



DIGITAL MANAGEMENT OF THE PAYMENT RECEIPTS

MSc Dissertation

ANDRÉ RIBEIRO ARAGÃO

UNIVERSITY'S SUPERVISOR
Prof. Miguel Pimenta Monteiro

COMPANY'S SUPERVISOR
Eng. Gabriel Silva

University of Porto
Faculty of Engineering of the University of Porto
Department of Electrical Engineering

Resumo

Os colaboradores Bosch realizam com frequência viagens de negócios a outros países e regiões com o objetivo de reunir com clientes ou outras partes de interesse. Em cada viagem, os colaboradores devem garantir que guardam todos os recibos que corroborem as despesas decorridas para que seja possível a respetiva submissão e reembolso.

Digital Management of the Payment Receipts é uma dissertação de mestrado que teve como objetivo o desenvolvimento de um sistema escalável que simplifique o processo de gestão dos recibos. O objetivo foi modelar, implementar e testar uma solução capaz de extrair os dados dos recibos e apresentar os resultados do processamento, integrando tecnologias de ponta que respondam aos requisitos do sistema. Para isso, a pesquisa teve como foco todos os aspectos importantes do sistema, desde a extração e o processamento de dados até às arquiteturas de *back-end* e sua escalabilidade. A solução consistiu no desenvolvimento de uma aplicação móvel suportada por um sistema *back-end* que segue uma arquitetura de micro-serviços.

A aplicação foi desenvolvida com a *framework* React Native e as suas principais características são permitir ao utilizador fotografar os recibos e exibir o seu estado atual, possibilitando assim a realização de ações específicas inerentes a esse mesmo estado. Nesse sentido, a aplicação móvel atua como uma ponte entre o utilizador e o sistema *back-end*, resultando em trocas regulares de dados entre os dois.

Os pedidos do aplicativo móvel são enviados a um *API Gateway* que os encaminha de acordo com serviço requisitado. Os serviços disponíveis são o envio de *e-mails*, através de Twilio SendGrid, a manipulação da base de dados, através de MongoDB Atlas, a extração e processamento de dados, através de Tesseract OCR, implementado sobre um servidor desenvolvido em Node.js, e o armazenamento de dados, através da solução AWS S3.

Tendo em conta a natureza dos dados transacionados entre serviços, a segurança foi um ponto particularmente relevante durante o desenvolvimento do projeto. Assim, de modo a garantir que toda a comunicação é segura, esta é feita através do protocolo HTTPS, que conta com uma camada de segurança que utiliza o protocolo SSL/TLS, com autenticação através de *tokens*. Relativamente ao armazenamento de dados, o *token* é guardado no dispositivo móvel com recurso a *keychain services*, no *iOS*, ou ao *Android keystore system*, no *Android*; as *passwords* são *hashed* e *salted* com o algoritmo Bcrypt antes de serem armazenadas na base de dados e o armazenamento dos dados no serviço AWS S3 é encriptado através do padrão criptográfico AES-256.

O sistema foi submetido a uma fase de testes dividida em dois domínios distintos: testes funcionais e testes de desempenho. Os testes funcionais foram executados na aplicação móvel e no *API Gateway* de maneira a garantir o comportamento correto de ambos os sistemas. O comportamento do sistema foi o esperado. Os testes de desempenho foram realizados para o serviço de extração e processamento de dados, no que diz respeito à percentagem de erro dos resultados e ao atraso de processamento. Os resultados foram insatisfatórios e mostram que a solução não está pronta para ser implementada. De maneira a solucionar este problema, é recomendado que os dados treinados usados pelo algoritmo OCR sejam baseados em recibos e que seja implementado pré-processamento das imagens.

Abstract

As a worldwide company, Bosch associates perform regular business trips to other countries and regions in order to meet customers or other stakeholders. During each business trip, all associates need to ensure they keep all expense receipts, as means to later submit them for reimbursement.

Digital Management of the Payment Receipts is a MSc dissertation that aimed to develop a scalable system that simplifies the receipts' management process. The objective was to model, implement and test a solution that is able to extract the receipts' data and present the processing results while integrating state of the art technologies that fulfill the system's requirements. In order to accomplish that, research was focused on every important aspect of the system, from data extraction and processing to back-end architectures and its scalability.

The solution consisted in the development of a mobile application supported by a back-end system that follows a micro-services architecture.

The application was developed using React Native and its main features are allowing the user to photograph receipts and displaying the current receipt stage, thus enabling the user to take stage specific actions. Therefore, the mobile application acts as a bridge between the user and the back-end system, ensuing regular data exchanges between the two.

The requests from the mobile application are sent to an API Gateway that forwards them accordingly to the required service. The available services are email sending, provided by Twilio SendGrid, database management, provided by MongoDB Atlas, data extraction and processing, provided by Tesseract OCR and implemented as a Node.js server, and data storage, provided by AWS S3.

As a result of the nature of the data being handled, security was taken seriously. In order to ensure that, all communication is done through HTTPS, which uses the security protocol SSL/TLS, with token authentication. With regard to the data storage, the token is saved in the mobile client using either *keychain services*, in *iOS*, or *Android keystore system*, in *Android*. The passwords are hashed and salted with the Bcrypt algorithm before being stored in the database, and the data storage, in the AWS S3 service, is encrypted with the cryptographic algorithm AES-256.

The system was subjected to a testing phase in two different domains: functional and performance testing. Functional testing was focused on the mobile application and the API Gateway to ensure the correct behavior of both systems. The system behaved as expected. Performance testing was carried out for the data extraction and processing service in regards to the error percentage of the results and the processing delay. The results were unsatisfactory and showed that the solution is not ready to be implemented. To further address this matter, it is recommended that the trained data used by the OCR algorithm is based on receipts and that the pictures are pre-processed.

Acknowledgments

Firstly, I would like to thank Bosch for the opportunity of working on such a challenging project. It is also worth mentioning that the way in which the company made me feel welcome, and made sure I was well integrated in the team and the company, was undoubtedly a door that I may never want to close. As the typical portuguese saying goes, *o bom filho a casa torna*.

To Eng. Gabriel Silva, my supervisor at Bosch, I thank for the endless support, and above all, for the companionship and compassion shown during the four months of continuous work.

To my supervisor at FEUP, Prof. Miguel Pimenta Monteiro, I would like to highlight the availability to always answer my questions and guide the development process.

Now, on a less professional tone, I thank my friends, colleagues and family.

To my greatest friend, Rui Saraiva, for helping me find myself, specially during difficult times as the ones experienced in 2020, and for being the one in whom I trust the most. Above all, thank you for always showing me the inconvenient side of things, and grounding my feet.

To João Madail, for always being there to support me, no matter which direction I take.

To André Matos and Margarida Marques, for the five years of University together. For all the good memories together, all the work, and most importantly, for showing me different perspectives.

To my hopefully to be engineers that have come a long way since our paths crossed, Gonçalo Santos and Nuno Barros. To Gonçalo, my oldest friend, for being by my side since the first grade of elementary school, and for experiencing with me almost every step in life, including the five years of University, with many projects completed together; and to Nuno Barros, for being there since day one and never looking back when it comes to support, advice or just a talk.

Lastly, I would like to thank my mother, Maria Gil, and my grandparents, Dialino Ribeiro and Maria José. Without them, this wouldn't be possible as they are the reason for me to be who am I today. For the countless years, and the many more to come, the last to be mentioned are always the first.

André Ribeiro Aragão

Contents

Resumo	1
Abstract	3
Acknowledgments	5
Contents	6
List of Figures	9
List of Tables	11
List of Code	13
Abbreviations and Symbols	15
1 Introduction	17
1.1 Overview of the intervention's domain	17
1.1.1 Bosch	17
1.2 Problem and characterization	18
1.3 Objectives	19
1.3.1 System requirements and functionalities	19
1.4 Expected benefits	21
2 OCR & hybrid frameworks for mobile development	23
2.1 Optical Character Recognition	23
2.1.1 Characterization	23
2.1.2 Existing solutions	25
2.2 Hybrid frameworks for mobile development	26
2.2.1 Characterization	26
2.2.2 Existing solutions	27
3 Proposed system solution	29
3.1 Mobile application	29
3.2 Back-end	30
3.3 Data storage	32
3.4 Communication	32
4 System implementation	33
4.1 Mobile application	33
4.1.1 UI Prototypes	33
4.1.2 UX Prototypes	40
4.1.3 Technologies	43
4.2 Back-end	44
4.2.1 Technologies	44
4.2.2 Cloud services	45
4.3 Problems & solutions	46
4.3.1 React Native & Node.js familiarization	46
4.3.2 Mobile application: UI & UX	46
4.3.3 Mobile application: Code bugs	46
4.3.4 Time management	46

5	Developed system	49
5.1	Mobile application	49
5.1.1	UI	49
5.1.2	UX	61
5.1.3	Security	62
5.2	Back-end	63
5.2.1	Architecture	63
5.2.2	Database service	64
5.2.3	Storage service	67
5.2.4	OCR Service	70
5.2.5	Emailing service	71
5.2.6	API Gateway's API	73
5.2.7	Security	75
5.3	System operation & execution	79
5.3.1	Receipt stages	79
5.4	Tests	85
5.4.1	Functional	85
5.4.2	Performance	88
6	Discussion and conclusion	91
6.1	Summary	91
6.2	Learning outcomes	91
6.3	Objective fulfillment	92
6.4	Further system improvements	92
6.4.1	Mobile application: Implementation of <i>pure</i> React Native	92
6.4.2	Mobile application & back-end: Administration page	92
6.4.3	Mobile application & back-end: Lessen the use of third party libraries	92
6.4.4	Back-end: BaaS implementation	92
6.4.5	Back-end: Receipt status service	93
6.4.6	OCR service: Trained data & pre-processing	93
6.4.7	OCR service: REGEX	93
6.4.8	OCR service: Translation feature	93
6.4.9	Further testing	93
	Appendices	95
.1	UI Prototypes: Version 1	95
.1.1	UI Prototype - V1: Camera screen	95
.1.2	UI Prototype - V1: Receipt submission screen	96
.1.3	UI Prototype - V1: Receipts screen	97
.1.4	UI Prototype - V1: Receipt options screen	98
.1.5	UI Prototype - V1: Trips screen	99
.2	UI Prototypes: Version 3	100
.2.1	UI Prototype - V3: Camera screen	100
.2.2	UI Prototype - V3: Receipt submission screen	101
.2.3	UI Prototype - V3: Receipts screen	102
.2.4	UI Prototype - V3: Receipt overview screen	103
.2.5	UI Prototype - V3: Trips screen	104
.2.6	UI Prototype - V3: Account screen	105
.3	Bosch section presentation	106
	Bibliography	115

List of Figures

1.1	Bosch's logo	17
1.2	Bosch's expenses reimbursement submission protocol	18
1.3	Proposed expenses reimbursement submission protocol	19
2.1	OCR steps [7]	23
2.2	Development cycle of native and hybrid applications [9]	26
2.3	Most used hybrid frameworks and native languages	27
2.4	Cross-platform mobile frameworks in use by market share	27
3.1	Proposed system overview	29
3.2	Proposed back-end architecture	30
4.1	UI Prototype: For You screen	33
4.2	UI Prototype: Camera screen	34
4.3	UI Prototype: Receipt submission screen	35
4.4	UI Prototype: Receipt overview screen	36
4.5	UI Prototype: Receipts screen	37
4.6	UI Prototype: Trips screen	38
4.7	UI Prototype: Side menu screen	39
4.8	UX Prototype: For You screen	40
4.9	UX Prototype: Camera screen	41
4.10	UX Prototype: Receipts screen	42
4.11	UX Prototype: Trips screen	42
4.12	Initial project planning	47
4.13	Project timeline	47
5.1	UI: Login screen	49
5.2	UI: Forgot credentials screen	50
5.3	UI: Register screen	51
5.4	UI: Register screen - Maximum length validation	52
5.5	UI: Register screen - Email validation	52
5.6	UI: Register screen - Password matching validation	53
5.7	UI: Register screen - Password matching validation (2)	53
5.8	UI: For You screen	54
5.9	UI: Camera screen	55
5.10	UI: Receipt submission screen	56
5.11	UI: Receipt overview screen	57
5.12	UI: Receipts screen	58
5.13	UI: Trips screen	59
5.14	UI: Side menu screen	60
5.15	UX: Login screen	61
5.16	Back-end system architecture	63
5.17	Data handling & linkage between micro-services	69
5.18	AWS: Policy summary	76

5.19	AWS: User summary	76
5.20	Receipt stages state-machine	79
5.21	Displayed status values for each receipt stage, featured on the screen "Receipt Overview"	80
5.22	API Gateway testing: Postman testing suite	85
5.23	API Gateway testing: Login error	86
5.24	API Gateway testing: Login success	86
5.25	API Gateway testing: fetchAllReceipts	87
5.26	OCR service testing: Easy	88
5.27	OCR service testing: Medium (1)	88
5.28	OCR service testing: Medium (2)	88
5.29	OCR service testing: Medium (3)	88
5.30	OCR service testing: Hard (1)	89
5.31	OCR service testing: Hard (2)	89

List of Tables

5.1	OCR service: Processing time of one and five workers	89
5.2	OCR service: Percentage of error	90

List of Code

5.1	Mobile application: Formik form validator	53
5.2	API Gateway: pictureKey generation	66
5.3	OCR service: ocrDataKey generation	66
5.4	API Gateway & OCR service's environmental variables: AWS S3 service	67
5.5	API Gateway & OCR service: AWS S3 operations implementation	67
5.6	OCR service: Worker and scheduler creation	70
5.7	OCR service: Worker and scheduler initialization	70
5.8	OCR service: Computing OCR algorithm	70
5.9	API Gateway's environmental variables: Twilio SendGrid service	71
5.10	API Gateway: Twilio SendGrid operation implementation	71
5.11	API Gateway: Twilio SendGrid service usage	72
5.12	API Gateway: Bcrypt's password hashing	75
5.13	API Gateway: Bcrypt's password comparing	75
5.14	API Gateway: JWT secret generation	77
5.15	API Gateway's environmental variables: JWT	77
5.16	API Gateway: JWT signing	77
5.17	API Gateway: JWT authentication's middleware	78
5.18	API Gateway: JWT authentication's middleware usage	78
5.19	OCR service testing: time measurement	89
5.20	OCR service testing: Medium (3) processing result	90

Abbreviations and Symbols

AES	Advanced Encryption Standard
AI	Artificial Intelligence
API	Application Programming Interface
ARS	Amazon Resource Name
AWS	Amazon Web Services
AWS S3	AWS Simple Storage Service
BaaS	Back-end-as-a-Service
BSc	Bachelor of Science
BSD	Berkeley Software Distribution
CORS	Cross Origin Resource Sharing
CSS	Cascading Style Sheets
ETAG	Entity Tag
GPL	GNU General Public License
HMAC	Hash-based MAC
HR	Human Resources
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IaaS	Infrastructure-as-a-Service
IAM	Identity and Access Management
IoT	Internet of Things
JS	JavaScript
JSON	JavaScript Object Notation
JWT	JavaScript Object Notation Web Token
MAC	Message Authentication Code
MERN	MongoDB, Express, React, Node
MSc	Master of Science
NPM	Node Package Manager
OCR	Optical Character Recognition
OWASP	Open Web Application Security Project
PDF	Portable Document Format
PaaS	Platform-as-a-Service
REGEX	Regular Expression
REST	Representational State Transfer
S.A.	Sociedade Anónima (Anonymous Society)
SaaS	Software-as-a-Service
SMTP	Simple Mail Transfer Protocol
SSE-S3	Server Side Encryption
SSL	Secure Sockets Layer
TLS	Transport Layer Security

UI	User Interface
URL	Uniform Resource Locator
UX	User Experience
VPN	Virtual Private Network
VSCo	Visual Studio Code
W3C	World Wide Web Consortium
WAI	Web Accessibility Initiative
WCAG	Web Content Accessibility Guidelines
WSTG	Web Security Testing Guide
WWW	<i>World Wide Web</i>

1. Introduction

1.1 Overview of the intervention's domain

Digital Management of the Payment Receipts is a MSc dissertation co-oriented between the Faculty of Engineering of the University of Porto and Bosch Car Multimedia, Braga. This dissertation aims to apply both the problem solving skills and the technical skills acquired over the five years of the Integrated Master in Electrical and Computers Engineering to a practical and specific industry problem proposed by Bosch Car Multimedia, Braga. Therefore, the proposed problem falls under the enterprise domain.

Furthermore, the solution of the problem comprehends the management of the payment receipts, which necessarily involves the management of people and resources thus expanding the scope of the intervention's domain.

1.1.1 Bosch



Figure 1.1: Bosch's logo

Robert Bosch started the company in Stuttgart, in 1866, as a *Workshop for Precision Mechanics and Electrical Engineering*.

Today, the Bosch group is a leading global supplier of technology and services. It employs roughly 400,000 associates worldwide and has either subsidiaries or regional companies in more than 60 countries. If the sales network is taken into account, it is safe to say that Bosch is present in nearly every country in the world. Its operations are divided into four main business sectors:

- **Mobility Solutions;**
- **Industrial Technology;**
- **Consumer Goods;**
- **Energy and Building Technology.**

Bosch is renowned for its expertise in sensor technology, software, and services. Its innovative strength, with more than 70,000 associates in research and development domains, led to the establishment of Bosch as a reference in cutting edge technology such as smart homes, IoT, industry 4.0, connected mobility and AI.

From a financial perspective, in 2020, Bosch group generated sales of 71.5 billion euros.

1.1.1.1 Bosch Portugal

Bosch has 4 different facilities in Portugal, each one with its own distinct forte:

- **Bosch Car Multimedia**, in Braga, which is focused in the development of multimedia solutions and sensors for the automotive industry;
- **Bosch Security Systems**, in Ovar, that specializes in the development of communication and security systems, fire alarms and electronic displays;
- **Bosch Termotecnologia**, in Aveiro, that develops solutions in the field of thermotechnology, such as water heating and heat pumps;
- **Robert Bosch S.A.**, in Lisboa, that is the company's headquarters in Portugal and is mainly focused on sales, marketing and communication.

Bosch Portugal has more than 6,000 associated and generated sales of 1.8 billion euros in 2019, mainly from exports.

1.1.1.1.1 Bosch Car Multimedia (Braga)

The company's facilities were not always part of the Bosch group: history goes back to the early 90s when Blaupunkt started manufacturing car radios in the same exact place.

As a result of the market demands, in 2009, Blaupunkt had to let go of the manufacturing site, and Bosch Car Multimedia (Braga) began its history, initially focused in the development and manufacturing of infotainment systems, clusters and security sensors for the automotive industry. Currently, Bosch Car Multimedia has over 300 engineers and 3000 other associates, from manufacturing to HR, and is the 4th biggest exporter in Portugal.

