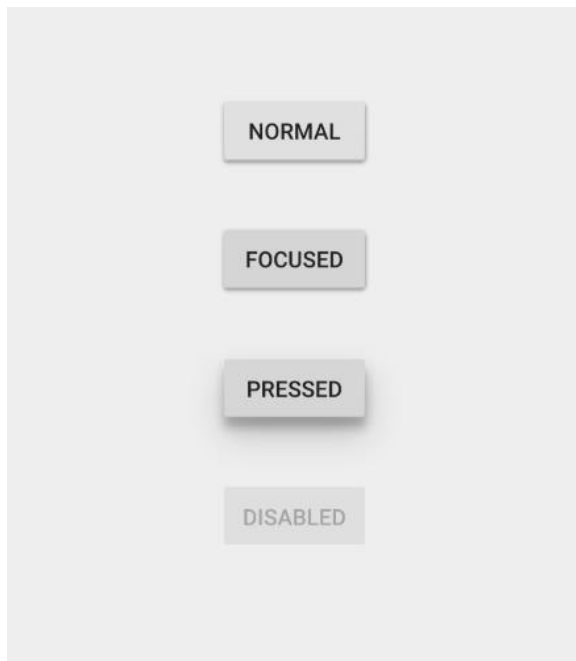


*Figure 4-20 Keys List with small trash icon indicating the possibility to delete a key*

#### 4.3.5 Buttons

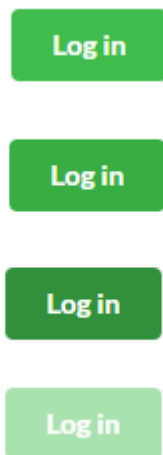
Buttons and the order in which they appear are important. There are also several different types of buttons like flat, raised, toggle, and ghost buttons and each one of them behaves differently (Babich, 2016). We use raised buttons throughout the website.

A raised button has a rectangular shape and displays to the users that they can click on it. In the following figure we can see how a raised button behaves in different states.

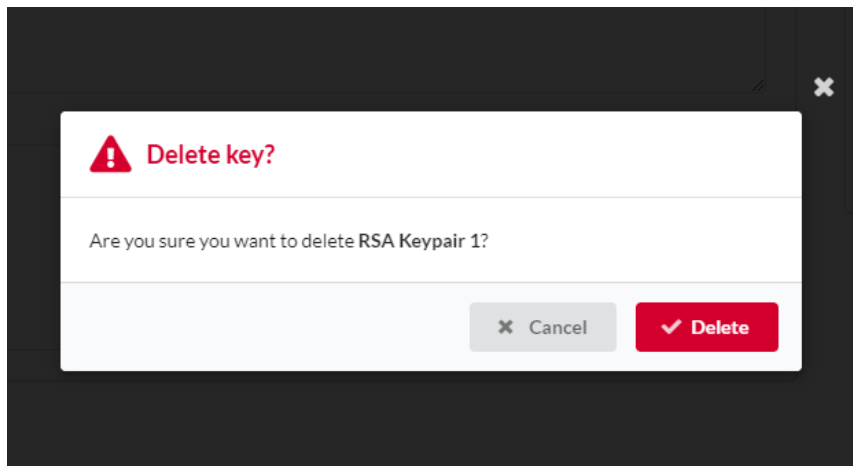


*Figure 4-21 Different states a button can be in*

(Babich, 2016)



*Figure 4-22 Different states of a button in the Chestnut solution*



*Figure 4-23 Modal warning the user before deleting a key*

The order of the buttons is also important. When the users perform an action like deleting a key pair that will have permanent consequences, we ask them to confirm their actions. We place the primary action on the right with high contrast and strong colors.

#### 4.3.6 Tooltips

The main reason for why we want to include tooltips is because it adds even more value for our users who are the students. Tippy.js gives us the ability to create and use tooltips. We use tooltips to add explanations for cryptography terms that students may not know. This way, they do not need to spend time searching terminologies.

Tooltips can be used in many other contexts like providing instruction on how to use a specific part of the application or just display info about an item.

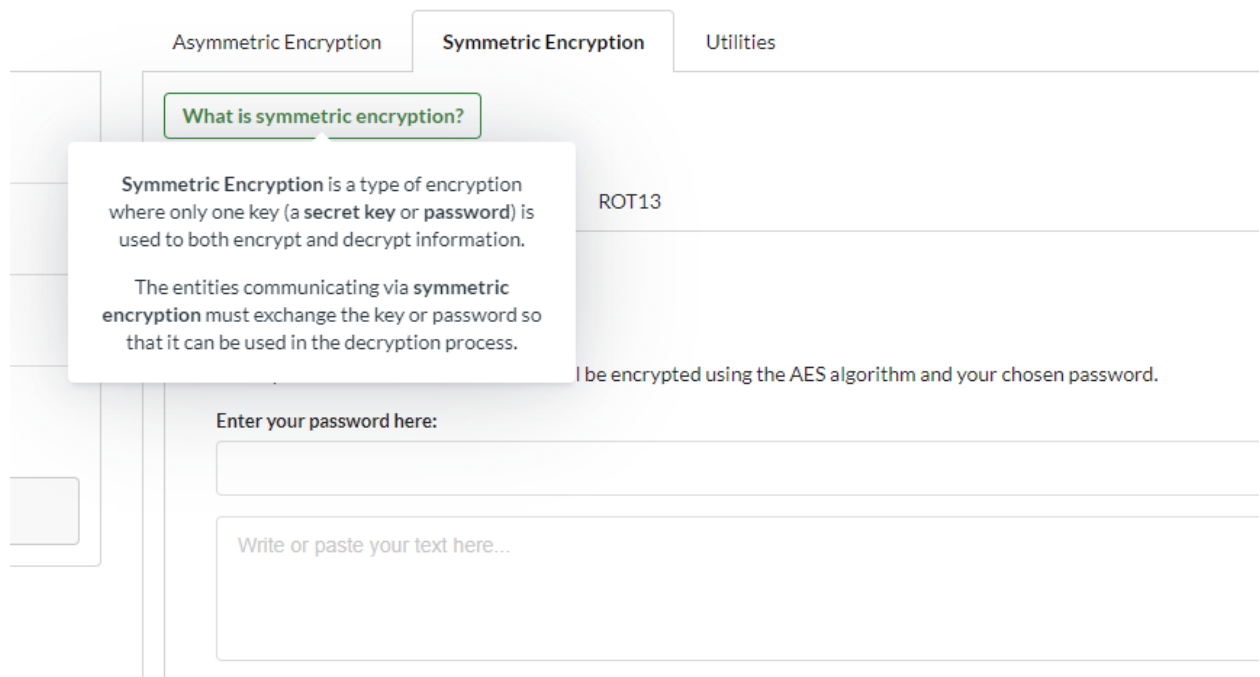


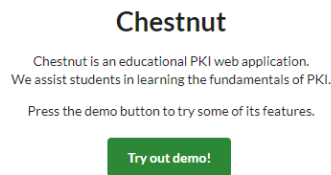
Figure 4-24 Educational tooltip with definition for symmetric encryption

## 4.4 Chestnut Web Application

We built the Chestnut web application based on the original JavaFX Chestnut application that Thomas Sødning, an associate professor for OsloMet had developed to teach his students the principles behind PKI.



### 4.4.1 Landing Page



### Log in to your account

Username

Password

Log in

Don't have an account? [Signup](#)

Figure 4-25 Chestnut landing page

The landing page consists of two main parts. The left section is a quick introduction to what chestnut is. Besides, it contains a call to action button. By clicking on the “Try out demo!”, a demo of the application replaces the description. In the demo component, users can explore Symmetric encryption as well as base64, UUID generator, and checksum generator without logging in.

Symmetric EncryptionUtilities

What is symmetric encryption?

AES EncryptionAES DecryptionROT13

What is encryption?

Write plain text in the first area and it will be encrypted using the AES algorithm and your chosen password.

Enter your password here:

Write or paste your text here...

Encrypted text appears here...