1.2 Problem and characterization

A company of this size needs to be able to guarantee that associates are able to travel around the world for a multitude of reasons, such as establishing contact with clients, participating in international projects, developing additional technical skills, and others. This exhibits a challenge regarding the management and processing of trip costs.

For each trip, a standard form needs to be filled by the associate with information concerning the total costs for each category, such as transportation and food, and then sent to the HR department along with every receipt, so that it can be verified, approved and the associated refunded.

The verification process is of particular importance, given that the trip costs are only refunded accordingly to the company's reimbursement policy. For instance, alcohol is not eligible for reimbursement.

This process presents an overhead of work for both the associate and the HR department. Furthermore, the associate has to safely store each receipt during the trip which can contribute to the overall stress, resulting in less productive work.

Figure 1.2 highlights the sequence of different steps each associate goes through in order to be reimbursed for the trip expenses.



Figure 1.2: Bosch's expenses reimbursement submission protocol

1.3 Objectives

The goal of this project is to simplify the whole process by **modelling, implementing** and **testing** a scalable system with two distinct components:

- A **mobile application**, developed for the two most popular mobile operating systems, *iOS* and *Android*, that is able to photograph the receipts and send them to a **back-end server**;
- A **back-end server** that **autonomously extracts the data from the receipt** and identifies useful values, such as the total amount spent, and then export the results in a universal format, such as .csv or .pdf.

Figure 1.3 shows the proposed reimbursement protocol.



Figure 1.3: Proposed expenses reimbursement submission protocol

1.3.1 System requirements and functionalities

The system requirements and expected functionalities are presented in the following sections.

1.3.1.1 Mobile application

Requirement 1 Should be developed in a way that it runs on both major mobile operating systems, *Android* and *iOS*;

Requirement 2 Should follow the design rules of the respective mobile operating system;

Requirement 3 Should follow the design rules outlined in Bosch Brand Guide;

Requirement 4 Should work offline by storing a local copy of the photo and uploading it as soon as possible;

Requirement 5 Should be scalable;

1.3.1.2 Back-end server

1.3.1.2.1 Architecture

Requirement 6 Should be scalable;

1.3.1.2.2 OCR

Requirement 7 Should not be proprietary;

Requirement 8 Should be computed in an external server for performance reasons;

Requirement 9 Should present the result in a reasonable time;

Requirement 10 Should send a message to the client in case of impossibility to compute the OCR algorithm;

Requirement 11 Should be able to work with different alphabets;

Requirement 12 Should have sufficient accuracy to justify the change of Bosch's reimbursement protocol;

Requirement 13 Should be able to identify regular expressions, *REGEX*;

1.3.1.2.3 Data storage

Requirement 14 Should be securely stored;

1.3.1.3 Communication requirements

Requirement 15 Should be secure;

1.3.1.4 Optional features

1.3.1.4.1 Mobile application's accessibility

WCAG 2.1 is the latest revision of the web accessibility guidelines published by the WAI and the W3C [1]. The primary objective is to define strict rules to ensure that the web content is presented in a suitable way for all users, including those with disabilities.

This rules are divided into four different categories:

- Perceivable
 - Provide text alternatives for non-text content;
 - Provide captions and other alternatives for multimedia;
 - Create content that can be presented in different ways, including by assistive technologies, without losing meaning;
 - Make it easier for users to see and hear content;
- Operable
 - Make all functionality available from a keyboard;
 - Give users enough time to read and use content;
 - Do not use content that causes seizures or physical reactions;
 - Help users navigate and find content;
 - Make it easier to use inputs other than keyboard;
- Understandable
 - Make text readable and understandable;
 - Make content appear and operate in predictable ways;
 - Help users avoid and correct mistakes;
- Robust
 - Maximize compatibility with current and future user tools.

WAI and W3C provide a WCAG 2.1 reference guide to help developers meet the requirements [2]. In order to make the mobile application usable for the highest number of Bosch associates, accessibility options should be taken into account. Therefore, an optional system feature should be defined:

Requirement 16 The mobile app should follow accessibility guidelines according to WCAG 2.1;

1.3.1.4.2 System's security strengthening

OWASP Foundation is a community driven association that provides developers and technologists with articles, methodologies, documentation, tools and technologies regarding web application security.

WSTG is one of the tools offered by the OWASP Foundation that stipulates methodologies to test the security of web applications and web services [3].

By following OWASP cheat sheet, most of the rules defined in the WSTG can be implemented in the project, thus defining another system optional feature [4]:

Requirement 17 The system should follow the OWASP cheat sheet recommendations;

1.3.1.4.3 Connection to Bosch servers

The connection between the system and Bosch's internal servers would improve the application's scope since more features can be implemented, adding more value to the mobile application.

Requirement 18 The system should be connected to Bosch servers.

1.4 Expected benefits

The expected outcomes of the successful development of a system that behaves accordingly to the previous section 1.3 encompass benefits in multiple spheres.

Firstly, such system would boost the company's efficiency by relieving HR of the tedious task of carefully verifying each form, which in turn would result in that time being used for more meaningful work.

Secondly, it would relieve the associate from the stress related with the responsibility of requesting, keeping and processing the receipts. Thereby, trips would be more comfortable, practical and worry-free, resulting in the associate focusing his efforts on the task at hand. For the company, this can be seen as a productivity increase.

Last but not least, the proposed solution would undoubtedly add value to the company by reducing the variable production costs, as the time spent by the associate and the HR takes away company profits. This is particularly relevant since Bosch operates under principles of economies of scale.

2. OCR & hybrid frameworks for mobile development

2.1 Optical Character Recognition

2.1.1 Characterization

OCR is the process of recognizing optical patterns, specifically characters, from an image [5]. It belongs to the family of machine recognition techniques that perform automatic identification [5]. OCR algorithms are particularly useful for text digitization, hence the importance of further understanding of the domain.

Traditional pattern recognition was the industry standard decades ago, whereas soft computing techniques are now the typical choice for the development of cost effective OCR systems [5]. Soft computing techniques are a collection of computational techniques that attempt to study, model and analyze problems for which conventional methods have not yielded low cost, analytic and complete solutions [5]. The principal domains of soft computing are machine learning, fuzzy logic, probabilistic reasoning, neural computing or artificial neural networks and evolutionary computing or evolutionary algorithms [5]. The computational process in a soft computing environment can be described as a transformation from a modeling space M to a solution space S to a decision space D , as shown in equation 2.1 [6].

$$M \rightarrow S \rightarrow D \quad (2.1)$$

This process can be seen as the iteration over the following steps: segmentation or pre-processing, feature extraction and classification [7].



Figure 2.1: OCR steps [7]

Figure 2.1 describes the generic OCR steps. Each step is explained in its corresponding section.

2.1.1.1 Optical scanning

Before anything else, it's important to mention that the OCR output quality is crucially impacted by the quality of the source [8]. In this step, the printed document is converted into a bi-level black and white image, given a certain threshold that directly influences the output quality of this stage [5]. The threshold value is of most importance, as it allows saving memory space and reducing the computational effort [5]. Most algorithms used either a fixed threshold for the whole document, which is enough for documents with high contrast and uniform background, or an adaptive threshold that varies based on properties such as brightness or contrast, therefore being of special interest for photographed documents in odd light conditions [5].

2.1.1.2 Location segmentation

When applied to text, location segmentation is the step where the detection of the position of potentially interesting information in the document occurs, namely characters or words [5]. Problems may arise when trying to isolate connected characters. The most common problems occur when [5]:

- Characters are fragmented;
- Characters overlap each other;
- Noise and text cannot be distinguished clearly;
- Graphics and geometry are misinterpreted as text, or vice versa.

2.1.1.3 Pre-processing

Pre-processing takes care of possible defects or anomalies in characters, by making them as viable as possible for correct recognition [5]. For this reason, it is a fundamental step, since it improves the overall system accuracy [5]. The main objectives are **noise reduction**, **data normalization** and **compression** [5]. Noise reduction is done by filtering, various morphological operations and noise modeling [5]. The most important methods for normalization are skew and slant normalization, size normalization and contour smoothing [5]. Popular compression techniques are thresholding and thinning [5].

2.1.1.4 Segmentation

Segmentation is responsible for separating the character, previously processed, into its sub-components for better detection [5]. Segmentation strategies are subdivided into three categories: **explicit segmentation**, **implicit segmentation** and **mixed strategies** [5].

2.1.1.5 Representation

Although it is possible to feed grey-level or binary images to a recognizer, it is not the best way to do it, accuracy wise [5]. This step is responsible for extracting a set of features unique to each character. Representation methods can be categorized as **global transformation and series expansion**, by using the Fourier transform, Gabor transform, wavelets, moments or Karhunen Loève expansion, **statistical representation**, being the major statistical features used for character representation zoning, crossing and distances and projections, or **geometrical and topological representation**, which can be grouped into various sub-types: extracting and counting topological structures, measuring and approximating geometrical properties, coding and graphs and trees [5].

2.1.1.6 Feature extraction

The objective of feature extraction is to acquire essential characteristics of symbols [5]. The simplest way to do this is by rasterizing the image, but it is not the most used [5]. Feature extraction can be done by extracting specific features through various techniques and then evaluate them according to factors such as noise sensitivity, deformation, and others [5].

After this, **classification** takes place. Classification identifies each character and associates it with the correct character class, through either theoretical or structural methods [5]. The character class is obtained by **training and recognition** of the OCR system [5].

2.1.1.7 Post-processing

Commonly used post-processing activities are **grouping** and **error detection and correction** [5]. This step is specially important because it takes into account the surroundings of the characters by trying to recognize the word it is part of and if it makes sense [5]. This way, the algorithm is able to recognize wrongfully detected characters [5]. Another way of correcting a miss-detected character is by looking in a dictionary and if the word does not exist it is replaced by the most probable one [5]. This method introduces a big overhead of time processing [5].

2.1.2 Existing solutions

Regarding licensing, available solutions fall under two categories: **proprietary** and **free software**, which is licensed under Apache, BSD or GPL. In terms of platforms, they fall under three different categories: **desktop**, **mobile** or **cloud** based.

2.1.2.1 Proprietary

Most common examples of proprietary OCR software are ABBYY FineReader and Google Docs OCR, also known as Google Cloud Vision. ABBYY FineReader is available for all operating systems and also as an online service. On the other hand, Google Docs OCR is only available as an online service, namely by using a browser in different platforms.

ABBYY Mobile OCR Engine, a child of ABBYY FineReader, is used by DBS Bank, a Singapore-based banking and financial services corporation. This reinforces the idea that OCR software is already perceived by companies and end-users as a quality solution that effectively accomplishes its purpose.

OmniPage Ultimate and Rossum are other examples of trusted proprietary OCR software.

2.1.2.2 Free software

One of the most used free OCR software is Tesseract OCR. Firstly created by Hewlett-Packard, this software was then developed by Google and now has its own online repository, developed through community contributions.

Other examples are FreeOCR and SimpleOCR, although the first one is open-source and the second one is not. FreeOCR runs Tesseract OCR engine which strengthens Tesseract OCR's market position.

2.1.2.3 Performance comparison

Tafti et al. carried out a study that compared the performance of Google Docs OCR, Tesseract, ABBYY FineReader and Transym [8]. Google Docs OCR and ABBYY FineReader produced better results, presenting lower standard deviation ($\approx 18\%$ versus $\approx 25\%$ for Tesseract and Transym) and consistent accuracy across different types of images of the data set in analysis. However, as expected, all of the examined OCR systems faced a problem with skewed, blurred, and noisy images.

2.2 Hybrid frameworks for mobile development

2.2.1 Characterization

First and foremost, it's important to define the difference between native and hybrid applications development for better understating of the different mobile development methodologies.

Native apps are written using natively supported programming languages, thus being written, tested and built for a specific system [9]. On the contrary, hybrid apps are written using the framework's programming language, resulting in only needing to be tested once, and then built for all the supported platforms [9]. Consequently, hybrid apps are particularly interesting for cross-platform development as the development process is faster and cheaper [9]. Figure 2.2 illustrates the development cycle differences.

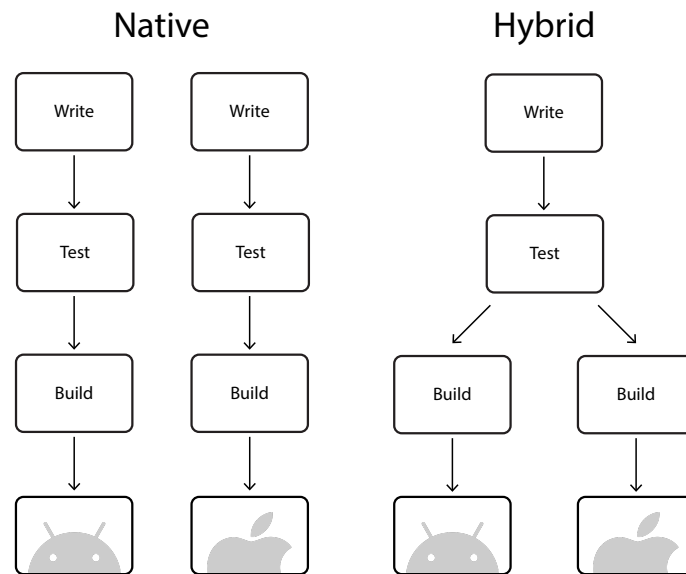


Figure 2.2: Development cycle of native and hybrid applications [9]

Hybrid applications can be based on **near-native stack** or **web stack**.

2.2.1.1 Near-native stack

Near-native stack means that the code follows the guidelines of the chosen framework and will be converted into native code for the desired platform(s) with the same functionality.

2.2.1.2 Web stack

Web stack uses a web engine to render elements and execute the logic in a browser instance, without the browser's frame around it, to ensure an immersive experience. Using a web engine means that web developing languages, such as JavaScript, HTML and CSS, are used. Most of web stack frameworks are built on top of Cordova Hybrid platform, which means that web developing languages run on top of Apache Cordova programming language.

2.2.2 Existing solutions

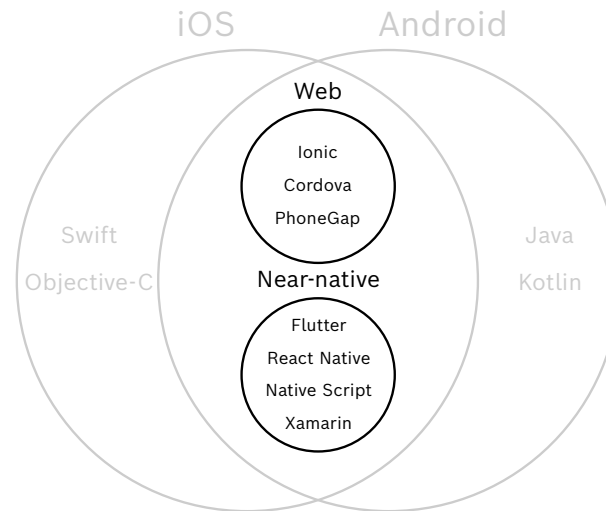


Figure 2.3: Most used hybrid frameworks and native languages

Figure 2.3 illustrates not only the most common hybrid frameworks in use, but also the native languages supported by both *iOS* and *Android* systems. It's important to highlight that React Native is implemented using React and JavaScript, Native Script using JavaScript and TypeScript, Flutter using Dart and Xamarin using C#.

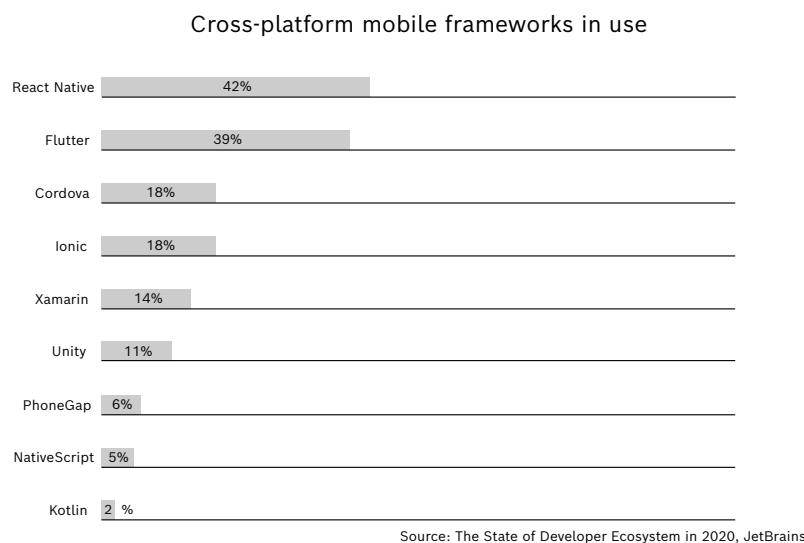


Figure 2.4: Cross-platform mobile frameworks in use by market share

Figure 2.4 shows the cross-platform mobile framework adoption in 2020 according to [10]. It's clear that React Native and Flutter are the leading cross-platform mobile frameworks. According to [10], the popularity of Flutter has increased over the last year, as its share has grown by 9%. Although its adoption has risen in the last years, the scientific community is not unanimous in terms of which previous stated frameworks performs best. On one hand, Wu states that React Native performs better; on the other, Jagiello supports the opposite [11] [12]. Both agree that when compared to native application development, the reducing performance is not restricting and the two frameworks are viable options, as long as the reduced performance is acknowledged and allowed.

The percentage of usage of Cordova, Ionic, and Xamarin, which represent the most used frameworks below the two leading ones, declined by roughly 10% each [10]. It's worth noting that the three frameworks mentioned above are based on **web stack**, thus demonstrating the ongoing trend of **near-native stack's** developers preference.

This results support the choice of a **near-native stack** framework.

3. Proposed system solution

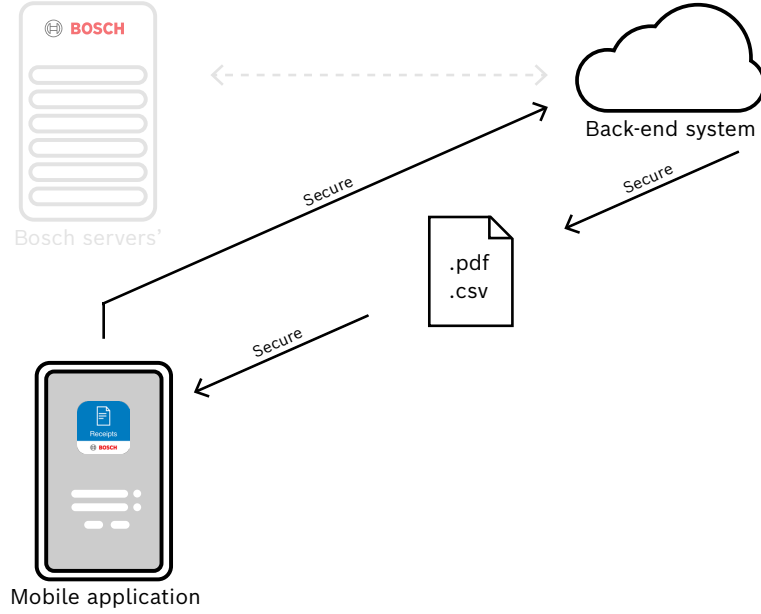


Figure 3.1: Proposed system overview

The system overview presented in Figure 3.1 illustrates the system's two main elements, as well as the data exchange between them. The communication path between the back-end system and Bosch's servers is also shown. This path is presented in section 1.3.1 as an optional functionality of the system, hence the different color and line type from the main communication path.