Back

### Log in to your account

Username

Password

Log in

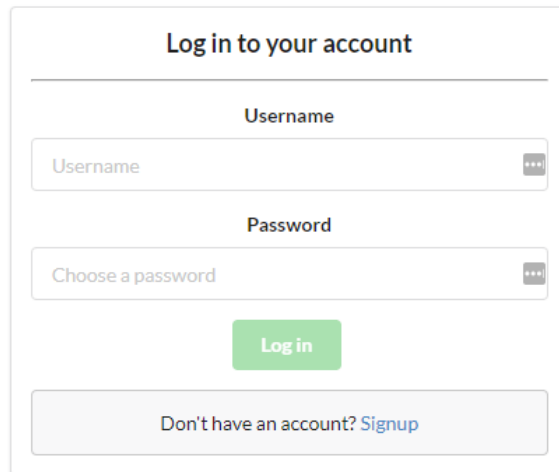
Don't have an account? [Signup](#)

Figure 4-26 Chestnut landing page with demo

80

#### 4.4.1.1 Logging in

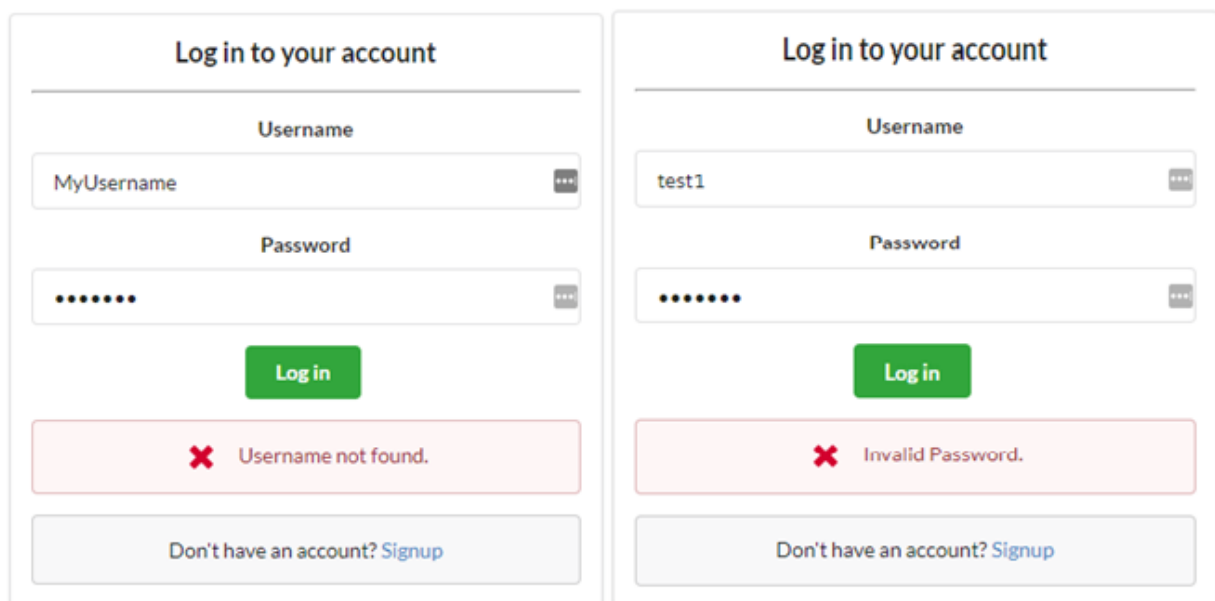
Users that have previously created accounts can log in by using their usernames and passwords. We have a disabled login button that only activates if both the username and the password fields are valid.



The login form is titled "Log in to your account". It contains two input fields: "Username" with a placeholder "Username" and "Password" with a placeholder "Choose a password". Both fields have a toggle icon on the right. Below the fields is a green "Log in" button. At the bottom, there is a link: "Don't have an account? [Signup](#)".

Figure 4-27 Log in box for Chestnut

After a user enters their credentials and clicks on the login button, the application sends an API request to compare the entered data against the database. The request first checks if a user with the given username exists. If the user exists, then it checks the password. If both the username and password match a record in the database, an authentication token will be created, and the request will be successful.



Two side-by-side login forms are shown, both titled "Log in to your account". The left form has "MyUsername" in the Username field and "\*\*\*\*\*" in the Password field. Below the fields is a green "Log in" button. A red error message box at the bottom says "✖ Username not found." The right form has "test1" in the Username field and "\*\*\*\*\*" in the Password field. Below the fields is a green "Log in" button. A red error message box at the bottom says "✖ Invalid Password." Both forms have a link at the bottom: "Don't have an account? [Signup](#)".

Figure 4-28 Errors for invalid credentials entered by the user

### 4.4.2 Homepage

After a successful login, the users are redirected to the homepage. From the homepage, the user can navigate to different sections of the application. Users can easily log out by clicking on the log out button in the top right corner of the page. When a user logs out, they are redirected to the landing page.

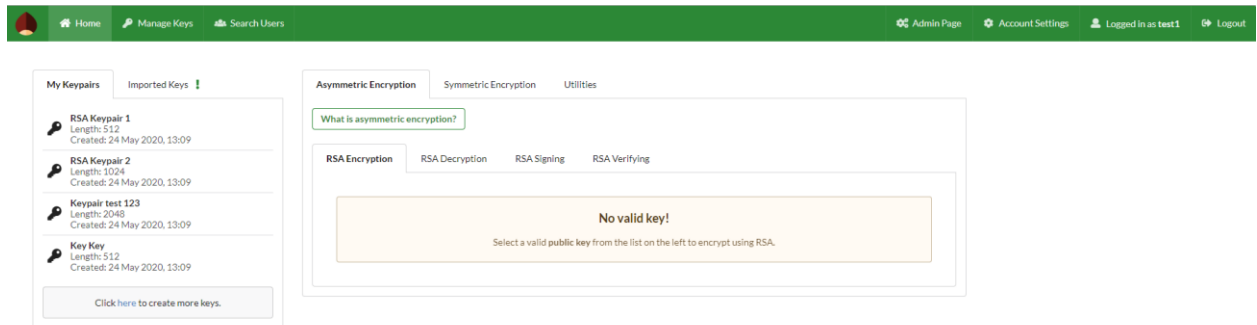


Figure 4-29 Chestnut homepage for logged in users

### 4.4.3 Manage Keys

This page has three parts that we will look at one by one.

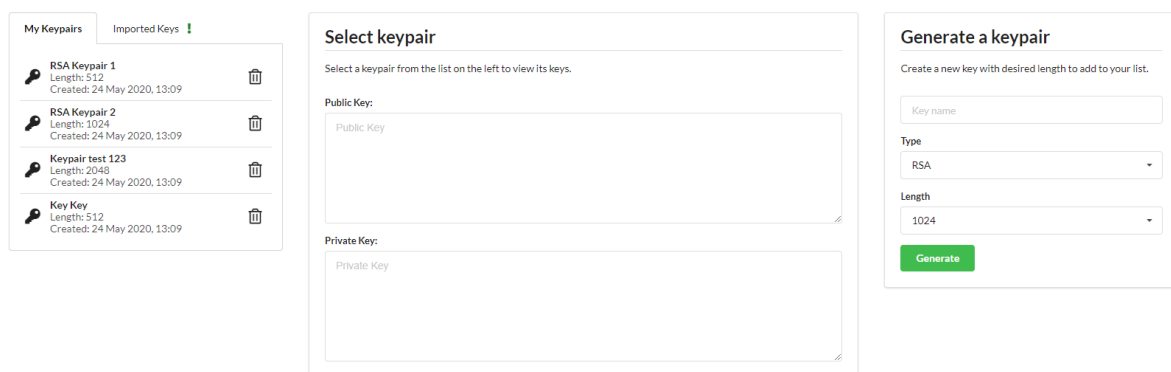


Figure 4-30 Manage keys page in Chestnut

#### 4.4.3.1 Admin page

The admin page is only accessible to users with admin privileges. An admin can see a complete list of users and can delete any user.

# Administrator settings

Overview of all registered users and possibility to manually delete select users.






Filter users		Q
	<b>test2</b> id: 2 test2@gmail.com	Delete user
	<b>thomas</b> id: 3 thomas@gmail.com	Delete user
	<b>konsta</b> id: 4 konstapascal@gmail.com	Delete user
	<b>test3</b> id: 5 test3@gmail.com	Delete user
	<b>abozar</b> id: 6 abozar@gmail.com	Delete user

Figure 4-31 Admin settings page in Chestnut

Clicking on the “Delete user” will prompt a modal component so users can confirm their action before application commits to change anything. If the admin confirms that he wants to delete a user, the user and all their keys will be deleted indefinitely.

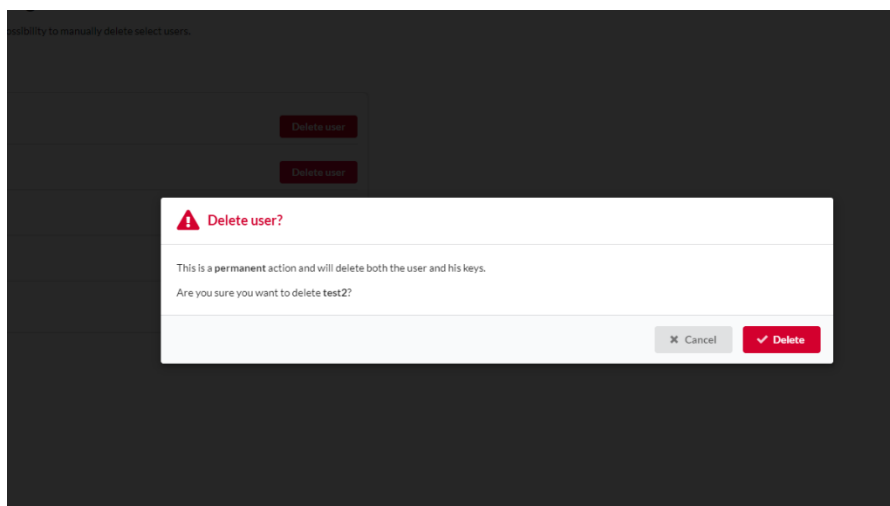


Figure 4-32 Modal warning the admin before deleting an user

#### 4.4.3.2 Account setting

The account setting page right now, consists of one option. A user can decide to delete their account at any given time. If a user deletes his account, all the records of him will also be deleted from our database.

## Account settings

Delete your account permanently, erasing all personal info and associated data such as your keys from the database.

Delete account

Figure 4-33 Account settings page in Chestnut with option to delete your account

If a user clicks on “Delete account”, the application props them with a confirmation box. If the user clicks on delete now, their account is deleted forever.

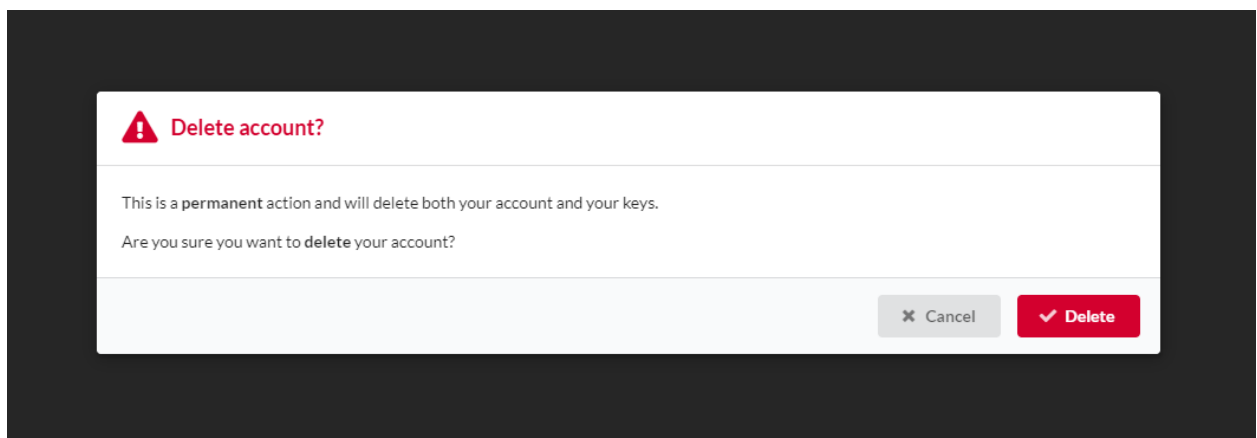


Figure 4-34 Modal warning the user before permanently deleting their account

#### 4.4.3.3 Generate a keypair

To generate a new keypair, a user needs to enter a name and choose the length of the key from the selector. When the user clicks on the “generate” button, if the provided name is valid, a new key pair will be generated and automatically added to the “My keypairs” list.

### Generate a keypair

Create a new key with desired length to add to your list.

Type

RSA

Length

2048

Generate

✓

 Keypair MyKeyPair was generated successfully!

Figure 4-35 Generate keypair component with successfull message

### Generate a keypair

Create a new key with desired length to add to your list.

Type

RSA

Length

1024

Generate

✗

 Name cannot be empty!

Figure 4-36 Generate keypair component with error message

## My Keypairs/Imported Keys

A user can choose to see their own generated keypairs or the public keys they have imported from other users.

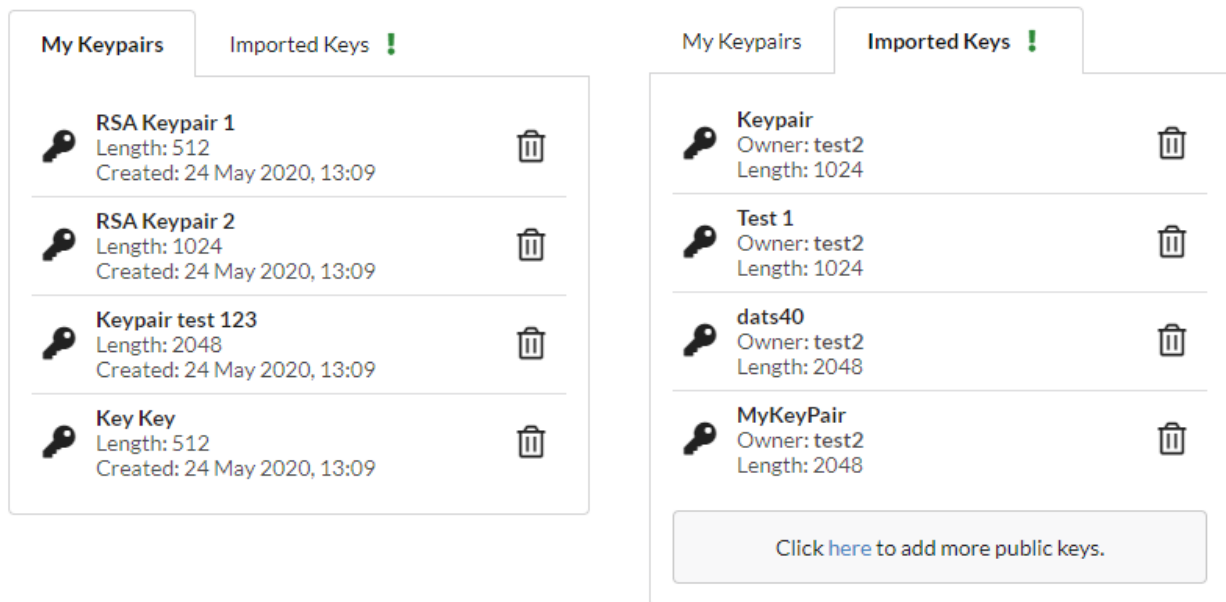


Figure 4-37 My keypairs list and imported keys list

To delete a key, the user can click on the trash icon for that specific key. After clicking on the icon, the application asks the user to confirm the action. If the user chooses to delete the key, the key is removed from the database via an API request and the key list automatically updates.

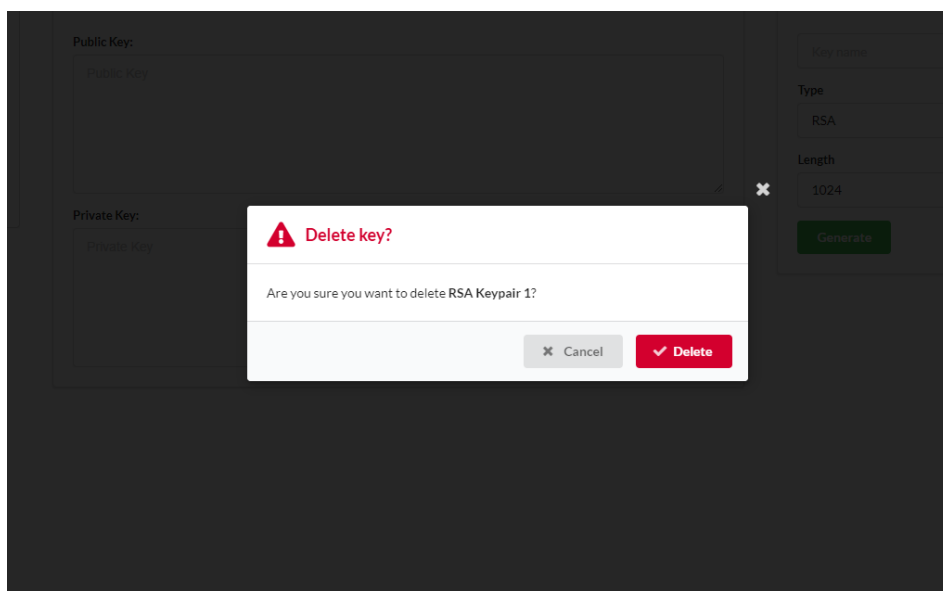


Figure 4-38 Modal warning the user before permanently deleting a keypair

#### 4.4.3.4 View key

The middle section of the page is where a user can view the content of a keypair they own or another user's public keys. To choose a key, the user simply clicks on one of the keys from the "My Keypairs" or "Imported Keys". The application highlights the selected key and displays the content on the existing text areas.