As a consequence of the characterization and research of the existing solutions for both system components, the chosen framework to implement the mobile application was **React Native** and the chosen OCR software was **Tesseract OCR**.

3.1 Mobile application

Firstly, by choosing an hybrid framework, the **requirement Requirement 1** is necessarily fulfilled, since the whole point of hybrid frameworks is to speed up the development process by following the motto *write once, run anywhere*.

Choosing **React Native** as the hybrid framework to support the application development is reasoned by the fact that it is the most used hybrid framework on the market, with tremendous community support, which results in optimal time to market[13] [10].

The amount and quality of React Native documentation available is another aspect worth mentioning [14].

Furthermore, an application developed with React Native is practically indistinguishable from a native one, from the point of view of both user experience and performance, particularly when taking into account that the application is graphically light, thus not requiring any type of 3D rendering or such [13] [15].

The successful implementation of the **requirements Requirement 2** and **Requirement 3** depends on the following, or not, of the design guidelines.

Requirements Requirement 4 and **Requirement 5** are focused on the code itself.

One way to fulfill the **requirement Requirement 4** is to implement an adequate data-structure, such as a queue, that has objects of type receipt. When the user finishes filling the details of the receipt and tries to upload the file to the back-end system, the application would push the receipt to a queue. The receipts would be popped from the queue as soon as a valid internet connection is acquired.

Regarding the **requirement Requirement 5**, by using a predictable state container such as Redux, the requirement is fulfilled.

3.2 Back-end

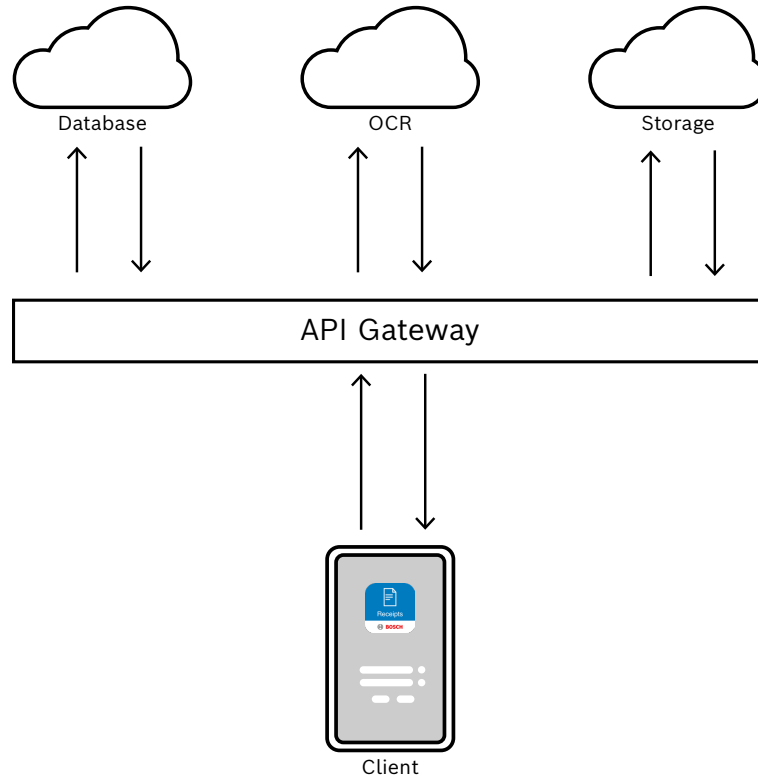


Figure 3.2: Proposed back-end architecture

The back-end system can be divided into 3 distinct micro-services: database, OCR and storage, integrated by an interface called API Gateway, as shown in figure 3.2.

A micro-services architecture presents a lot of benefits when compared to a monolithic one, specially when the objective is to achieve scalability. Consequently, the **requirement Requirement 6** is satisfied.

Mantoro, Sobri, and Usino ran a series of tests using standalone and server-based OCR on mobile devices, and compared the time, character and format recognition accuracy for each [16]. Server-based OCR obtained 5% higher character recognition accuracy than the standalone OCR, and a format recognition accuracy of 99.8% [16]. In short, the server-based OCR computation is quicker and more accurate, since the web server has more processing power than the mobile device. At the same time, it simplifies the process of selecting the OCR algorithm, since it is completely independent from the mobile platform.

Therefore, considering the results of Mantoro, Sobri, and Usino, the **requirement Requirement 9** should be fulfilled. Additionally, by implementing the OCR algorithm as a web service, the **requirement Requirement 8** is satisfied.

Regarding the **requirement Requirement 9**, since there is no maximum processing time associated with the requirement, it is up to the developer to implement practices that may speed up the processing time.

This practices can be in multiple domains, such as:

- Provide custom language-specific trained data based on receipt inputs;
- Implement concurrent processing;
- Ensure the OCR processing is running in a resource-heavy machine.

Requirement Requirement 10 can be contended by ensuring that, in case of processing errors during the OCR computation, the user is notified. Once more, this can be achieved through various methods. One method to accomplish this is to add a flag to the receipts' database entry that states whether or not there was an error during the OCR processing. This way, asynchronous processing is still possible and there isn't a need to route an answer from the OCR micro-service to the client, through the API Gateway.

The choice of the OCR algorithm followed the same principles as the choice of the cross-platform framework. **Tesseract OCR** presents itself among the top OCR systems worldwide, since it was initially announced in 1985 [8]. The community around it is very active, with more than 130 contributors and almost daily updates [17].

Another important reason is that it is an open-source software, which fulfills the **requirement Requirement 7**. This requirement is of particular significance because the information handled by the OCR processing algorithm is confidential, so the use of external services is to be avoided. Besides that, the support of more than 100 languages in the JS version, **Tesseract.js**, meets the **requirement Requirement 11**.

Concerning the OCR extraction accuracy, defined in the **requirement Requirement 12**, tests will be carried out to conclude if the the accuracy justifies the change of Bosch's reimbursement protocol. It is worth mentioning that, by providing custom trained data for each supported language, based on similar data to the one being processed, which means training the OCR engine with receipts, the accuracy can be vastly improved.

Requirement Requirement 13 shall be implemented with special focus on the extraction of the total receipt value.

3.3 Data storage

To ensure that the data is securely stored, it should be encrypted. Additionally, sensitive data, namely passwords, should be hashed. Data stored temporarily on the client side should be encrypted as well. By doing this, **requirement Requirement 14** is fulfilled.

3.4 Communication

Not only the data should be securely stored, but also securely exchanged between the client, the API Gateway and the implemented micro-services. A way to meet **requirement Requirement 15** is to make sure all communication is done through HTTPS, that makes use of SSL/TLS, thus ensuring privacy and integrity of the data by encrypting it. In addition, the implementation of token based authentication with a strictly defined expiration date and a renewal protocol may prove to be useful.

4. System implementation

4.1 Mobile application

4.1.1 UI Prototypes

This section characterizes, in detail, each mobile application's prototype screen.

4.1.1.1 For You screen

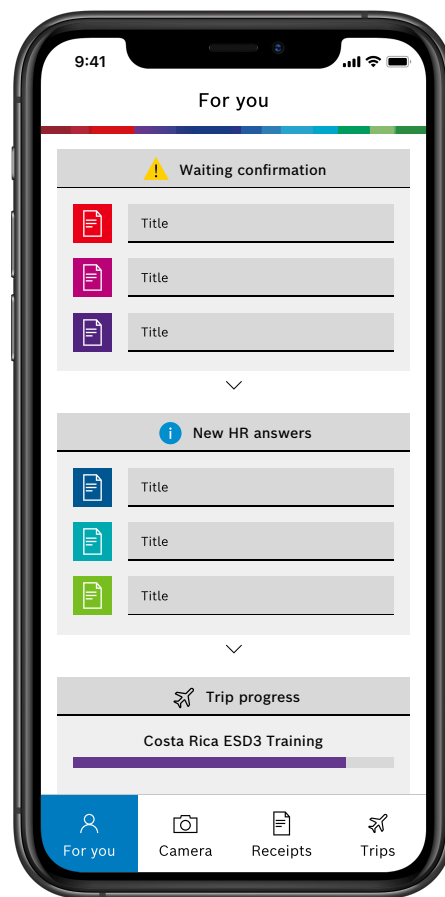


Figure 4.1: UI Prototype: For You screen

The proposed **for you screen** acts as landing page after the login. It can be seen as a dashboard that shows the user's receipts that require immediate action to further advance in the processing stage. This is possible by fetching all associated receipts and then order them chronologically so the receipts that are waiting for user confirmation, or just received a reply from the HR, are shown.

4.1.1.2 Camera screen



Figure 4.2: UI Prototype: Camera screen

The **camera screen** prototype shown is almost identical to the first prototype version. On the bottom left, the *landscape* icon opens the phone's gallery so that a previously taken photo, already saved in the phone's camera roll, can be used as input. On the bottom right, the *flash* icon enables or disables the flash, based on its previous state. To take a picture, the user can intuitively touch any other part of the screen.

4.1.1.3 Receipt submission screen

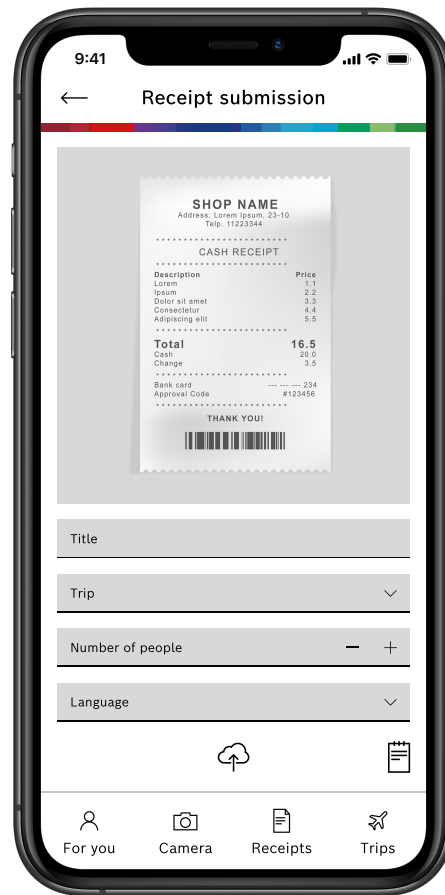


Figure 4.3: UI Prototype: Receipt submission screen

The proposed **Receipt submission screen** shows a preview of the chosen picture, four input fields and two icons. The input fields are self explanatory, given the placeholder text shown: *title*, *trip*, *number of people* and *language*. On the bottom right, the *notepad* icon opens a pop-up field so that the user can add an additional comment. When touched, the bottom centered icon opens a pop-up action sheet that asks the user if the receipt is ready for submission, and, if so, the receipt is uploaded to the database.

4.1.1.4 Receipt overview screen

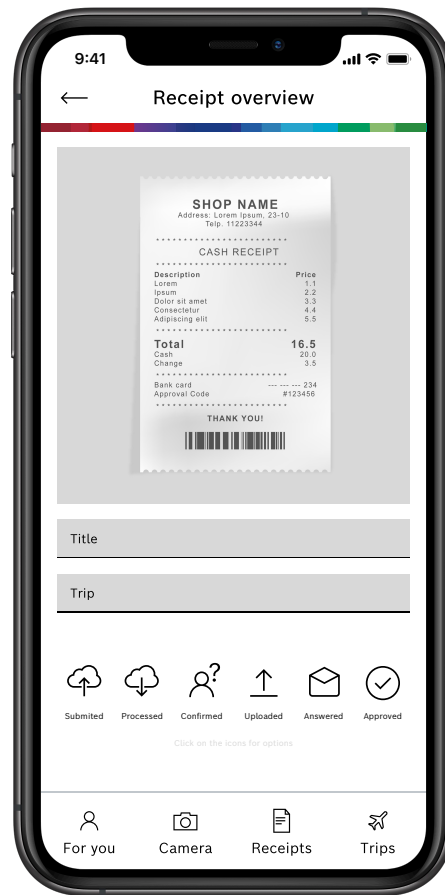


Figure 4.4: UI Prototype: Receipt overview screen

The shown prototype for the **receipt overview screen** closely resembles the **receipt submission screen**. The main difference is that *title* and *trip* fields are not modifiable and below them the receipt submission stages are shown. To open the available options for each stage, the user must touch the associated icon. For accessibility reasons, in case the icon is not obvious, the stage name is shown below the corresponding stage icon.

4.1.1.5 Receipts screen

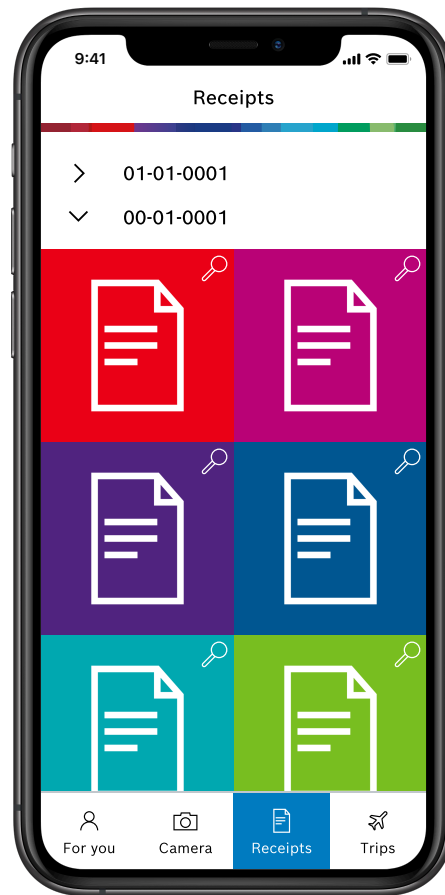


Figure 4.5: UI Prototype: Receipts screen

Figure 4.5 shows the developed prototype of the **receipts screen**. This screen displays the collection of receipts associated with the user's account. When the user touches a receipt, the respective receipt's overview screen is opened. For accessibility reasons, a small magnifying glass icon was added at the top right of each receipt to induce the user that an action will happen when the receipt is touched.

4.1.1.6 Trips screen

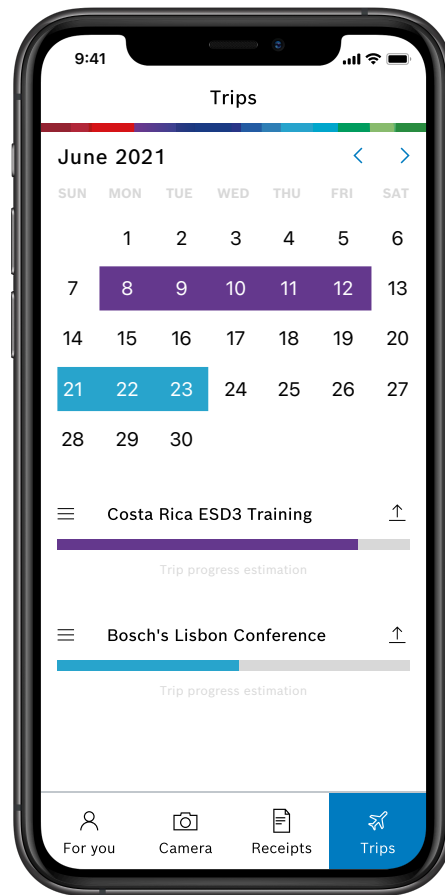


Figure 4.6: UI Prototype: Trips screen

The shown prototype for the **trips screen** presents a calendar and planned user's trips. Apart from that, for each trip, a progress bar is also shown, based on the number of receipts submitted and the trip day. This calculation takes into account that a user is supposed to submit two receipts per day.

4.1.1.7 Side menu screen

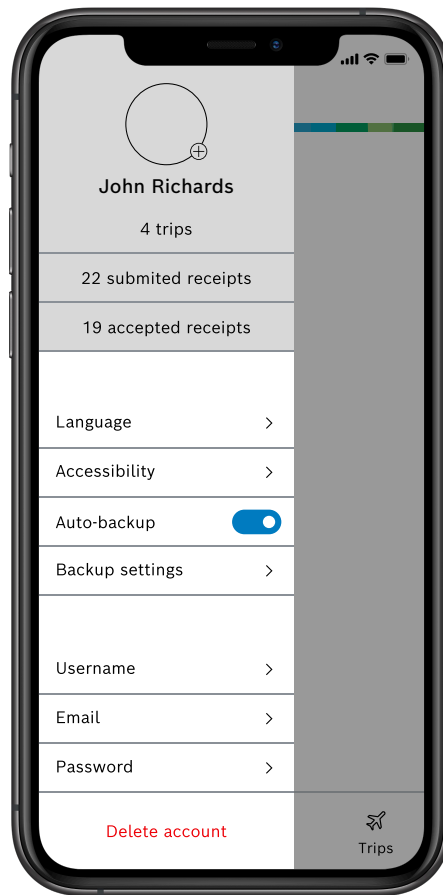


Figure 4.7: UI Prototype: Side menu screen

The **side menu screen** prototype presents the user's avatar, the user's first and last names, account statistics and options.

4.1.2 UX Prototypes

For better understandability of the pages' flow, four different user flow examples are shown for each main screen. It is worth mentioning that, the **Side menu screen** is accessible by performing a screen swipe from any point of the left side of the screen to the middle.