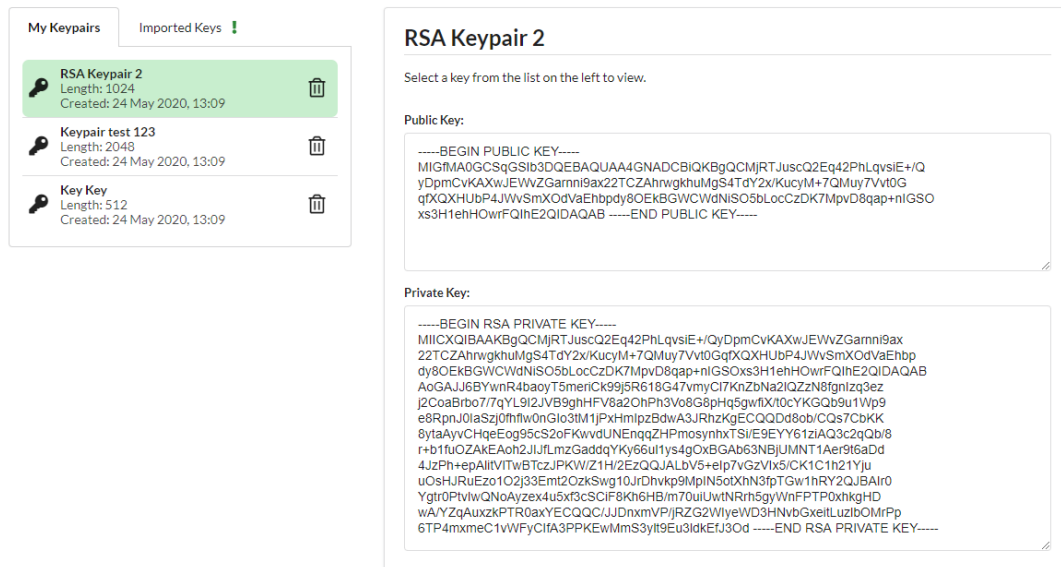


Figure 4-39 Selecting a keypair to show its contents: public and private key

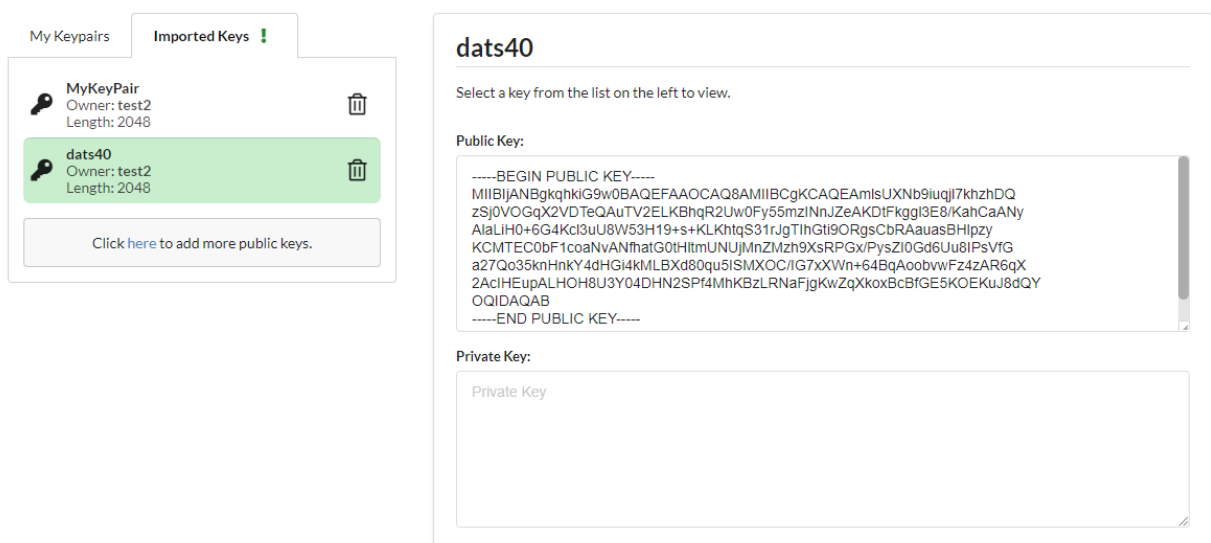


Figure 4-40 Selecting imported key to view its public key




#### 4.4.4 Search Users


A user can search for other users via the search box and easily import their public keys by clicking on a button. The button's label changes to let the user know that the key has been added to the imported-keys list.


### List of users


This is a list of all registered users and their public keys. You may filter users and easily add their public keys to your keys list.




test2

 **Keypair**  
Length: 1024

 **Test 1**  
Length: 1024

 **dats40**  
Length: 2048

 **MyKeyPair**  
Length: 2048

Add

Add

Added


Added

Figure 4-41 List of registered users filtered by a search bar

The application will inform the user if the search term does not match any records.

### List of users

This is a list of all registered users and their public keys. You may filter users and easily add their public keys to your keys list.




 No user results for AnotherUsername.

Figure 4-42 List of users showing no results for a search

#### 4.4.5 Asymmetric Encryption

The asymmetric encryption components will only be displayed after the user chooses a key that they have generated or a key they imported. After choosing a key, the selected key will be highlighted.

The user can type or paste text in the text area and click on the “Encrypt” button. It will send an API request to encrypt the message. If the request is completed, the application displays the request’s response which is the encrypted message, and lets the user know the encryption was successful. The decryption works the same except it expects encrypted data as input and will display plain text as output.

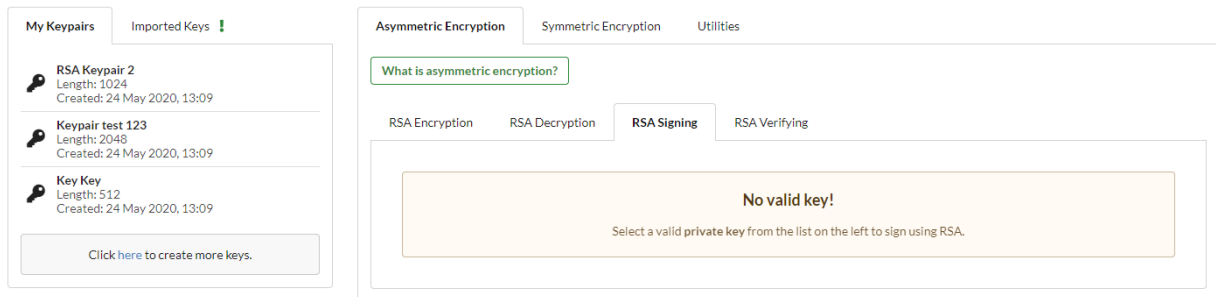


Figure 4-43 Asymmetric encryption is blocked until the user selects valid key

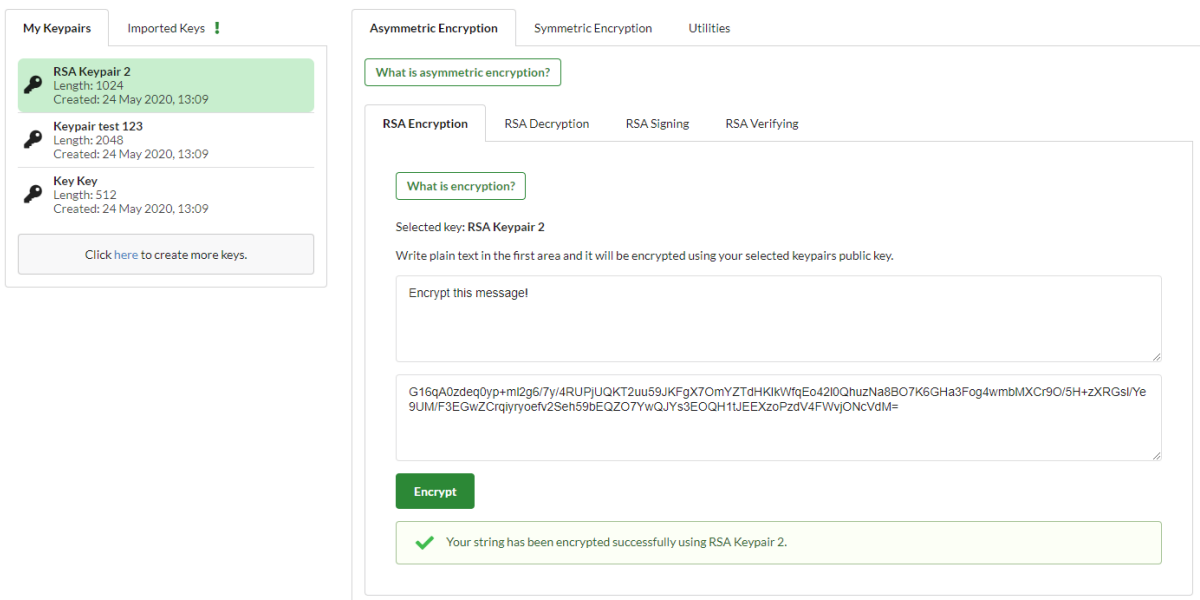


Figure 4-44 Encrypting a message using the selected keypairs public key

The screenshot shows the 'RSA Encryption' tab selected. At the top, there are four tabs: 'RSA Encryption', 'RSA Decryption', 'RSA Signing', and 'RSA Verifying'. Below the tabs, there is a green button labeled 'What is encryption?'. The text 'Selected key: RSA Keypair 2' is displayed. Below this, a message states: 'Write plain text in the first area and it will be encrypted using your selected keypairs public key.' There are two text input fields. The first field contains the placeholder text 'Write or paste your text here...'. The second field contains the placeholder text 'Encrypted text appears here...'. Below the first field is a green button labeled 'Encrypt'. At the bottom, a red error message box with a red 'X' icon contains the text: 'Your field cannot be empty!'.

Figure 4-45 Error when trying to encrypt an empty string with RSA

The screenshot shows the 'RSA Decryption' tab selected. At the top, there are four tabs: 'RSA Encryption', 'RSA Decryption', 'RSA Signing', and 'RSA Verifying'. Below the tabs, there is a green button labeled 'What is decryption?'. The text 'Selected key: RSA Keypair 2' is displayed. Below this, a message states: 'Paste in your encrypted text in the first area and it will be decrypted using your selected key.' There are two text input fields. The first field contains the text 'Some random data that are not encrypted'. The second field contains the placeholder text 'Plain text appears here...'. Below the first field is a green button labeled 'Decrypt'. At the bottom, a red error message box with a red 'X' icon contains the text: 'Could not decrypt your data using RSA Keypair 2.'.

Figure 4-46 Error when trying to decrypt an empty string with RSA

#### 4.4.6 Symmetric Encryption

In the symmetric section we have two different algorithms. Let us look at how can a user encrypt and decrypt using these algorithms.

To encrypt data, the user needs to provide a password. The application will then create a symmetric key derived from this password. The application encrypts and displays the encrypted data almost simultaneously as the user enters the data.

AES Encryption   AES Decryption   ROT13

**What is encryption?**

Write plain text in the first area and it will be encrypted using the AES algorithm and your chosen password.

Enter your password here:

MyPassword

This message will be encrypted.

4EWhPXNneAur6w+If6AI3rCX6T7N7tBm9C6Pc2mz/Tw=

Figure 4-47 AES encryption component in Chestnut

AES Encryption   **AES Decryption**   ROT13

**What is decryption?**

Paste in your decrypted data, write in the correct password and it will be decrypted using the AES algorithm.

Enter your password here:

MyPassword

4EWhPXNneAur6w+If6AI3IdAM2Y1fi/kpDodNn4ALJU=

This message will be encrypted

Figure 4-48 AES decryption component in Chestnut

Enter your password here:

MyWrongPassword

4EWHPXNneAur6w+lf6Al3idAM2Y1fj/kpDodNn4ALJU=

ø]0[-²0É080iWÔ0}}@0pVø00j\*\_\_80[

Figure 4-49 Providing an invalid password to AES decryption will not decrypt the message

To decrypt the data, the user needs to provide the same password. Providing the wrong password will not decrypt the data.

#### 4.4.6.1 Rot13

Rot13 (rotated by 13 places) is another symmetric algorithm. The application lets users encrypt and decrypt rot13 data.

What is ROT13?

Simple conversion of plain text to ROT13 text.

Encrypt My Data

Rapelcg Zl Qngn

Figure 4-50 ROT-13 text conversion in Chestnut

#### 4.4.7 Utilities

When a user navigates to this section, they can use a few utility tools such as UUID generator, checksum generator, or base64 encoder.

##### 4.4.7.1 UUID generator

A user can type in how many UUIDs they want to generate and click on the “Generate” button to see a list of generated UUIDs. Have a set a limit of maximum 20 UUIDs that a user can generate because we do not want to have a long list of scrolling data.

How many UUID do you want to generate?

20

Generate

901b1417-ae90-413e-8e55-e354733aa567  
d5551e33-064f-4e79-96da-3c1a063f9f26  
9e065a44-6653-44d0-a4c9-95b33e038e72  
1499a361-053d-4878-80d9-a265676b0070  
7edd84f6-1285-4004-9832-d936eda46f27  
8bb9603f-98f1-4a80-8945-ea41e77c1262  
4f84da45-fef6-411e-84da-b43f689e8185  
0377b070-debe-4d5d-88e9-83f36aae7998  
fe241f66-6d3c-43b6-a29b-17ce1ea080da  
223d988f-6b03-4d6c-8b9f-856fcb7a4b2c  
77cf05fb-4630-4a99-8b44-841977c73998  
9a14ffc2-c3e0-471b-ab39-82423890766f  
105a77b1-f78a-4c2a-a124-3b31a8c8687a  
32c904e4-6117-4b91-90b7-e184926911ca  
b34b0e1a-4a64-418c-afa8-1d1d156291f0  
4de89383-d3a2-4041-bde0-2f88f478aa40  
69b5c21c-c7bf-4ab1-a335-9abbcbf6311b  
50b29f21-66ff-45a4-b93f-ae2b8d578a3  
0c2d3cfc-f91e-40fc-a769-ce8aa2f27cd6  
5d72ce1e-a7b1-48b7-8212-ef844ea75cc1

Figure 4-51 UUID generation in Chestnut

Not providing a number or typing a greater number than 20 will give an error.

What is a UUID?

How many UUID do you want to generate?

30

Generate

✖ Choose an amount between 1 and 20.

Figure 4-52 Error for providing invalid amount of UUIDs

#### 4.4.7.2 Checksum generator

This utility tool creates checksums for user data or an uploaded file using four different hash functions. The MD5, SHA1, SHA256, and SHA512. It is important to note that we do not upload the file to our server or our database. The checksums are calculated inside the browser.

[What is Checksum?](#)

Write or paste in plain text in the first area or choose a .txt file to generate hash values based on its content.

Text to checksum:

Kanban is a work management system

[Choose File](#) [English] What is Kanban\_ - Agile Coach (2019) [DownSub.com].txt

MD5

76366096826c18d9067437b77e2bd63d

SHA-1

75672646503ab008263291dce3bee7c3b022d58e

SHA-256

ab081359b41b97194a4db90242155afac3f23191b2ba2492e4f9bd3970aa114b

SHA-512

41d7806b48fdbac3dd7a26b8cbbd3156e99730185a16c9dfc98594746970dde8ec158bce8c873df041e18c0ff5a12892ab8ec45894d083

Figure 4-53 Checksum generator from file or string text in Chestnut

#### 4.4.7.3 Base64 encoder/decoder

A user can simply encode or decode data from the base64 format. The application performs encoding/decoding at the same time as the user inputting the data.