4.1.2.1 For You screen

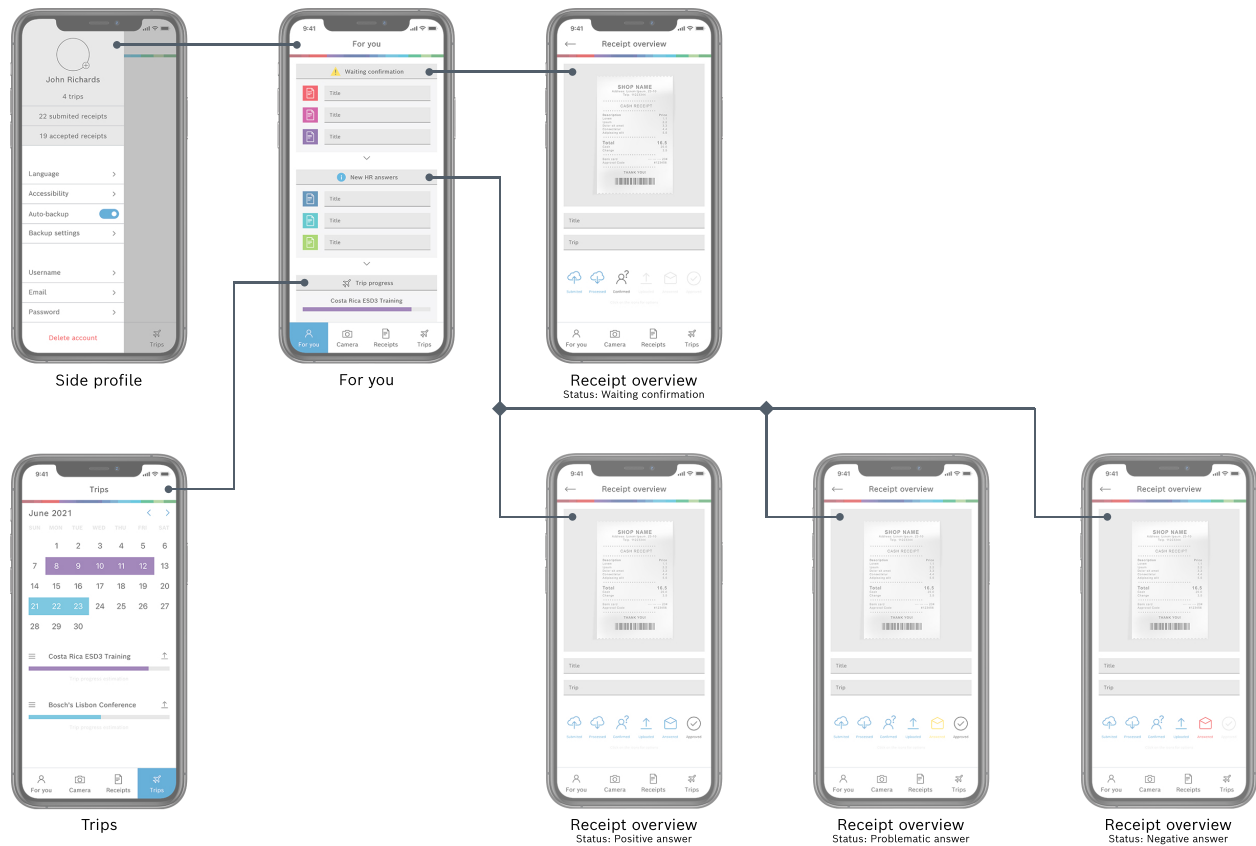


Figure 4.8: UX Prototype: For You screen

The **For You** screen leads to two additional screens, besides the **Side menu** screen:

- **Receipt overview** screen
- **Trips** screen

Depending on the stage of the receipt, the **Receipt overview** screen shows the corresponding stage, hence why 4.8 shows multiple examples of the screen.

4.1.2.2 Camera screen

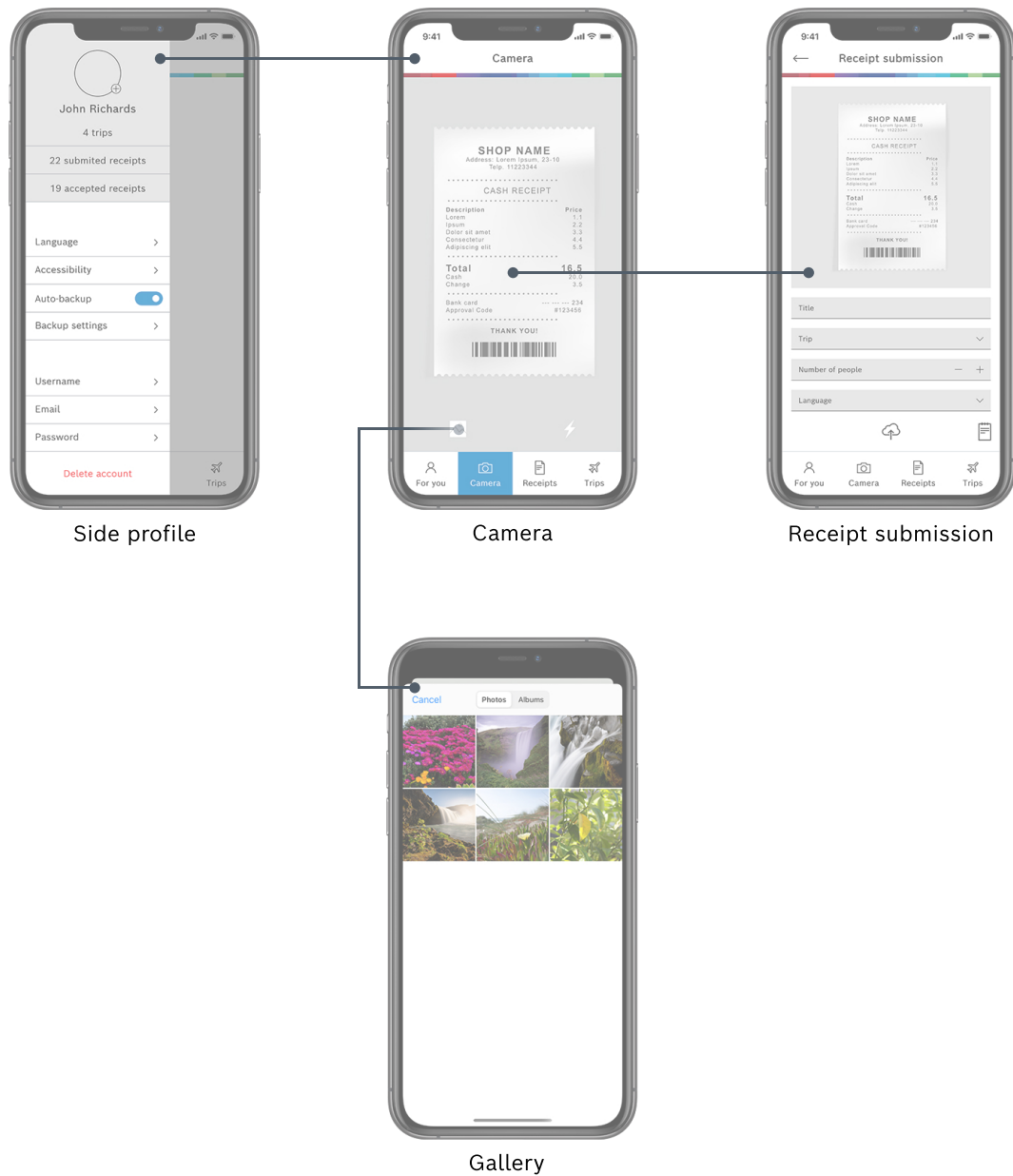


Figure 4.9: UX Prototype: Camera screen

The **Camera screen** leads to two additional screens, besides the **Side menu screen**:

- **Receipt submission screen**
- **Gallery screen**

The **Gallery screen** is not part of the mobile application but from the operating system itself.

4.1.2.3 Receipts screen

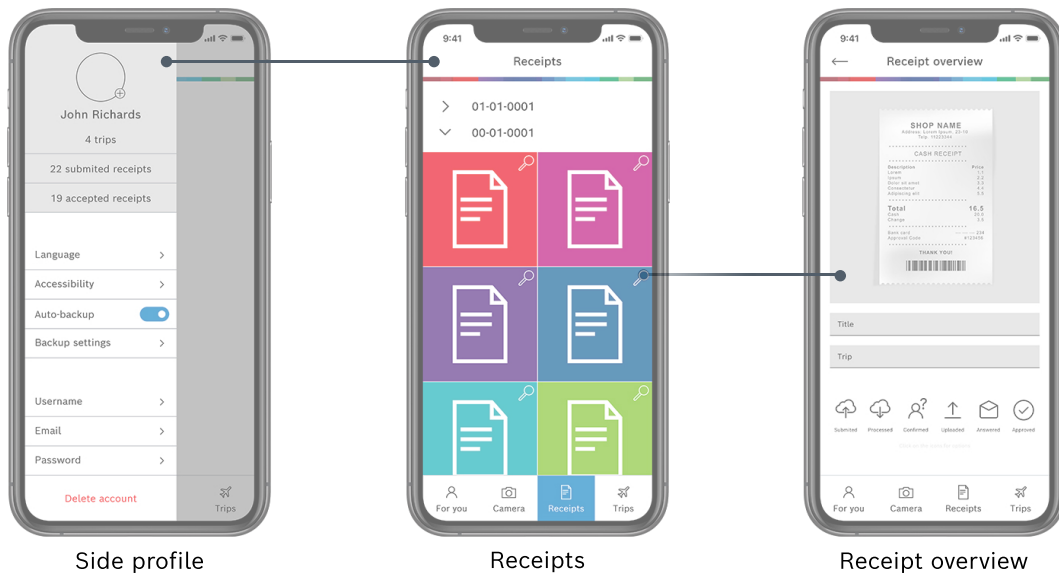


Figure 4.10: UX Prototype: Receipts screen

The **Receipts** screen leads to one additional screen, besides the **Side menu** screen: **Receipt overview** screen.

4.1.2.4 Trips screen

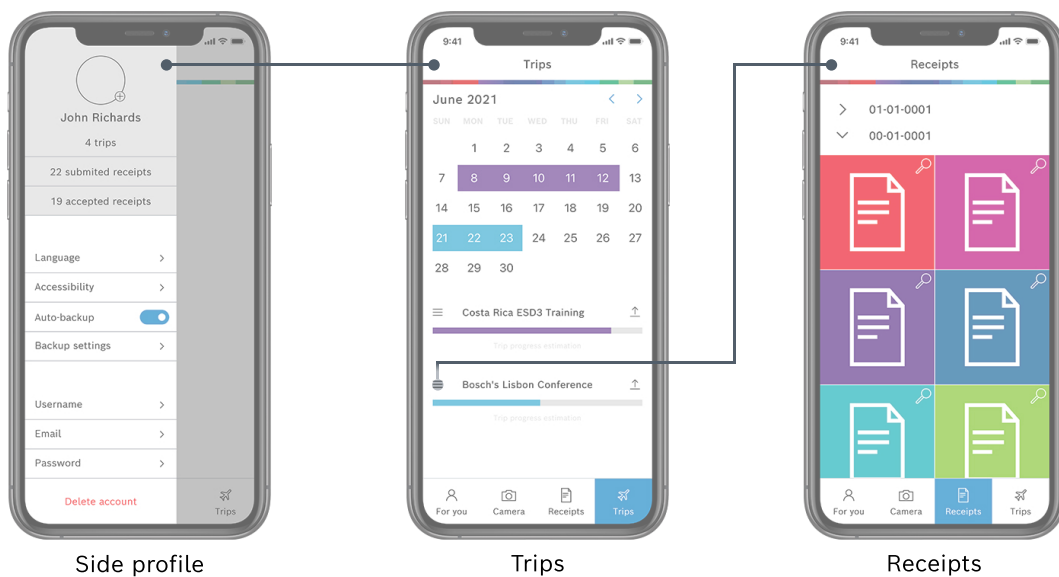


Figure 4.11: UX Prototype: Trips screen

The **Trips** screen leads to one additional screen, besides the **Side menu** screen: **Receipt** screen.

4.1.3 Technologies

This section briefly describes the technologies used to develop the mobile application.

4.1.3.1 Expo

Expo is a framework and a platform for React applications. It provides a set of tools and services built around React Native that help during all application development stages: development, build and deployment.

In addition, Expo also provides a quick and effective way to run the application on native platforms, such as *iOS* and *Android*, or in a simulated environment. If the developed application is running on a native platform, by default, the application is re-built every time the code is changed and saved, providing immediate feedback to the developer, accelerating the pace of development.

Moreover, Expo also features some libraries to speed up development. In this specific project, the following libraries are used:

- **Expo-status-bar**
- **Expo-font**
- **Expo-camera**
- **Expo-image-picker**
- **Expo-media-library**
- **Expo-secure-store**

4.1.3.2 Formik

Formik is a form library for React and React native. It provides form-level and field-level validation, custom-error messages, and other features. It is used to validate the forms with ease in the mobile application.

4.1.3.3 Axios

Axios is a promise based HTTP client. It provides a simple to use library in a small package with a very extensible interface, and is used both by the API Gateway and the mobile application to establish the connection between the two.

4.1.3.4 Redux

Redux is a predictable state container for JavaScript applications. It ensures the application's consistent behavior in its different stages.

4.2 Back-end

4.2.1 Technologies

This section briefly describes the technologies used to develop the back-end system.

4.2.1.1 Node

Node.js is a cross-platform, back-end, JavaScript runtime environment that makes possible executing JavaScript code outside a web browser. It makes possible developing server side applications and is used by both the API Gateway and the OCR service.

4.2.1.2 Express

Express is a back-end web application framework for Node.js that provides a robust set of features to build web applications and APIs. It is used by both API Gateway and OCR service.

4.2.1.3 Bcrypt

Bcrypt is a password-hashing function. The JavaScript library name is BcryptJS, which is an implementation of the generic hashing function in JavaScript. It is used by the API Gateway to hash the password before saving the user password to the database. The use of Bcrypt is detailed in the chapter 5.

4.2.1.4 Crypto

CryptoJS is a library that provides standard and secure cryptographic algorithms implemented in JavaScript using best practices and patterns. It is used by the API Gateway for generating new credentials when a forgot credentials request is received.

4.2.1.5 Jsonwebtoken

JWT is a standard that defines a way for securely transmitting information between parties as a JSON object. The information can be verified and trusted thanks to the digital signature. It is used by the API Gateway to send and verify client tokens. The use of JWT is detailed in the chapter 5.

4.2.1.6 Cors

CORS is a Node.js package that provides a middleware, used in express, that enables CORS. It is used by the API Gateway to be able to receive requests from another origin, in other words, the client.

4.2.1.7 Aws-sdk

Aws-sdk provides the API to connect user developed apps to the AWS services. It is used by both the API Gateway and the OCR service.

4.2.1.8 Nodemailer

Nodemailer is a module for Node.JS applications that allows effortless email sending. It is used by the API Gateway to connect to the Twilio SendGrid service.

4.2.1.9 Dotenv

Dotenv is a module that loads environment variables from a .env file into process.env. It is used by the API Gateway and the OCR service to load easily configured values, such as access keys.

4.2.1.10 Mongoose

Mongoose is a MongoDB schema-based object modeling tool. It is used by the API Gateway for all API requests that are related to database operations.

4.2.1.11 Mongodb

Mongodb is the official MongoDB driver for Node.js. It is used by the OCR service to establish the change stream connection to the database and process the receipts as soon as a new one is added.

4.2.1.12 Tesseract.js

Tesseract.js is a pure Javascript implementation of the popular Tesseract OCR engine. It is used by the OCR service to process the receipts.

4.2.2 Cloud services

4.2.2.1 Heroku

Heroku is PaaS that enables developers to build, run, and operate applications entirely in the cloud.

4.2.2.2 MongoDB Atlas

MongoDB Atlas is a SaaS database service that can be deployed on the most popular cloud service providers: AWS, Microsoft Azure, and Google Cloud.

4.2.2.3 AWS S3

AWS S3 is a SaaS storage that enables easy deployment of scalable storage solutions.

4.2.2.4 Twilio SendGrid

Twilio SendGrid is a SaaS email provider that enables uncomplicated email delivery through different protocols. SMTP is one of the supported protocols but for this project Twilio SendGrid email API was implemented.

4.3 Problems & solutions

4.3.1 React Native & Node.js familiarization

Since the student was not familiarized with any of the technologies, the chosen solution was to start with small projects that would later be included in the final work. Particularly, **react-navigation**, **expo-camera** and **expo-image-picker** libraries were implemented, as well as CSS styling, for a better understanding of the underlying foundation of the application. Furthermore, an early version of the data extraction and processing service was successfully implemented with the **Tesseract.js** library.

4.3.2 Mobile application: UI & UX

UI and UX are two of the most important aspects of a mobile application from the perspective of the end user. Considering that the student has an engineering background, thus lacking specific skills in those domains, guidance was requested from Bosch's experienced associates to overcome this difficulty.

A traditional design methodology was not followed, since the design was not a product of the conventional steps: wireframe, mockup and then prototype, but rather started with a basic prototype and was updated with incremental changes. The first and third iteration of this prototypes can be found on sections .1 and .2, respectively, part of the appendices.

Mandatory compliance with Bosch's design standards, by following Bosch's Brand Guide, emerged as a new requirement, which resulted in the development of the final version of the prototype, that can be found on the sections 4.1.1 and 4.1.2.

4.3.3 Mobile application: Code bugs

As expected during the development, bugs were found. To mitigate the defects, the student relied on books and online resources. Nevertheless, there is still one bug in the mobile application that is unresolved: the stack order doesn't behave as expected when a page is opened from the **side menu screen**. The behavior doesn't follow any apparent logical pattern since different pages, with exactly the same code and the same hierarchy position in the stack, lead to different results: in one there is a go back button, located at the top left of the screen, and in the other there is not. The problem was posted on *stackoverflow* on 7th of June and has remained unanswered.

4.3.4 Time management

Due to not only the lack of expertise from the student in the required technical domains for the successful development of the project but also from the scope of the project itself, specially when considering the whole project was to be done in a rather small time frame, special attention to time management was taken.

The initial planning contemplated different phases. From a front-end perspective, the first three weeks would be spent learning the required skills to develop the project, specifically React Native. From a back-end perspective, the first five weeks would be spent investigating the available services, APIs, and familiarizing with the chosen cloud service provider. Simultaneously, during the third week of the project, Bosch's *onboarding* activities would take place. After that, approximately two and half months would be spent implementing firstly the **main features** and then the **secondary features**. Documentation would take place during the implementation phase, and the last three weeks of the project would be allocated to the implementation of **optional features**. A Gantt chart was developed and is presented in figure 4.12.