Base64 Encoding

Base64 Decoding

UUID Generator

Checksum

What is Base64?

Write plain text in the first area and it will automatically be encoded into Base64.

Encode this to base64 format

RW5jb2RlIHRobXMgdG8gYmFzZTY0IGZvcmlhdA==

Figure 4-54 Base64 encoding in Chestnut

Base64 Encoding

Base64 Decoding

UUID Generator

Checksum

What is Base64?

Paste in your Base64 encoded text in the first field and it will be automatically decoded.

RW5jb2RlIHRobXMgdG8gYmFzZTY0IGZvcmlhdA==

Encode this to base64 format

Figure 4-55 Base64 decoding in Chestnut

## 4.5 Testing

Testing is a central part of development. The goal of testing is to get feedback on whether or not the application and its function do what they are supposed to do. Performing tests highlight the mistakes and make it possible to fix them.

### 4.5.1 Unit testing



Unit testing refers to performing tests for each method and function independently to see if they are fit to be used. It is common practice to try and make the functions fail and see if it throws error correctly. Due to the project's workload, it was difficult for us to allocate time for unit testing. Instead, we tried to do as many integration tests as possible during the development phase.

## 4.5.2 Integration testing

First, we performed integration tests on our backend and API using Swagger and Postman. Later we performed integration tests for the whole application.

### 4.5.2.1 Swagger

We used Swagger to perform simple API tests. We could execute several endpoints from swagger to check if they work or not.

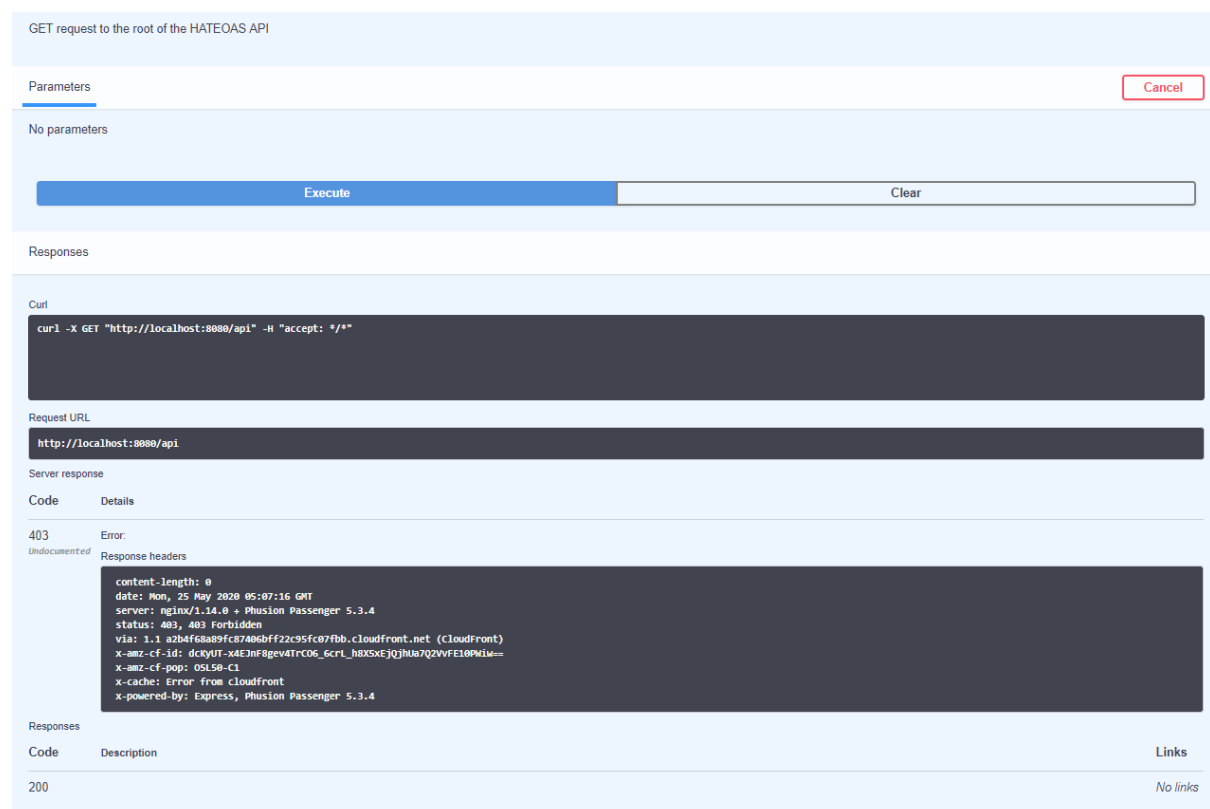


Figure 4-56 Performing API call tests on Swagger

### 4.5.2.2 Postman

In Postman, we can create collections of API requests. Each collection can contain multiple folders. This allows us to create a similar structure to Swagger.

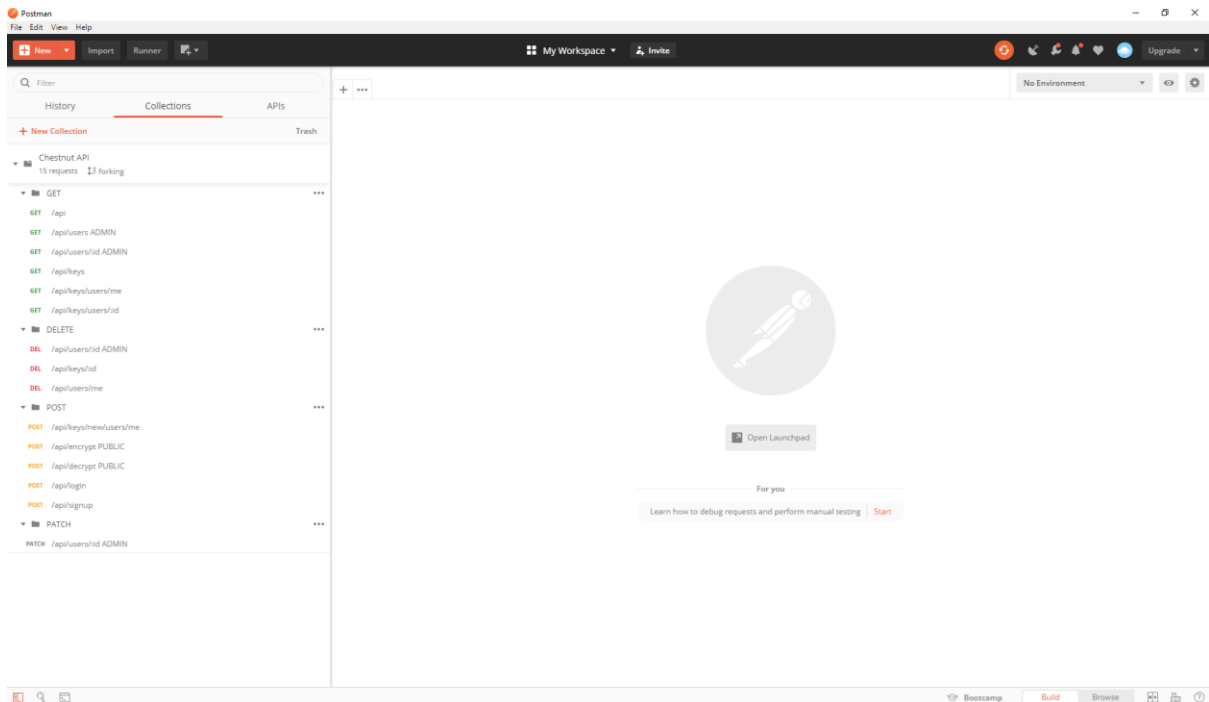


Figure 4-57 Main view of Postman with our Chestnut API endpoints collection

After creating the collection, we can select any of the API requests by clicking on it. To send a login request, we need to select it from the POST folder. A new section will appear on the right part of the Postman application.

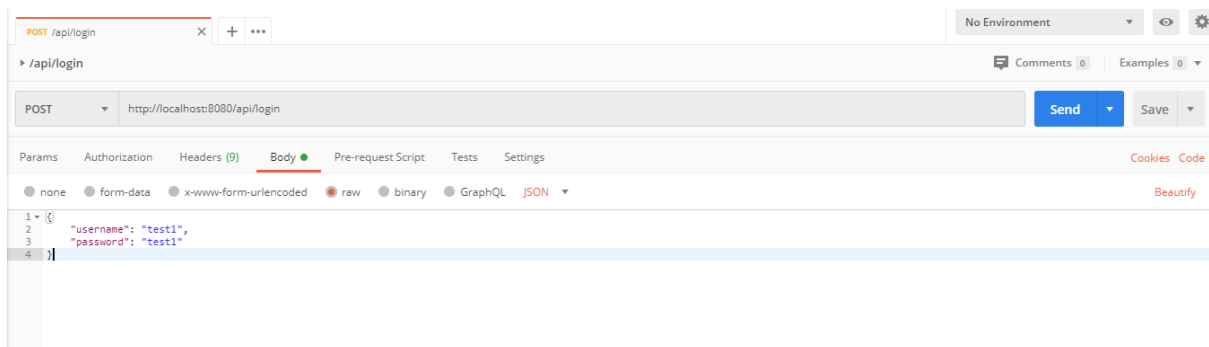


Figure 4-58 Sending a POST request in Postman

To be able to send our login credentials, we need to first click on the “Body” and then choose “raw” from the radio selector and “JSON” from the dropdown selector. Now we can write our login info as JSON and send the request by clicking on the blue “send” button.

Postman returns a response depending on if the request was successfully handled or not.



### 5.3 Cipher methods for AES encryption

The AES encryption/decryption uses CBC (cipher block chaining). The next step will be adding support for different ciphers such as cipher feedback (CFB) or output feedback (OFB).

### 5.4 Conclusion

There are quite a few features that can be ideal to add to Chestnut. However, it is important to reflect on how any change can impact this application and its perceived value as an educational tool.

## 6 Bibliography

- Adobe. (2020). *UI/UX design and collaboration tool*. Retrieved from Adobe: <https://www.adobe.com/products/xd.html>
- Avraham, S. B. (2017, 9 5). *What is REST — A Simple Explanation for Beginners, Part 2: REST Constraints*. Retrieved from Medium: <https://medium.com/extend/what-is-rest-a-simple-explanation-for-beginners-part-2-rest-constraints-129a4b69a582>
- Babich, N. (2016, 3 15). *Button UX Design*. Retrieved from uxplanet: <https://uxplanet.org/button-ux-design-best-practices-types-and-states-647cf4ae0fc6>
- Backend Definition*. (2020, 4 11). Retrieved from TechTerms: <https://techterms.com/definition/backend>
- bcrypt*. (2020). Retrieved from npmjs: <https://www.npmjs.com/package/bcrypt>
- body-parser*. (2020). Retrieved from npmjs: <https://www.npmjs.com/package/body-parser>
- Bogdan, G. (2016, 4 10). <https://www.lynda.com/Flask-tutorials/CRUD-REST-basics/521200/533063-4.html>. Retrieved from Lynda: <https://www.lynda.com/Flask-tutorials/CRUD-REST-basics/521200/533063-4.html>

Cockburn, A., & Highsmith, J. (2001, 11). Agile software development: The people factor. *Computer*, 34(11), 131-133. doi:10.1109/2.963450

CommonLounge.com. (2018). *What is Crypttography? Why Cryptography?* Retrieved from CommonLounge: <https://www.commonlounge.com/discussion/921db548a81f4d5d91cd03fc22f4b0a1>

*Crypto Programs*. (2019, 11 23). Retrieved from Crypto Programs: <http://www.cryptoprograms.com/>

*Cryptool Online*. (2019). Retrieved from Cryptool Online: <https://www.cryptool.org/en/cto-documentation/wiki>

*CSS Tutorial*. (n.d.). Retrieved from w3schools: <https://www.w3schools.com/css/default.asp>

Drummond, R. (2013, 5 8). *How to build a REST Web API on a Raspberry PI in JavaScript*. Retrieved from Robert Drummond: [http://www.robert-drummond.com/2013/05/08/how-to-build-a-restful-web-api-on-a-raspberry-pi-in-javascript-2/?ak\\_action=reject\\_mobile](http://www.robert-drummond.com/2013/05/08/how-to-build-a-restful-web-api-on-a-raspberry-pi-in-javascript-2/?ak_action=reject_mobile)

*Express*. (2017). Retrieved from Expressjs: <https://expressjs.com/>

*express-hateoas-links*. (2019). Retrieved from npmjs: <https://www.npmjs.com/package/express-hateoas-links>

*facebook/react*. (2020). Retrieved from GitHub: <https://github.com/facebook/react>

Fielding, T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine.

Fowler, M. (2010, 3 18). *Richardson Maturity Model*. Retrieved from MartinFowler: <https://martinfowler.com/articles/richardsonMaturityModel.html>

*Front-end web development*. (2020, 5 18). Retrieved from Wikipedia: [https://en.wikipedia.org/wiki/Front-end\\_web\\_development](https://en.wikipedia.org/wiki/Front-end_web_development)

Fruhlinger, J. (2020, 2 10). *The CIA triad: Definition, components and examples*. Retrieved from csoonline: <https://www.csoonline.com/article/3519908/the-cia-triad-definition-components-and-examples.html>

GitHub. (2019). *About GitHub*. Retrieved from GitHub: <https://github.com/about/>

Gruber, J. (2020). *Markdown*. Retrieved from Daring Fireball: <https://daringfireball.net/projects/markdown/>

*HATEOAS and REST: The hypermedia principle*. (2018, 2 14). Retrieved from ionos: <https://www.ionos.com/digitalguide/websites/web-development/hateoas-information-on-the-rest-constraint/>

*HATEOES*. (n.d.). Retrieved from Wikipedia: <https://en.wikipedia.org/wiki/HATEOAS>

Henry, K. (2020, 3 4). *Why do We Use the CIA-Information Security Triad?* Retrieved from readynez: <https://www.readynez.com/en/blog/kevin-henry-why-do-we-use-the-cia-information-security-triad/>

*HTTP response status codes*. (2020, 5 21). Retrieved from Mozilla developer: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

*Intro to React*. (2020). Retrieved from ReactJS: <https://reactjs.org/tutorial/tutorial.html>

*Introducing JSX*. (2020). Retrieved from ReactJS: <https://reactjs.org/docs/introducing-jsx.html>

*Introduction to JSON Web Tokens*. (2020). Retrieved from JWT: <https://jwt.io/introduction/>

ISO. (2010). *Ergonomics of human-system interaction*. Retrieved from Iso.org: <https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-1:v1:en>

*Jitsi Meet - Instant free videoconferencing*. (2020). Retrieved from Jitsi Meet: <https://jitsi.org/jitsi-meet/>

*JSX - a faster, safer, easier JavaScript*. (2013). Retrieved from <https://jsx.github.io/>

MERTZ, N. (2012). *How and Why Icons Improve Your Web Design*. Retrieved from usabilla: <https://usabilla.com/blog/how-and-why-icons-improve-you-web-design/>

*mysql2*. (2020). Retrieved from npmjs: <https://www.npmjs.com/package/mysql2>

*Node.js*. (2020). Retrieved from Node.js: <https://nodejs.org/en/about/>

*Pair Programming - Does it really work?* (n.d.). Retrieved from Agile Alliance: <https://www.agilealliance.org/glossary/pairing/>

Patel, P. (2018, 4 18). *What exactly is Node.js?* Retrieved from freecodecamp: <https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/>

PCmag. (2020). *Software portability*. Retrieved from PCmag: <https://www.pcmag.com/encyclopedia/term/software-portability>

*Principles behind the Agile Manifesto*. (2001). Retrieved from AgileManifesto.org: <https://agilemanifesto.org/principles.html>

*React JSX*. (n.d.). Retrieved from w3schools: [https://www.w3schools.com/react/react\\_jsx.asp](https://www.w3schools.com/react/react_jsx.asp)

*REST APIs must be hypertext-driven*. (2008, 10 20). Retrieved from Roy T. Fielding: <https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>

*REST Architectural Constraints*. (2018). Retrieved from restfulapi: <https://restfulapi.net/rest-architectural-constraints/>