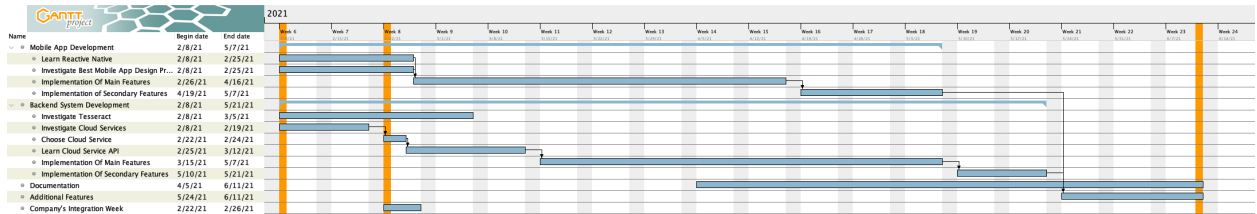


Figure 4.12: Initial project planning

In reality, the first two weeks were spent familiarizing with JavaScript and choosing the cloud provider for the server deployment. Bosch's *onboarding* activities took place not only during the third week of the project but rather during the third and the fourth weeks. The activities were not compatible with simultaneous project developments since physical attendance at Bosch Car Multimedia, Braga, was mandatory. Mandatory attendance was not planned, but due to Bosch's work policy, associates can only remotely attend company activities if they were already given a company laptop, which was not the case. After the *onboarding*, for about two weeks, successful implementation of specific parts of the code took place to guarantee a smooth start. During daily meetings some ideas came up, as well as new requirements. Consequently, one major project delay surfaced - the development of Bosch's Brand Guide compliant UI prototypes. Approximately five weeks were spent working in this new requirement, with three incremental version updates and one final version, developed from scratch with the company's recommended software, Sketch. The development of the mobile application and the back-end system took place simultaneously during approximately one and a half months. This resulted in a significant reduction in development time, which ultimately led to some features not being implemented as originally planned and a rather extensive list of improvements suggestion. Figure 4.13 shows the sequence of events that was described.

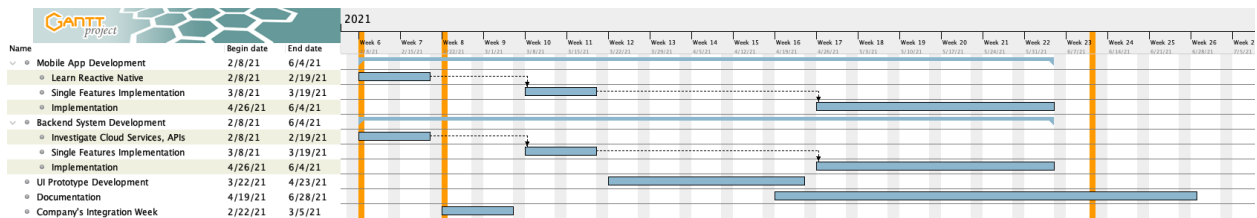


Figure 4.13: Project timeline

When figure 4.12 and 4.13 are compared, two key differences should be mentioned:

- The initial plan contemplated a lot of parallel work in different areas of the project; Figure 4.13 shows that parallel work only took place almost half way through the project development, having in mind that the parallel actions in the beginning were not necessarily parallel but rather a variety of experiments that encompass both development domains;
- Figure 4.12 proposes an overload of work in the beginning, with progressive slow down of tasks at hand; The actual course of actions was exactly the opposite.

In the end, it is clear that the project scope was massively underestimated and the lack of preparation regarding the anticipation of setbacks is undoubtedly the result of the student's inexperience in dealing with projects as complex as this one.

5. Developed system

5.1 Mobile application

5.1.1 UI

This sections presents a overview of the developed UI of the mobile application.

5.1.1.1 Login screen

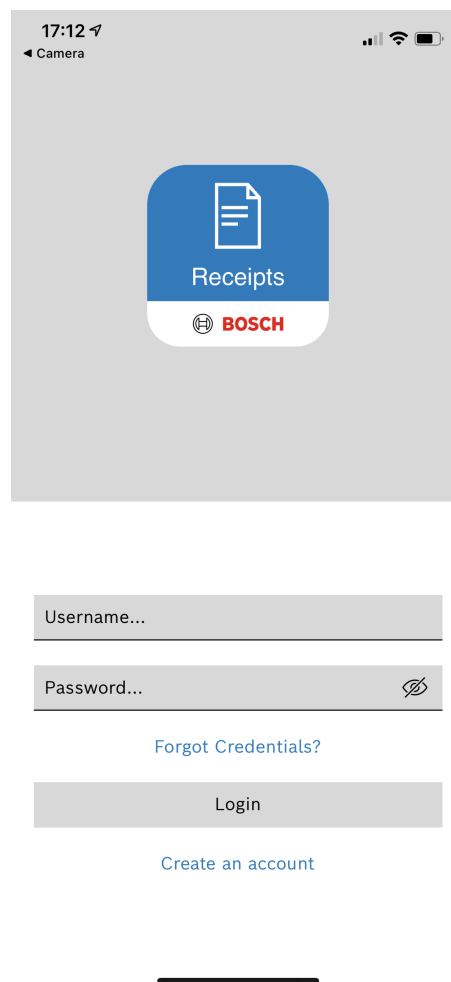


Figure 5.1: UI: Login screen

The **Login screen** shows is divided in two sections:

- The applications logo, on the top;
- The input fields and action buttons, on the bottom;

The input fields and action buttons presented are standard in any application that requires authentication.

5.1.1.2 Forgot credentials screen

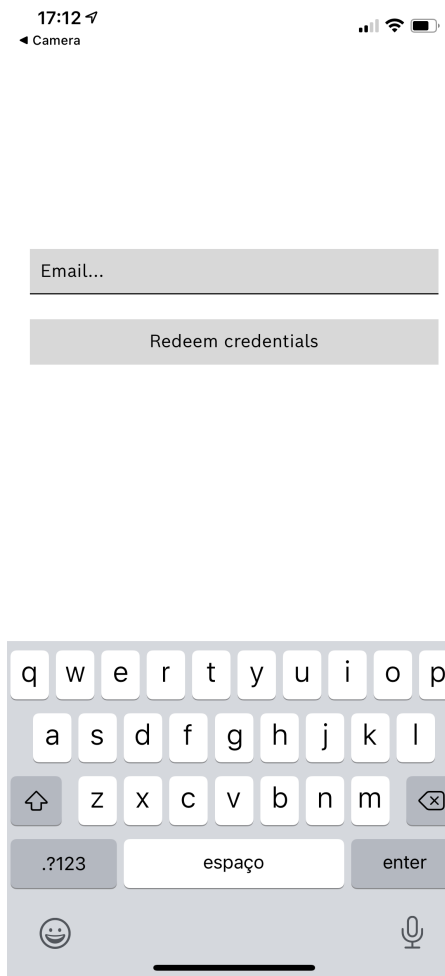


Figure 5.2: UI: Forgot credentials screen

The **Forgot credentials screen** only has an input field, for the email, and a button. As mentioned for the **Login screen**, this screen is also standard in any application that requires authentication.

5.1.1.3 Register screen

17:12 ◀
◀ Camera

First name...

Last name...

Username...

Email...

Password...

Confirm password...

Register

[Reset fields](#)

q w e r t y u i o p
a s d f g h j k l
⬆ z x c v b n m ⬇
.?123 espaço seguinte
😊 🎤

Figure 5.3: UI: Register screen

The **Register screen** has input fields that correspond to the user's database schema, and two buttons: one to actually go forward with the registration and another to reset all fields' values. As mentioned for the **Login screen** and the **Forgot credentials screen**, this screen is also standard in any application that requires authentication.

5.1.1.3.1 Register form checker

17:12

Camera

D

AAAAAAAAAAAAAAAAAAAAAAAAAAAA

Must be 20 characters or less

Username...

Required

Email...

Email is required

Password...

Confirm password...

Register

Reset fields

Q W E R T Y U I O P

A S D F G H J K L

↑ Z X C V B N M ↵

123 espaço seguinte

😊 🎤

Figure 5.4: UI: Register screen - Maximum length validation

17:13

Camera

André

Aragão

aragao

aragao

Email is invalid

Password...

Confirm password...

Register

Reset fields

Suggested Username
andribeiroaragao@gmail.com

Suggested Username
admin

q w e r t y u i o p

a s d f g h j k l

↑ z x c v b n m ↵

123 espaço seguinte

😊 🎤

Figure 5.5: UI: Register screen - Email validation

Figures 5.4, 5.5, 5.6 and 5.7 show the form validation in action. The form and field validation was implemented in the front-end using **Formik**.

Yup is a JavaScript schema builder for value parsing and validation. **Formik** documentation recommends the use of **Yup** library, although similar libraries can be used over it. Listing 5.1 shows the field restrains as well as the custom displayed messages.

17:13 Camera

André

Aragão

araragao

araragao@gmail.com

testpassword

testpass

Password and confirm password should match

Register

Reset fields

Figure 5.6: UI: Register screen - Password matching validation

17:13 Camera

André

Aragão

araragao

araragao@gmail.com

testpassword

testpassword

Register

Reset fields

Figure 5.7: UI: Register screen - Password matching validation (2)

Listing 5.1: Mobile application: Formik form validator

```

1  const FormValidationSchema = Yup.object({
2    firstname: Yup.string()
3      .max(15, 'Must be 15 characters or less')
4      .required('Required'),
5    lastname: Yup.string()
6      .max(20, 'Must be 20 characters or less')
7      .required('Required'),
8    username: Yup.string()
9      .max(20, 'Must be 20 characters or less')
10     .required('Required'),
11    email: Yup.string()
12      .email('Email is invalid')
13      .required('Email is required'),
14    password: Yup.string()
15      .min(8, 'Password must be at least 8 charaters')
16      .required('Password is required'),
17    confirmPassword: Yup.string()
18      .oneOf([Yup.ref('password'), null], 'Password and confirm password ...
19        should match')
20      .required('Confirm password is required'),
21  })

```

5.1.1.4 For You screen

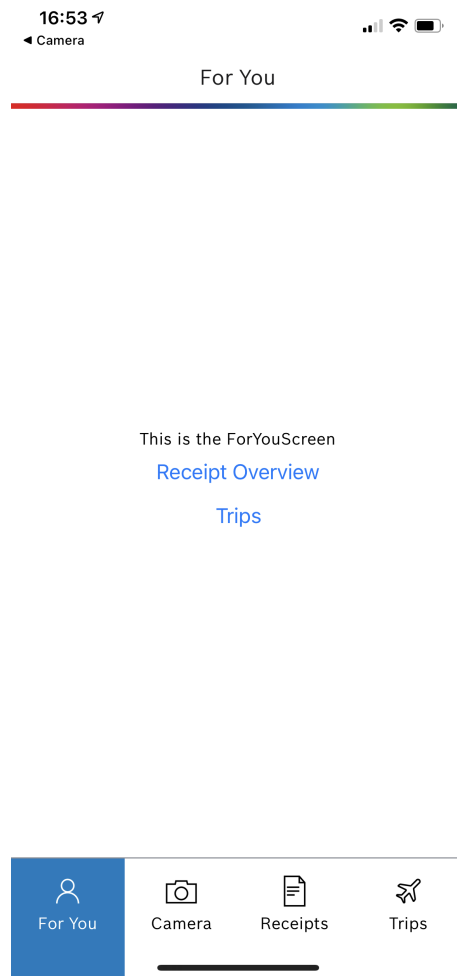


Figure 5.8: UI: For You screen

The **For You screen** was not completed since the dynamic squares that display the user's receipts and the respective stages were not implemented due to time constraints. Nevertheless, the stack that implements the **For You screen** is ready for the completion of the screen's features as the navigation to both **Receipt overview screen** and **Trips** screen is already implemented.

5.1.1.5 Camera screen

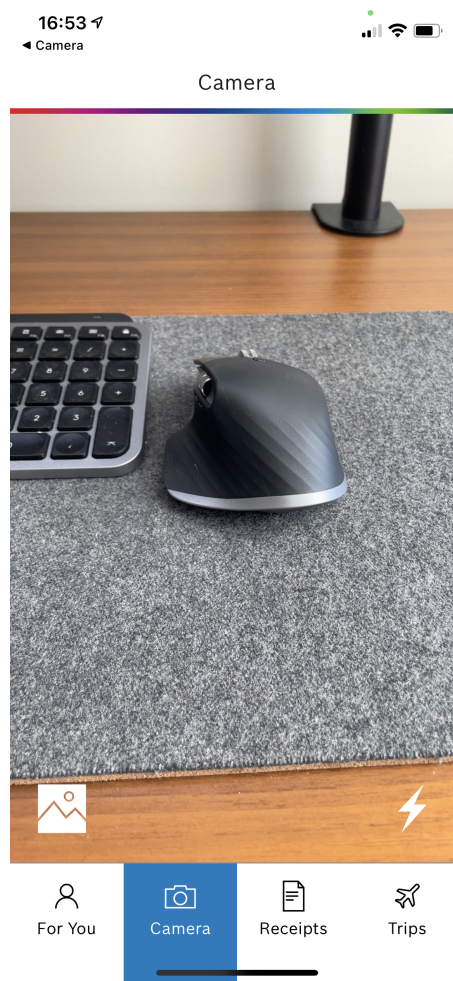


Figure 5.9: UI: Camera screen

The **Camera screen** is practically indistinguishable from its prototype version.

5.1.1.6 Receipt submission screen

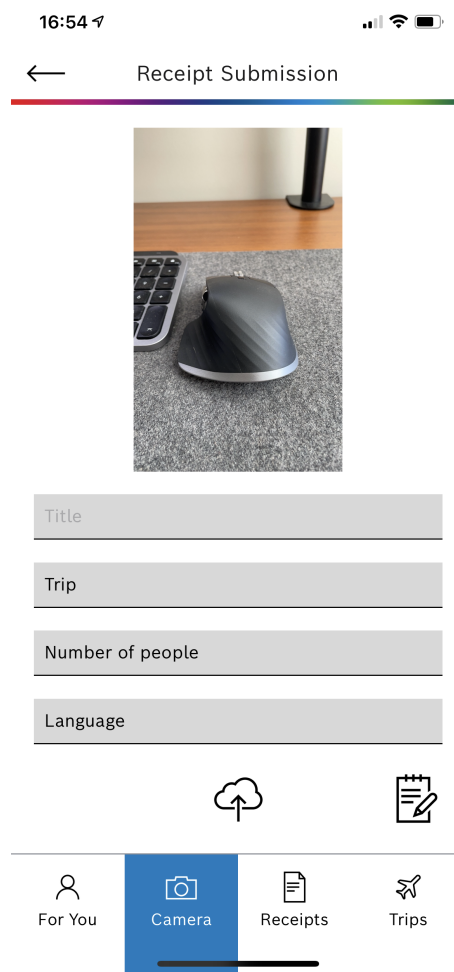


Figure 5.10: UI: Receipt submission screen

The **Receipt submission screen** is practically indistinguishable from its prototype version.

5.1.1.7 Receipt overview screen

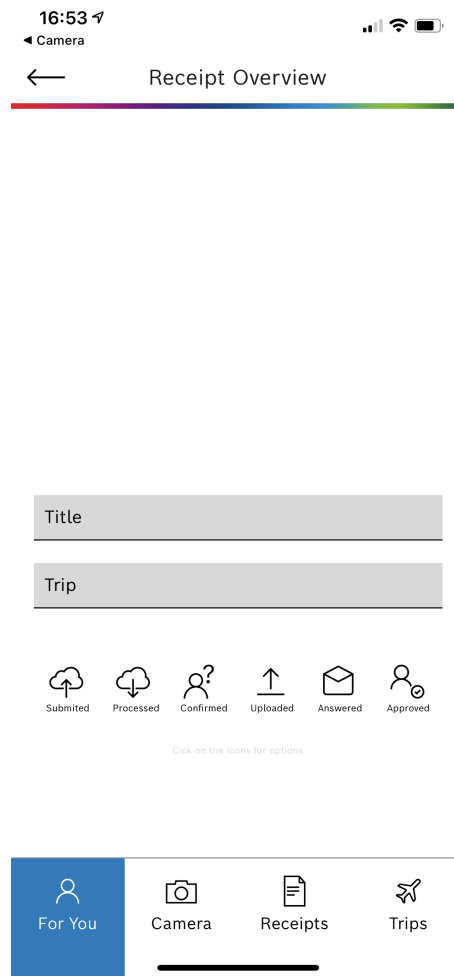


Figure 5.11: UI: Receipt overview screen

The **Receipt overview screen** is practically indistinguishable from its prototype version.

5.1.1.8 Receipts screen

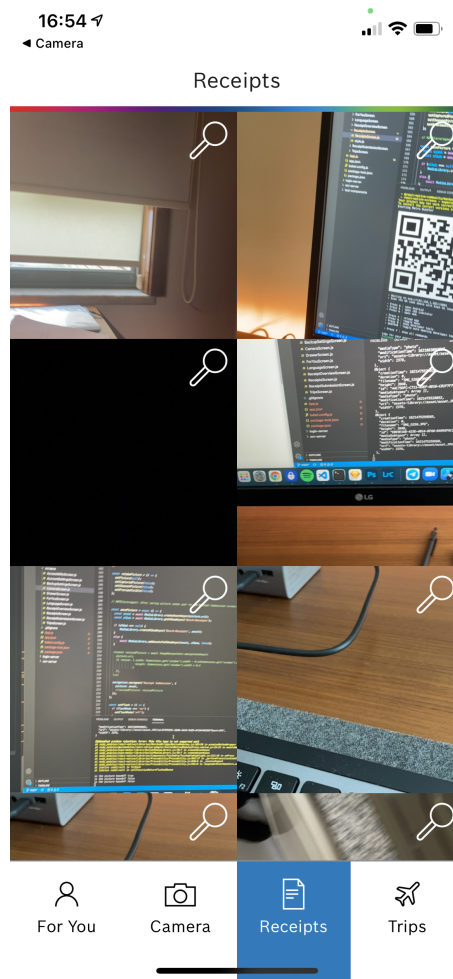


Figure 5.12: UI: Receipts screen

The **Receipts** screen is practically indistinguishable from its prototype version.

5.1.1.9 Trips screen

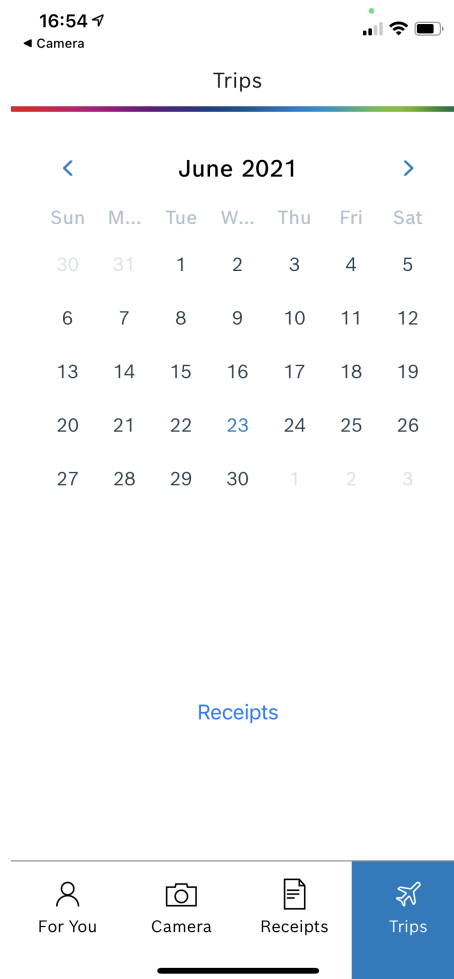


Figure 5.13: UI: Trips screen

The **Trips** screen is not completed as there is no way of showing trip details.

5.1.1.10 Side menu screen

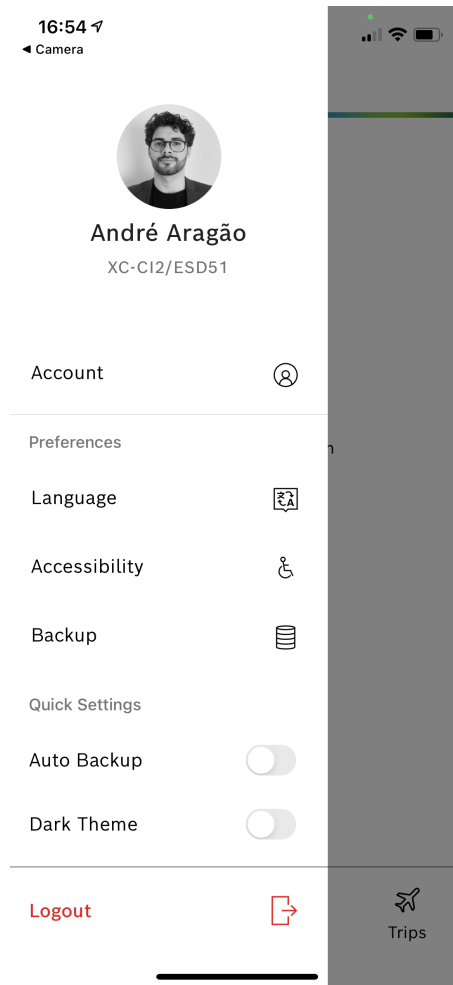


Figure 5.14: UI: Side menu screen

The **Side menu screen** was remarkably improved from its prototype version. Besides displaying the associate's picture, the **Side menu screen** also displays the associate's department and section. Account settings are now available in one single screen, which can be accessed by touching **Account** on **Side menu screen**. Although inoperable, a switch button that changes the application's theme was implemented by taking into account accessibility concerns. Lastly, the bottom button is no longer used to **Delete account** but rather to **Logout** as it is a much more convenient and frequent option. **Delete account** was moved to the **Account screen**.

5.1.2 UX

The developed UX meticulously follows the prototypes shown in section 4.1.2. As a result, the UX for the corresponding screens is not discussed again in this section. Nevertheless, three new screens were developed for authentication purposes - **login**, **forgot credentials** and **register screen**. Section 5.1.2.1 shows the screen flow between them.

5.1.2.1 Login screen

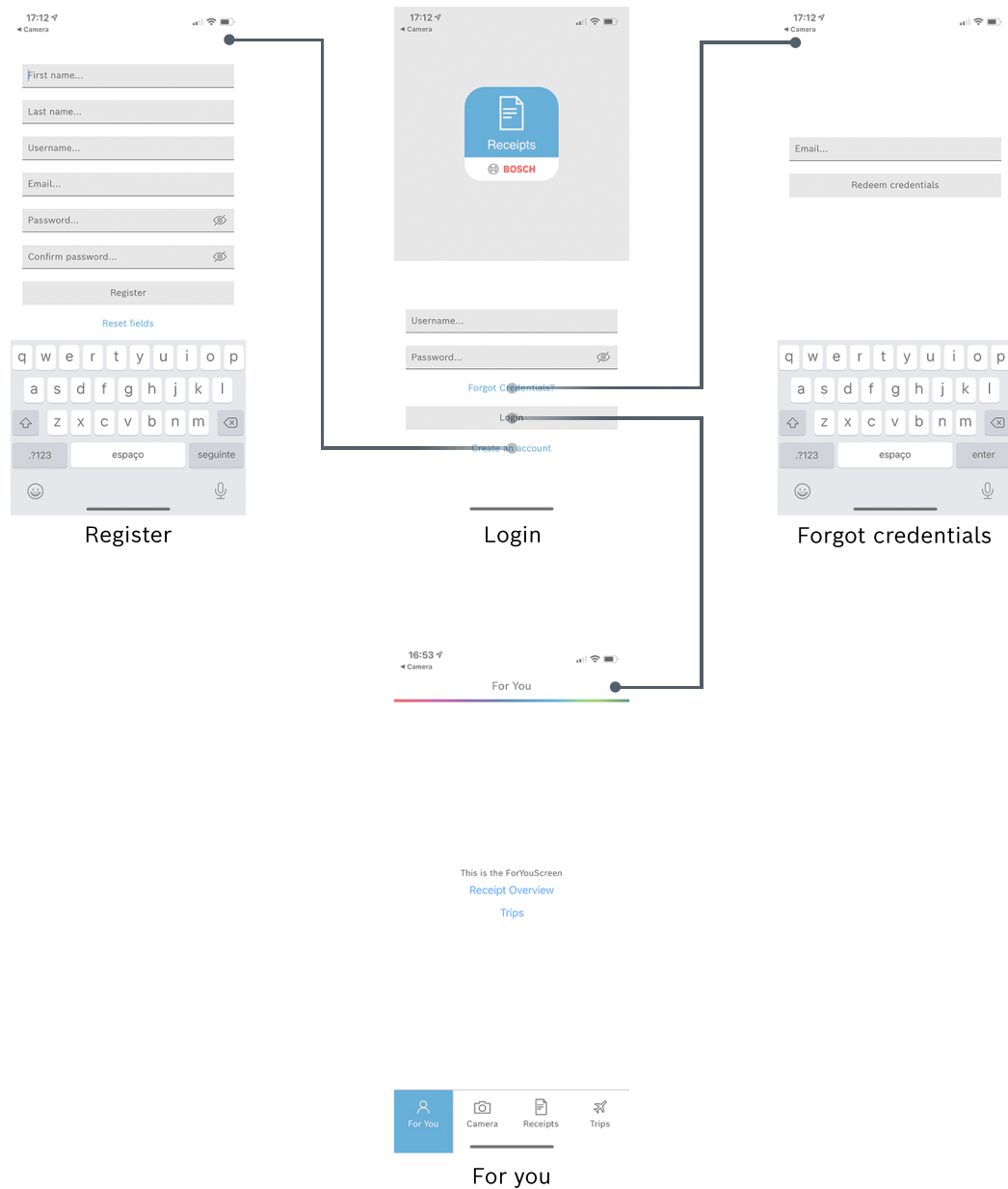


Figure 5.15: UX: Login screen

Figure 5.15 shows the navigation options available from the **Login screen**.

5.1.3 Security

The only aspect of the mobile application that requires additional security measures is the storage of the JWT token, necessary for the constant data exchange between the mobile application and the back-end system. It is of extreme importance given that the token is used to request information from the back-end system that otherwise cannot be requested.

In order to ensure that no other applications have access to the token, which could be wrongfully used, **SecureStore** was used. **SecureStore** provides a way to encrypt and securely store key–value pairs locally on the device and was implemented with the **expo-secure-store** library. To achieve encryption, **SecureStore** uses *keychain services*, in *iOS*, and *Android keystore system*, in *Android*.

5.2 Back-end

5.2.1 Architecture

The back-end was implemented following a micro-services architecture with the main objective being **scalability** of both services provided and processing power.

Since this project serves as a *skeleton* for Bosch to deploy mobile-centered applications, it was important to guarantee that the implemented architecture is easily scalable regarding the number of available services. At the same time, given the number of Bosch associates described in section 1.1.1, it is of utmost importance to guarantee that the system can process the massive number of requests that might occur.

The API Gateway was implemented as a **Node.js** server and deployed using Heroku. The data extraction and processing was implemented as a **Node.js** server that uses the **Tesseract.js** library to apply the OCR algorithm when a new receipt is submitted to the database, by listening to database changes. It was deployed using **Heroku**. The emailing service used was **Twilio SendGrid** for the ease of use and integration. For the database, **MongoDB Atlas** was used since it provides a cloud database storage that runs **MongoDB**. The storage service used was **AWS S3**.

Figure 5.16 presents the implemented architecture.

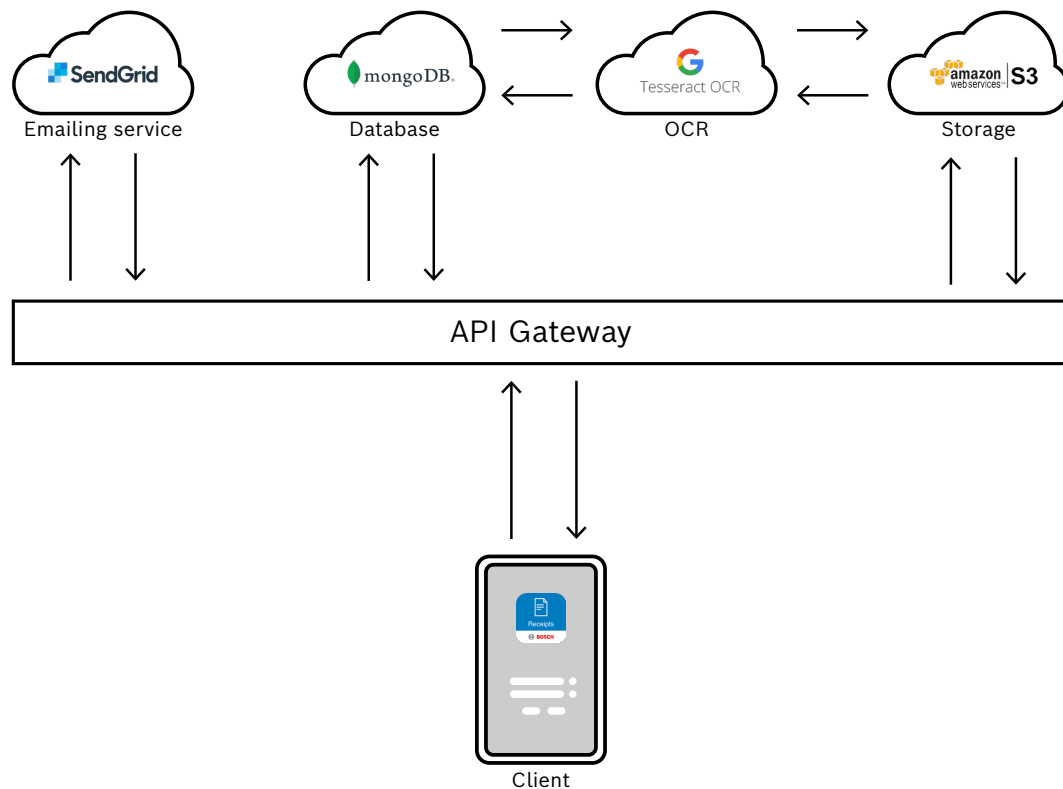


Figure 5.16: Back-end system architecture

It is important to mention that the usage of a storage service different than the database is essential to guarantee that the database doesn't get unnecessarily big. Since the data stored in the AWS S3 micro-service is substantially bigger than the database entries, if the data was directly stored in the database it would result in longer requests' processing time.

5.2.2 Database service

5.2.2.1 Database schemas

5.2.2.1.1 User

Associate details

- **firstname** (String)
- **lastname** (String)
- **username** (String)
- **email** (String)
- **password** (String)

Associate information

- **avatar** (Boolean)
- **department** (String)
- **team** (String)

Control

- **createdAt** (Date)

The **avatar** field is of type Boolean because its value is equal to the existence, or not, of a picture in the associated Amazon S3 bucket. If it has the value **true**, the image is stored with **username** as the object key. The relationship between the database entry and the storage data is detailed in section 5.2.3.1.

When a new database entry is created, **MongoDB** also adds two default fields, **__id** and **__v**. **__id** is the unique identifier of the database entry. It serves as the primary key of the database entry. **__v** is the versionKey of the database entry. versionKey is a property set that represents the internal revision version of the entry.

5.2.2.1.2 Receipt

Receipt details

- **title** (String)
- **trip** (String)
- **numberOfPeople** (String)
- **language** (String)
- **receiptDetails** (Boolean)
- **receiptDetailsContent** (String)

Messages

- **userConfirmationMessage** (Boolean)
- **userConfirmationMessageContent** (String)
- **hrMessage** (Boolean)
- **hrMessageContent** (String)
- **userReplyHRMessage** (Boolean)
- **userReplyHRMessageContent** (String)
- **userApprovalMessage** (Boolean)
- **userApprovalMessageContent** (String)

Picture

- **picture** (Boolean)
- **pictureKey** (String)

OCR

- **ocrError** (Boolean)
- **ocrData** (Boolean)
- **ocrDataKey** (String)

Control

- **stage** (String)
- **status** (String)
- **associatedAccount** (String)
- **createdAt** (Date)

The **receiptDetails** field is of type Boolean because its value is equal to the existence, or not, of the **receiptDetailsContent**. If it has the value **true** then **receiptDetailsContent** is not empty. **userConfirmationMessage**, **hrMessage**, **userReplyHRMessage** and **userApprovalMessage** respectively follow the same approach to **userConfirmationMessageContent**, **hrMessageContent**, **userReplyHRMessageContent** and **userApprovalMessageContent**.

The **stage** field can have different values according to the current stage of the receipt:

1. **Submitted**
2. **Processed**
3. **Confirmed**
4. **Uploaded**
5. **Answered**
6. **Approved**

The **status** field can have different values according to the status of the current stage of the receipt:

- **Success**
- **Problem**
- **Error**

The usage of both **stage** and **status** fields is explained in detail in section 5.3.1.

The **picture** field is of type Boolean because its value is equal to the existence, or not, of the **pictureKey**.

Listing 5.2 shows the code responsible for the generation of **pictureKey** which is the result of a concatenation between two strings: the username and the date. This method ensures that each **pictureKey** is unique. It is of extreme importance to guarantee that each **pictureKey** is unique since it will be used to retrieve the data from the storage service.

Listing 5.2: API Gateway: pictureKey generation

```
let today = new Date();
today = today.toISOString();
const pictureKey = req.user.username + '&'amp;' + today;
```

ocrData and **ocrDataKey** have the same relationship as **picture** and **pictureKey**. Moreover, the code used for the generation of **ocrDataKey** is the same as the one used for the generation of **pictureKey**, and is presented in listing 5.3.

Listing 5.3: OCR service: ocrDataKey generation

```
let today = new Date();
today = today.toISOString();
const ocrDataKey = next.fullDocument.associatedAccount + '&' + today;
```

5.2.3 Storage service

Although both the API Gateway and the OCR service use the AWS S3 service, they are two distinct servers. In order to develop code that can be re-used in both servers, the functions **uploadToS3**, **getFromS3** and **deleteFromS3** were written in a generic way.

Firstly, some variables should be defined. Since the defined variables are constant, and part of the servers' configuration, they were placed in the *config.env* file. An excerpt of the *config.env* file is presented in listing 5.4.

Listing 5.4: API Gateway & OCR service's environmental variables: AWS S3 service

```

1 # AWS CONFIGURATION PARAMETERS
2 AWS_ACCESS_KEY = *****
3 AWS_SECRET_KEY = *****
4
5 # AWS S3 CONFIGURATION PARAMETERS
6 AWS_BUCKET_REGION = eu-west-3
7 AWS_AVATARS_BUCKET_NAME = dmpr-avatars
8 AWS_PICTURES_BUCKET_NAME = dmpr-pictures
9 AWS_OCR_BUCKET_NAME = dmpr-services-ocr

```

The code listing 5.5 shows the implementation of the functions mentioned previously. The use of the library **aws-sdk** made the implementation simple and quick, although the available documentation, provided by Amazon, is not straight-forward and other guides had to be followed.

Since all message exchanges between the client, the API Gateway and the services in use are done through JSON, and JSON doesn't support picture encoding such as *.jpeg* or *.png*, the pictures have to be converted to the base64 format. The argument *isPicture* was implemented to ensure that, when needed, the conversion is done by **uploadToS3** and **getFromS3**, respectively from base64 to a binary buffer and from a binary buffer to base64, resulting in the main code being more organized and easy to read.

Listing 5.5: API Gateway & OCR service: AWS S3 operations implementation

```

1 import dotenv from 'dotenv';
2 dotenv.config({ path: './config.env' });
3
4 import AWS from 'aws-sdk';
5
6 const region = process.env.AWS_BUCKET_REGION;
7 const accessKeyId = process.env.AWS_ACCESS_KEY;
8 const secretAccessKey = process.env.AWS_SECRET_KEY;
9
10 const s3 = new AWS.S3({
11   region,
12   accessKeyId,
13   secretAccessKey
14 });
15
16 export const uploadToS3 = async (file, filekey, bucketName, isPicture) => {
17   let fileToSend, uploadParams;
18
19   if(isPicture) {
20     // NOTE(araragao): Conversion from base64 string to buffer
21     fileToSend = Buffer.from(file, 'base64');
22
23     uploadParams = {
24       Key: filekey,
25       Bucket: bucketName,
26       Body: fileToSend,
27       ContentEncoding: 'base64',
28       ContentType: 'image/jpeg',
29     }
30   } else {
31
32     fileToSend = file;

```

```
33     uploadParams = {
34         Key: fileKey,
35         Bucket: bucketName,
36         Body: fileToSend,
37     }
38 }
39
40 return await s3.upload(uploadParams).promise();
41 }
42
43 export const getFromS3 = async (fileKey, bucketName, isPicture) => {
44     const downloadParams = {
45         Key: fileKey,
46         Bucket: bucketName,
47     }
48     try {
49         const data = await s3.getObject(downloadParams).promise();
50         if(isPicture) {
51             // NOTE(araragao): Conversion from buffer to base64 string
52             return data.Body.toString('base64');
53         } else {
54             return data.Body;
55         }
56     } catch(error) {
57         return error;
58     }
59 }
60
61 export const deleteFromS3 = async (fileKey, bucketName) => {
62     const deleteParams = {
63         Key: fileKey,
64         Bucket: bucketName,
65     }
66     try {
67         await s3.deleteObject(deleteParams).promise();
68         return 'Object removed successfully';
69     } catch(error) {
70         return error;
71     }
72 }
```

5.2.3.1 Data linkage between database & storage service

As the database fields and the storage service operations have already been detailed, figure 5.17 serves the purpose of clarifying any doubt that might still exist and presenting a brief overview of both sections 5.2.2.1 and 5.2.3.