*Roy T. Fielding*. (2012). Retrieved from roy.gbiv: <https://roy.gbiv.com/>

*SageMath*. (2020). Retrieved from SageMath: <https://www.sagemath.org/>

*Sequelize ORM*. (2020). Retrieved from Sequelize: <https://sequelize.org/>

*Swagger documentation*. (2020). Retrieved from Swagger: <https://swagger.io/docs/specification/about/>

tkroff. (2019, 12 12). *FrontendFrameworksPopularity*. Retrieved from <https://gist.github.com/tkroff/b1caa4c3a185629299ec234d2314e190>

Trello. (2020). *What is Trello?* Retrieved from Trello: <https://trello.com/about>

Visual Studio. (2020). *Visual Studio Code*. Retrieved from Visual Studio: <https://code.visualstudio.com/>

*What is a Functional Requirement?* (2020). Retrieved from guru99:  
<https://www.guru99.com/functional-requirement-specification-example.html>

*What is Non-Functional Requirement?* . (2020). Retrieved from guru99:  
<https://www.guru99.com/non-functional-requirement-type-example.html>

*What is the Richardson Maturity Model?* (2016). Retrieved from restcookbook:  
<http://restcookbook.com/Miscellaneous/richardsonmaturitymodel/>

*What is User Centered Design?* (n.d.). Retrieved from Interaction design: <https://www.interaction-design.org/literature/topics/user-centered-design>

## 7 Table of figures

Figure 1-1 Three principles known as the CIA triad.....	8
Figure 1-2 Figure of ssh-keygen commands.....	9
Figure 1-3 Figure of ssh-keygen command with additional options.....	9
Figure 1-4 Result output from ssh-keygen command .....	10
Figure 1-5 Chestnut as a JavaFX application .....	11
Figure 1-6 Chestnut as a JavaFX application (Symmetric Enc).....	11
Figure 1-7 Chestnut as a JavaFX application (UUID).....	12
Figure 1-8 Chestnut as a JavaFX application (Generate keypair) .....	12
Figure 1-9 Cryptoprograms' web implementation of the Beaufort cipher.....	15
Figure 1-10 Cryptoprograms' Beaufort substitution matrix .....	16
Figure 1-11 CryptTool website AES implementation.....	18
Figure 1-12 CryptTool website RSA implementation .....	18
Figure 1-13 SageMath mathematical software interface.....	19
Figure 1-14 Downloads per year comparison for React, Angular, Vue.....	20
Figure 2-1 Trello being used as our digital Kanban board (Early tasks) .....	27
Figure 2-2 Trello being used as our digital Kanban board (Backend tasks) .....	28
Figure 2-3 Trello being used as our digital Kanban board (Frontend tasks) .....	28
Figure 2-4 User centered design diagram .....	31
Figure 2-5 Chestnut as a JavaFX application .....	32
Figure 2-6 Project phases shown on the work timeline .....	35
Figure 2-7 Early design sketch for the landing page and demo component .....	36
Figure 2-8 Early design sketch for website after authentication.....	36
Figure 2-9 Early interactive prototype made in Adobe XD.....	37
Figure 2-10 Figure showing file structure of Chestnuts backend directory.....	38
Figure 2-11 Overview of all Chestnut API endpoints.....	39
Figure 2-12 Figure of the chestnut database with keypairs and users tables .....	40
Figure 2-13 Chestnut as a web application .....	41
Figure 3-1 Chestnut API response in JSON format .....	46
Figure 3-2 Communication between a client and an API .....	46
Figure 3-3 Overview of all the API endpoints, image taken from Postman.....	48
Figure 3-4 HTTP verbs compared to CRUD operations.....	49

Figure 3-5 Figure showing Richardson's maturity levels for REST APIs .....	50
Figure 3-6 Chestnut API response to GET /api/keys/users/me .....	52
Figure 3-7 Figure showcasing both the backend and the frontend running simultaneously on different servers .....	53
Figure 3-8 Authorization token within the header portion of a GET request to Chestnut .....	54
Figure 3-9 Authorization token payload contents.....	55
Figure 3-10 Controller and route folders in the backend directory .....	56
Figure 3-11 Controller that handles deletion of currently logged in user .....	57
Figure 3-12 Delete currently logged in user route .....	57
Figure 3-13 Figure showing a 422 status code response.....	58
Figure 3-14 Figure showing a 400 status code response.....	58
Figure 3-15 Middleware function checking for authorization token.....	60
Figure 3-16 Connecting to the database using Sequelize.....	61
Figure 3-17 Figure showing the Swagger Editor.....	63
Figure 3-18 Overview of all endpoints taken from the published page of Chestnut API on Swagger .	64
Figure 4-1 Chestnut manage keys components layout .....	66
Figure 4-2 Chestnut Keys List component layout.....	66
Figure 4-3 Creating a folder named chestnut2020spring and creating template of a react app.....	67
Figure 4-4 Terminal output after create-react-app has finished downloading all files .....	68
Figure 4-5 Overview of installed packages for the frontend part of Chestnut .....	68
Figure 4-6 Declaring a button in JSX.....	70
Figure 4-7 Corresponding button rendered in HTML from the previous JSX button.....	70
Figure 4-8 Making use of augmentation to declare a menu item as a link .....	71
Figure 4-9 Subcomponent showcase in JSX .....	71
Figure 4-10 Analogous color wheel.....	72
Figure 4-11 Compound color wheel.....	72
Figure 4-12 Triadic color wheel.....	72
Figure 4-13 Color picker in Adobe Color .....	73
Figure 4-14 The whole navbar of Chesnut web app.....	73
Figure 4-15 Zoomed in on the left side of the navbar .....	73
Figure 4-16 Zoomed in on the right side of the navbar being logged in as admin .....	73
Figure 4-17 Zoomed in on the right side of the navbar being logged in as normal user .....	74
Figure 4-18 Checking the contrast between white text on our shade of green .....	74
Figure 4-19 Figure of sub-navbars on the home page of the application.....	75
Figure 4-20 Keys List with small trash icon indicating the possibility to delete a key .....	76
Figure 4-21 Different states a button can be in .....	77
Figure 4-22 Different states of a button in the Chestnut solution .....	77
Figure 4-23 Modal warning the user before deleting a key .....	78
Figure 4-24 Educational tooltip with definition for symmetric encryption .....	79
Figure 4-25 Chestnut landing page .....	80
Figure 4-26 Chestnut landing page with demo .....	80
Figure 4-27 Log in box for Chestnut.....	81
Figure 4-28 Errors for invalid credentials entered by the user.....	81
Figure 4-29 Chestnut homepage for logged in users .....	82
Figure 4-30 Manage keys page in Chestnut .....	82
Figure 4-31 Admin settings page in Chestnut .....	83
Figure 4-32 Modal warning the admin before deleting an user.....	83



Figure 4-33 Account settings page in Chestnut with option to delete your account .....	84
Figure 4-34 Modal warning the user before permanently deleting their account .....	84
Figure 4-35 Generate keypair component with successfull message .....	85
Figure 4-36 Generate keypair component with error message .....	85
Figure 4-37 My keypairs list and imported keys list .....	86
Figure 4-38 Modal warning the user before permanently deleting a keypair .....	86
Figure 4-39 Selecting a keypair to show its contents: public and private key .....	87
Figure 4-40 Selecting imported key to view its public key .....	87
Figure 4-41 List of registered users filtered by a search bar .....	88
Figure 4-42 List of users showing no results for a search .....	88
Figure 4-43 Asymmetric encryption is blocked until the user selects valid key .....	89
Figure 4-44 Encrypting a message using the selected keypairs public key .....	89
Figure 4-45 Error when trying to encrypt an empty string with RSA .....	90
Figure 4-46 Error when trying to decrypt an empty string with RSA .....	90
Figure 4-47 AES encryption component in Chestnut .....	91
Figure 4-48 AES decryption component in Chestnut .....	91
Figure 4-49 Providing an invalid password to AES decryption will not decrypt the message .....	92
Figure 4-50 ROT-13 text conversion in Chestnut .....	92
Figure 4-51 UUID generation in Chestnut .....	93
Figure 4-52 Error for proving invalid amount of UUIDs .....	93
Figure 4-53 Checksum generator from file or string text in Chestnut .....	94
Figure 4-54 Base64 encoding in Chestnut .....	95
Figure 4-55 Base64 decoding in Chestnut .....	95
Figure 4-56 Performing API call tests on Swagger .....	96
Figure 4-57 Main view of Postman with our Chestnut API endpoints collection .....	97
Figure 4-58 Sending a POST request in Postman .....	97
Figure 4-59 Getting an HTTP response back in Postman .....	98