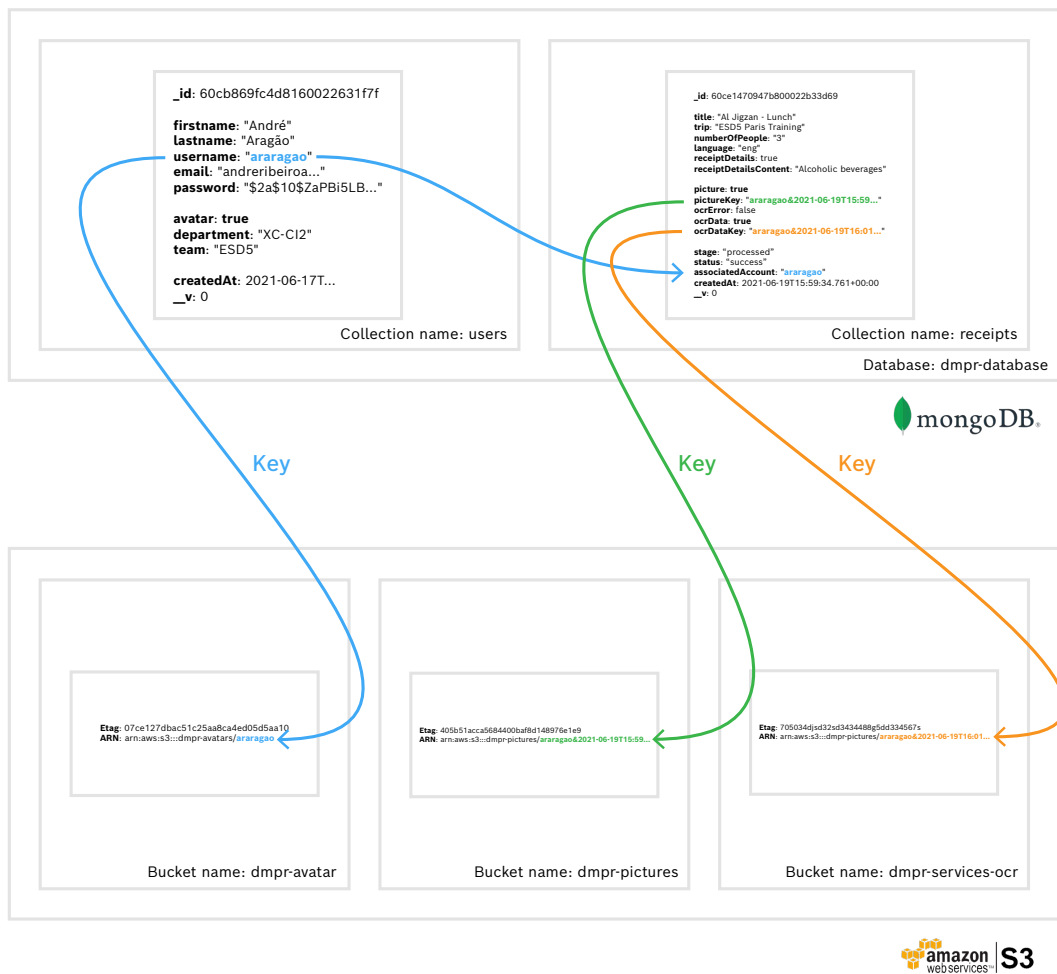


Figure 5.17: Data handling & linkage between micro-services

5.2.4 OCR Service

The OCR service was developed as a **Node.js** server. For that, **Express**, **Cors**, **Mongodb** and the developed functions mentioned in section 5.2.3 were used. However, the purpose of this section is to explain the implementation of the **Tesseract.js** library.

For performance reasons, a scheduler was used in order to deploy more than one worker to perform the OCR algorithm and the trained data is cached in a server's folder. This implementation methodology ensures faster results when compared to the use of a single worker and to the need of fetching the required language for each request. Listing 5.6 shows the creation of both the scheduler and the workers, as well as the worker configuration parameters to ensure the use of the cached trained data.

Considering that five workers are used by the scheduler, line 10 should be perceived as an omission of repetitive code. The same approach was used for 5.7 and 5.8.

Listing 5.6: OCR service: Worker and scheduler creation

```

1  const { createWorker, createScheduler } = require('tesseract.js');
2
3  const scheduler = createScheduler();
4
5  const worker1 = createWorker({
6      cacheMethod: 'readOnly',
7      langPath: './languages',
8  });
9
10 ...
11
12 const worker5 = createWorker({
13     cacheMethod: 'readOnly',
14     langPath: './languages',
15 });

```

Listing 5.7 shows the initialization process of the workers as well as the scheduler.

Listing 5.7: OCR service: Worker and scheduler initialization

```

1  await worker1.load();
2  ...
3  await worker5.load();
4
5  await worker1.loadLanguage(next.fullDocument.language);
6  ...
7  await worker5.loadLanguage(next.fullDocument.language);
8
9  await worker1.initialize(next.fullDocument.language);
10 ...
11 await worker5.initialize(next.fullDocument.language);
12
13 scheduler.addWorker(worker1);
14 ...
15 scheduler.addWorker(worker5);

```

Listing 5.7 shows the **tesseract.js** function that applies the OCR algorithm to the provided picture.

Listing 5.8: OCR service: Computing OCR algorithm

```

1  const { data: { text } } = await scheduler.addJob('recognize', ...
    Buffer.from(picture, 'base64'));

```

5.2.5 Emailing service

Firstly, the parameters configured in the SendGrid service should be defined. Since they are part of the server's configuration, they were placed in the *config.env* file. An excerpt of the *config.env* file is presented in listing 5.9.

Listing 5.9: API Gateway's environmental variables: Twilio SendGrid service

```

1 # SENDGRID CONFIGURATION PARAMETERS
2 EMAIL_SERVICE = SendGrid
3 EMAIL_USERNAME = *****
4 EMAIL_PASSWORD = *****
5 EMAIL_FROM = *****

```

The implementation of the SendGrid service required the use of the **Nodemailer** library. Although **Nodemailer** uses SMTP protocol as its main transport to deliver messages, sendMail transport was used to speed up the implementation process. The implementation is shown in listing 5.10.

Listing 5.10: API Gateway: Twilio SendGrid operation implementation

```

1 import nodemailer from 'nodemailer';
2
3 const sendEmail = (options) => {
4   const transporter = nodemailer.createTransport({
5     service: process.env.EMAIL_SERVICE,
6     auth: {
7       user: process.env.EMAIL_USERNAME,
8       pass: process.env.EMAIL_PASSWORD,
9     }
10  })
11
12   const mailOptions = {
13     from: process.env.EMAIL_FROM,
14     to: options.to,
15     subject: options.subject,
16     html: options.text,
17   }
18
19   transporter.sendMail(mailOptions, function(error, info) {
20     if(error) {
21       console.log(error);
22     } else {
23       console.log(info);
24     }
25   })
26 }
27
28 export default sendEmail;

```

Listing 5.11 shows an excerpt of the API Gateway's function **forgotCredentials** in which the usage of the presented function in listing 5.10 is shown. An email is sent using basic HTML styling. It is worth nothing that, since the passwords saved in the database are hashed by **bcrypt**, under no circumstances can they be retrieved, unless brute force, paired with an extraordinary processing power, is used. Therefore, a new password has to be generated. To do so, the **crypto** library is used. Line 1 of listing 5.11 shows the generation of the password.

Listing 5.11: API Gateway: Twilio SendGrid service usage

```
1 user.password = crypto.randomBytes(12).toString('hex');
2
3 const message = `
4 <h1>Credentials request</h1>
5 <p>Username=${user.username}</p>
6 <p>Password=${user.password}</p>
7 `;
8
9 try {
10   await sendEmail({
11     to: user.email,
12     subject: "Bosch Digital Receipts Management - Credentials request",
13     text: message,
14   });
```

5.2.6 API Gateway's API

5.2.6.1 Authentication API

register HTTP Method POST

URL path /auth/register

Protected No

login HTTP Method POST

URL path /auth/login

Protected No

forgotCredentials HTTP Method POST

URL path /auth/forgotcredentials

Protected No

5.2.6.2 User API

resetPassword HTTP Method PUT

URL path /user/password

Protected Yes

removeUser HTTP Method DELETE

URL path /user/remove

Protected Yes

setDepartment HTTP Method PUT

URL path /user/department

Protected Yes

getDepartment HTTP Method GET

URL path /user/department

Protected Yes

setTeam HTTP Method PUT

URL path /user/team

Protected Yes

getTeam HTTP Method GET

URL path /user/team

Protected Yes

setAvatar HTTP Method PUT

URL path /user/avatar

Protected Yes

getAvatar HTTP Method GET

URL path /user/avatar

Protected Yes

5.2.6.3 Receipts API

fetchAllReceipts HTTP Method GET

URL path /receipts

Protected Yes

getReceipt HTTP Method GET

URL path /receipt

Protected Yes

addReceipt HTTP Method POST

URL path /receipt

Protected Yes

getReceiptPicture HTTP Method GET

URL path /receipt/picture

Protected Yes

changeReceiptStatus HTTP Method PUT

URL path /receipt/status

Protected Yes

removeReceipt HTTP Method DELETE

URL path /receipt

Protected Yes

5.2.7 Security

5.2.7.1 Password hashing and salting with Bcrypt

Bcrypt is a password-hashing function that uses an algorithm derived from the symmetric-key block cipher Blowfish, Eksblowfish (Expensive Key Schedule Blowfish). Blowfish's key generation requires the pre-processing equivalent of encrypting about 4 kilobytes of text, which is very slow compared to other block ciphers. What might appear as a general disadvantage, when applied to password security, it is in fact a very useful feature, since the extra computational effort required guarantees protection against **dictionary attacks** (i.e. systematically entering every word in a dictionary as a password). Eksblowfish combines the Blowfish algorithm with a variable number of iterations, thus exploiting the expensive key setup phase of Blowfish to increase the workload and duration of hash calculations, further reducing threats from **brute force attacks**. Furthermore, **Bcrypt** incorporates a salt into the hash to protect against **rainbow table attacks**.

Equation 5.1 shows the **Bcrypt** generic result.

$$[\text{algorithm}][\text{cost}][\text{22 character salt}][\text{31 character hash}] \quad (5.1)$$

To further explain the implementation of **bcrypt**, equations 5.2 and 5.3 show an example input and output of **bcrypt**'s algorithm, respectively.

$$\text{password} \quad (5.2)$$

$$\text{\$2a\$10\$ZaPBi5LBWKhEzc8T7dqCJ.7RW8BtwhhHXGcYzAfqlFCGfDCDMU4F6} \quad (5.3)$$

2a The hash algorithm identifier, bcrypt

10 Cost factor ($2^{10} \Rightarrow 1,024$ rounds)

ZaPBi5LBWKhEzc8T7dqCJ. 16-byte salt, radix64-encoded to 22 characters

7RW8BtwhhHXGcYzAfqlFCGfDCDMU4F6 24-byte hash, radix64-encoded to 31 characters

MongoDB allows pre and post saving actions. In this specific case, **Bcrypt** is used to hash the password before saving it to the database. Listing 5.12 shows the implementation of the password hashing.

Listing 5.12: API Gateway: Bcrypt's password hashing

```

1 UserSchema.pre("save", async function (next) {
2   // NOTE(araragao): If the password did not change it shouldn't be rehashed
3   if(!this.isModified("password")){
4     next();
5   }
6
7   const salt = await bcrypt.genSalt(10);
8   this.password = await bcrypt.hash(this.password, salt);
9   next();
10 });
```

Listing 5.13 shows the implementation of the function that compares the received password with the database stored password, using the `bcrypt.compare()` function.

Listing 5.13: API Gateway: Bcrypt's password comparing

```

1 UserSchema.methods.matchPasswords = async function(receivedPassword) {
2   return await bcrypt.compare(receivedPassword, this.password);
3 }
```

5.2.7.2 AWS S3 permissions

To configure the AWS access, IAM was used to configure the user and associated policy. Figure 5.18 shows the policy in use, **dmpr-bucket-policy**, which allows users associated with it to perform only three actions:

- **GetObject**
- **DeleteObject**
- **PutObject**

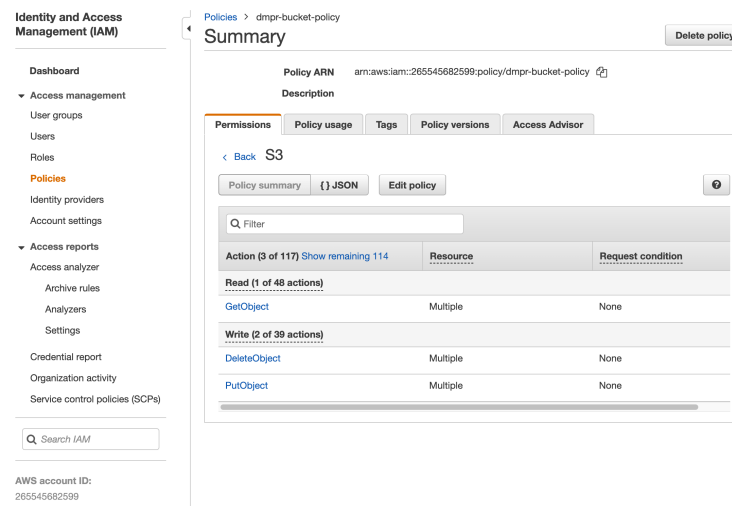


Figure 5.18: AWS: Policy summary

Figure 5.19 exhibits the user, **dmpr-r-w**, that is used to access the S3 service, and the respective associated policy.

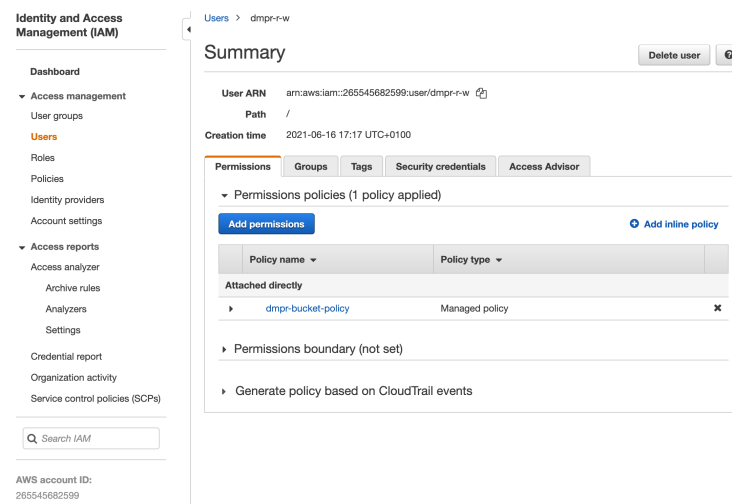


Figure 5.19: AWS: User summary

By ensuring the user can only perform a fraction of the available actions, to be specific, 3 out of 117, even if the credentials are intercepted, the damages are evidently less catastrophic than if the user had full access to all actions.

5.2.7.3 AWS S3 encryption with AES-256

AWS S3 was configured to use SSE-S3, which uses AES-256, with a key size of 256 bits, that encrypts each object with an unique key. As an additional safety measure, the key is encrypted with a master key that is rotated regularly. The encryption and decryption of the hosted files is not a concern since it is autonomously handled by AWS S3.

5.2.7.4 HTTPS Communication

The implementation of HTTPS communication was effortless since all services used are already SSL certified. This presents a clear advantage of implementing a back-end based on micro-services that are either SaaS or deployed with PaaS.

5.2.7.5 JWT

JWT is used for authentication purposes by allowing the user to access protected API routes. The first thing that was done was generating the JWT secret. To do that, the **Crypto** library was used to generate a random string of 35 bytes. The JWT secret creation was done via the command line as shown in listing 5.14.

Listing 5.14: API Gateway: JWT secret generation

```
1 require('crypto')
2 crypto.randomBytes(35).toString('hex')
```

Then, the JWT secret, as well as the expiration time, were defined as part of the server's configuration, and placed in the *config.env* file. An excerpt of the *config.env* file is presented in listing 5.15.

Listing 5.15: API Gateway's environmental variables: JWT

```
1 # JSONWEBTOKEN CONFIGURATION PARAMETERS
2 JWT_SECRET = *****
3 JWT_EXPIRE = 10min
```

When a user logs in, the API Gateway generates a JWT for the session. JWT was implemented with the default option of the **Jsonwebtoken** library, HMAC-SHA256. HMAC is a common type of MAC that uses both a cryptographic hash function and a secret cryptographic key to calculate a hash specific for each message. In this case, the cryptographic hash function used is the SHA-256, and the cryptographic key used is the generated secret. Listing 5.16 shows the token generation implementation.

Listing 5.16: API Gateway: JWT signing

```
1 UserSchema.methods.getSignedToken = function() {
2   return jwt.sign(
3     { id: this._id },
4     process.env.JWT_SECRET,
5     { expiresIn: process.env.JWT_EXPIRE });
6 }
```

From that moment on, when the user tries to access a protected route, in addition to the request data, the header will be authenticated with the generated JWT. If the JWT is valid, the request will go through, otherwise the user will receive as response *401 Unauthorized*.

This is possible since HTTP provides a framework for controlling access to protected resources. In this case, Bearer, one of many HTTP authentication schemes, authentication was used. Bearer token authentication is done by sending the security token with every HTTP request that is made to the server. The validity verification is shown in listing 5.17.

Listing 5.17: API Gateway: JWT authentication's middleware

```

1 export const protect = async (req, res, next) => {
2   let token;
3
4   if(req.headers.authorization && ...
     req.headers.authorization.startsWith("Bearer")) {
5     // NOTE(araragao): req.headers.authorization = Bearer **token**
6     // NOTE(araragao):
7     // NOTE(araragao): Hence why it is needed to split by " " and truncate ...
8     // the token as the second item on the array, i.e., the actual token
9     token = req.headers.authorization.split(" ")[1];
10
11   }
12
13   if(!token) {
14     // NOTE(araragao): 401 UNAUTHORIZED - Authorization missing from the ...
15     // request
16     return next(new ErrorResponse("Not authorized", 401));
17   }
18
19   try {
20     const tokenDecoded = jwt.verify(token, process.env.JWT_SECRET);
21
22     const user = await User.findById(tokenDecoded.id);
23
24     if(!user) {
25       // NOTE(araragao): 404 NOT FOUND - Targeted resource does not exist
26       return next(new ErrorResponse("No user found with the associated ...
27         token", 404));
28     }
29
30     req.user = user;
31     next();
32   } catch(error) {
33     // NOTE(araragao): 401 UNAUTHORIZED - Authorization missing from the ...
34     // request
35     return next(new ErrorResponse("Not authorized to access this route", 401))
36   }
37 }

```

In order to actually protect the routes with the authentication's middleware shown in 5.17, the middleware should be added to the route logic before the controller's logic, as shown in 5.18. This way, the JWT authentication's middleware, imported as `protect`, is invoked before the controller `fetchAllReceipts`.

Listing 5.18: API Gateway: JWT authentication's middleware usage

```

1 import express from 'express';
2 import { protect } from '../middleware/auth.js';
3 import {fetchAllReceipts} from '../controllers/receipts.js';
4
5 const router = express.Router();
6
7 router.route("/receipts").get(protect, fetchAllReceipts);
8
9 export default router;

```

5.3 System operation & execution

5.3.1 Receipt stages

For a good understanding of this section, API Gateway's function *changeReceiptStatus* and the OCR service's code should be detailed. *changeReceiptStatus*'s logic is implemented with a switch...case structure together with the necessary database service requests to update the receipt fields. As the implementation complexity of the *changeReceiptStatus* function is very low, the code is not shown. The OCR service code is reviewed in depth in the previous subsection 5.2.4.

Although the receipt stages are not a proper state machine, since the OCR service is responsible for the first transition and the others are handled by a request sent to the API Gateway that calls the function *changeReceiptStatus*, it was modelled as one for a better understanding of the system stages. As a way of improving the system, the implementation of a dedicated receipt status service is proposed in subsection 6.4.5.

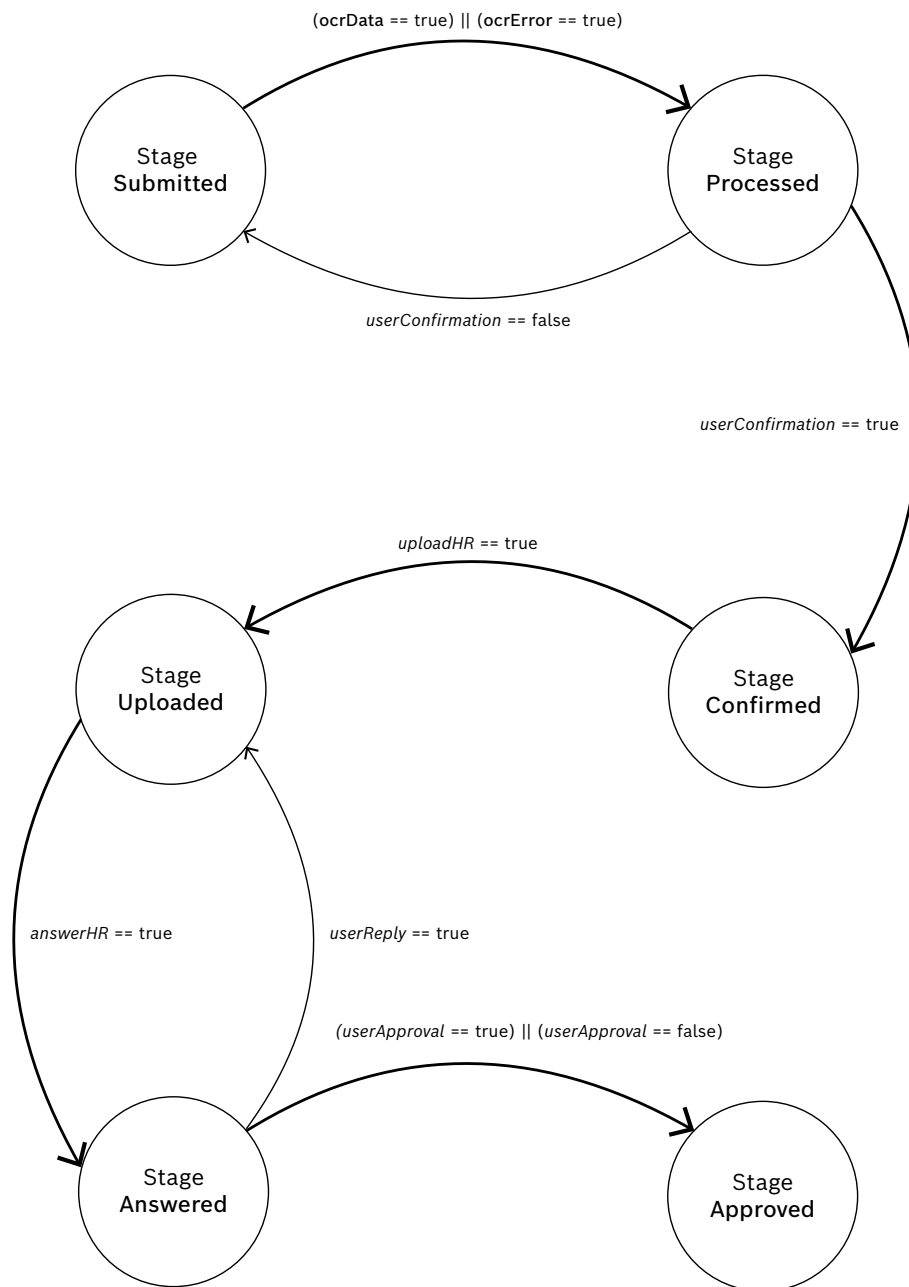


Figure 5.20: Receipt stages state-machine

Figure 5.20 shows the implemented state-machine. It is relevant to remark that the state changes

can be due to a change in the database values, represented in figure 5.20 by transactions with the variable's text in **bold**, or due to an explicit call of the implemented API Gateway request function *changeReceiptStatus* with a body that matches the transactions variable's text name, represented in the same figure by transactions with the variable's text in *italic*.

Figure 5.21 exhibits the different possible status shown in the **Receipt Overview** screen. It is worth mentioning that, although it shows a possible status of **Default** and **Waiting connection**, those are only relevant for the mobile application, and not valid status field values in the database. The **Default** status is a visual indicator that shows the user that he doesn't have access to the stage related actions. This happens because the stage is ahead of the current stage for the receipt. The **Waiting connection** status is a temporary status that implies that, when the receipt was created, the mobile application tried to send it to the API Gateway but there was no connection. Therefore, a copy of the receipt is stored locally and will be sent to the API Gateway as soon as there is internet connection.

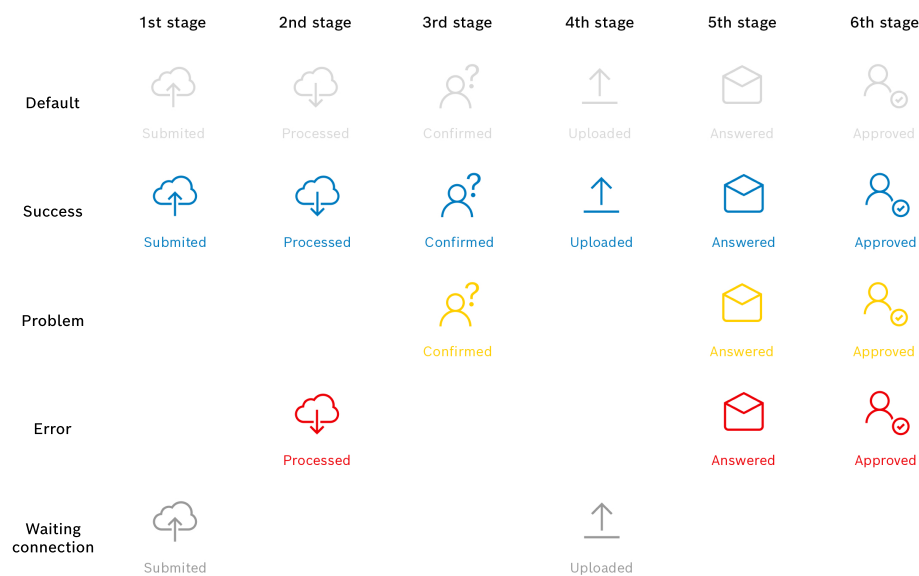


Figure 5.21: Displayed status values for each receipt stage, featured on the screen "Receipt Overview"

The following subsections explain in detail each stage shown in figure 5.20.

5.3.1.1 Stage: Submitted

This is the first stage of the state-machine, which is represented by the field **stage** having the value **submitted**. This stage represents that the receipt was successfully added to the database and can now be processed by the OCR service, therefore the **status** value of **success**.

It should be noted that, as stated in 5.2.1, the OCR service is triggered as soon as there is a change in the database, by listening for the database's change stream.

5.3.1.2 Stage: Processed

In this state, the parameter **stage** has the value **processed**. This state is reached after OCR service processed the receipt. It is the OCR service that is responsible for this transition, thus the OCR service updates the particular database entry.

The OCR processing can be successful or not, hence why there are two possibilities for transitioning from the first stage to the second, as seen in 5.20.

5.3.1.2.1 Successful processing

If the receipt processing was successful, the icon is displayed in blue in the receipt overview screen of the mobile application, a consequence of the field **status** having the value of **success**. In addition to the change of the receipt's **state**, the **ocrData** field should now have the value **true**, against its previous value of **false**, and the **ocrDatakey** field the key to access the storage and retrieve the OCR processing data.

After the receipt being successfully processed, the user has to take action regarding the OCR result:

- If the user completely agrees with the result, a state change request should be sent to the API Gateway with the parameter *userConfirmation*, with the assigned value of *true* to it. This would prompt a change to the next **stage**, **confirmed**, and set the field **status** to **success**;
- If the user partially agrees with the result, the receipt can go forward in the receipt processing stages by sending a request to the API Gateway with the parameter *userConfirmation*, with the assigned value of *true* to it, and also a second parameter **userConfirmationMessageContent** with the value *true* to represent that there is a message, **userConfirmationMessageContent**, the user wants to add to the receipt about the receipt's processing. This action results in the setting of the field **status** to **problem**;
- If the user doesn't agree with the result, there is the possibility of completely discarding the previous OCR result by either submitting a replacement picture for the same receipt entry, which results in the change of **pictureKey**, or by forcing the OCR result to be computed again for the same picture, by making a request to the API Gateway with the parameter *userConfirmation*, with the assigned value of *false* to it.

5.3.1.2.2 Failed processing

If the receipt processing failed, the icon is displayed in red color in the receipt overview screen of the mobile application, a result of the field **status** having the value of **error**. This means that, although the **state** has the value **processed**, the **ocrError** field should now have the value **true**, against its previous value of **false**.

To address the failure, the user can take the same action as the one lastly described in case of successful processing. The difference is not from a users' perspective but rather from the API Gateway, since it is required to change the value of **ocrError** from **true** to **false**. Otherwise, the stage cannot go back to **submitted**, since that in the **submitted stage** the OCR processing has yet to happen.

5.3.1.3 Stage: Confirmed

In this state, the parameter **stage** has the value **confirmed**. This stage is reached after the user has confirmed the results of the OCR service regarding the analysis and data extraction of the receipt.

5.3.1.3.1 Successful confirmation

If the receipt was successfully confirmed, the icon is displayed in blue in the receipt overview screen of the mobile application, a consequence of the field **status** having the value of **success**.

5.3.1.3.2 Problematic confirmation

Another possibility is that the receipt was confirmed even though the user found a problem in the processing. In this case, the database's receipt field **userConfirmationMessageContent** should contain the problem's description. In the receipt overview screen of the mobile application, the icon is displayed in yellow, a consequence of the field **status** having the value of **problem**.

5.3.1.3.3 Further actions

The only action that the user can take is to upload the receipt to the HR. In order to do that, a request should be sent to the API Gateway with the field *uploadHR* set to *true*. Although the Gateway API is not configured to actually upload the receipt, and its attributes, to Bosch servers', it would be a matter of adding Bosch servers' as a service to the architecture described in the subsection 5.2.1, and use the appropriate upload function.

5.3.1.4 Stage: Uploaded

In this state, the parameter **stage** has the value **uploaded**.

Given the nature of this stage, the only color option for the icon in the receipt overview screen of the mobile application is blue, a consequence of the field **status** having the value of **success**.

HR intervention is required for the receipt to advance to the next processing stage. Consequently, the receipt stage will *freeze* in this stage until a response from HR is received.

The stage change occurs by the explicit call of the implemented API Gateway request function *changeReceiptStatus* with the following body variables:

- *answerHR*, which should have the value *true*;
- **status**, so that the receipt status field can be updated;
- **hrMessage** with the value *true* if the updated **status** is either **problem** or **error**;
- **hrMessageContent** if **hrMessage** has the value *true*.

5.3.1.5 Stage: Answered

In this state, the parameter **stage** has the value **answered**. This state is reached after the HR analyzed the receipt submission.

5.3.1.5.1 Successful answer

This status is a consequence of the successful approval of the receipt from the HR. Since **status** has the value of **success**, the icon is displayed in blue. Considering that there is only one stage left that corresponds to the user's approval, to further advance in the receipt processing stages, a request with the following body variables should be made to the API Gateway's corresponding *changeReceiptStatus* function:

- *userApproval*, which should have the value; *true* if the user approves, or *false* if he doesn't
- **status**, so that the receipt status field can be updated;
- **userApprovalMessage** with the value *true* if the updated **status** has the value **problem**;
- **userApprovalMessageContent** if **userApprovalMessage** has the value *true*.

5.3.1.5.2 Problematic answer

This status is a consequence of the partial approval of the receipt from the HR. Since **status** has the value of **problem**, the icon is displayed in yellow. The receipt field **hrMessageContent** should contain the problem description so the user can reply to the HR or approve the receipt with the stated problem.

The user has two options:

- Approve the receipt knowing that a problem was found, by sending a request to the API Gateway as stated above;
- Reply to the HR, and wait for another message from them, as detailed below.

In order to reply to the HR, a request with the following body variables should be made to the API Gateway's corresponding *changeReceiptStatus* function:

- *userReply* with the value *true*;
- **userReplyHRMessageContent** with the desired reply message.

In this case, the *changeReceiptStatus* function would change the **status** back to **uploaded** and reset the **hrMessage** and **hrMessageContent** fields.

5.3.1.5.3 Failed answer

This status is a consequence of the failed approval of the receipt from the HR. Since **status** has the value of **error**, the icon is displayed in red. In this specific case, there is no going back in the receipt processing stage. The receipt is *frozen* indefinitely and the user is prompt to contact the HR.

5.3.1.6 Stage: Approved

In this state, the parameter **stage** has the value **approved**. This state represents the end of the receipt processing cycle.

5.3.1.6.1 Successful approval

If the receipt was successfully approved, the respective icon is displayed in blue in the receipt overview screen of the mobile application, a consequence of the field **status** having the value of **success**.

5.3.1.6.2 Problematic approval

Another possibility is that the receipt was approved, although there is still an issue to be discussed. In this case, the database's receipt field **userApprovalMessageContent** should contain the problem's description. The icon is displayed in yellow in the receipt overview screen of the mobile application, a consequence of the field **status** having the value of **problem**.

5.3.1.6.3 Failed approval

Lastly, the receipt can be not approved by the user. In this case, the database's receipt field **userApprovalMessageContent** should contain the reason why. The icon is displayed in red in the receipt overview screen of the mobile application, a consequence of the field **status** having the value of **error**.

5.4 Tests

Functional testing To ensure that the implemented features work correctly, functional testing was carried out for both the mobile application and the back-end API.

Performance To ensure that the OCR processing results are worth a change of the expenses reimbursement protocol, performance testing was carried out in two domains: **accuracy and precision** and **processing delay**.

5.4.1 Functional

5.4.1.1 Mobile application

Functional testing is usually carried out in four different sequential phases:

1. **Unit testing**, carried out for each feature or function, usually in a simulated environment;
2. **Integration testing**, conducted for more than one feature at the same time;
3. **System testing**, performed for the entire system;
4. **Acceptance testing**, to verify if the implemented features correspond to the requirements.

Beyond question, all tests took place during the development cycle thanks to the convenience of Expo framework. Besides the bugs reported in subsection 4.3.3, every other bug that occurred is solved.

5.4.1.2 Back-end API

The back-end API was tested independently of the mobile application. As means to achieve comprehensive testing, Postman was used. Postman is a collaboration platform for API development that allows sending requests and review the responses of REST, SOAP, and GraphQL protocols. Furthermore, it supports the use of variables, which was particularly useful for storing the JWT token. Figure 5.22 shows the verified requests.

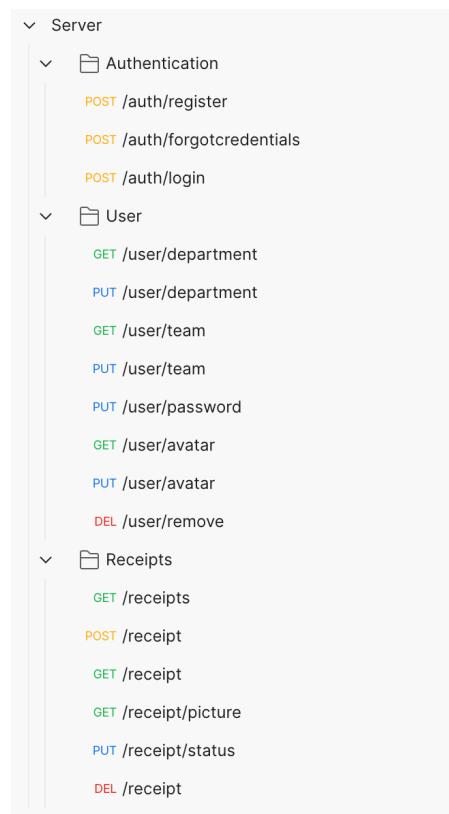


Figure 5.22: API Gateway testing: Postman testing suite

Figure 5.23 shows an example of the correct behavior of the API: Invalid credentials lead to *404 Not found*, as expected.

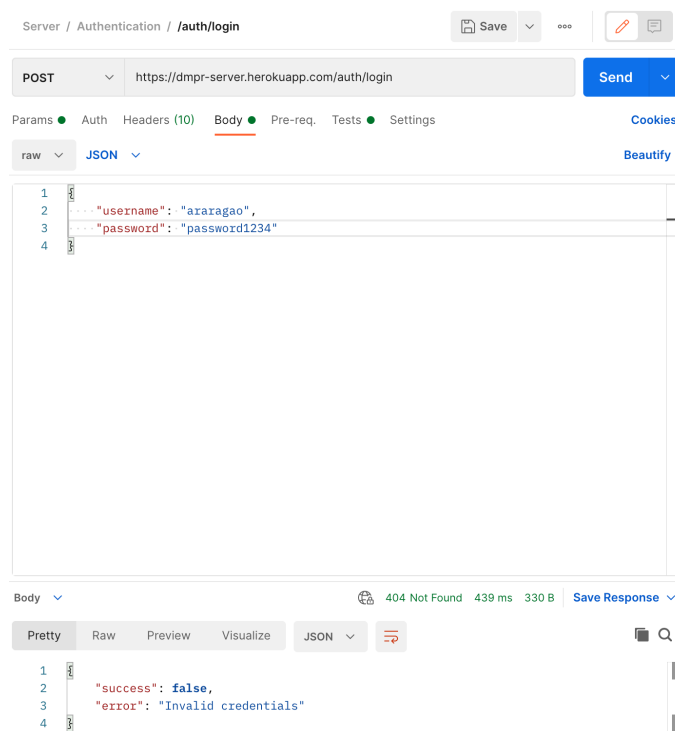


Figure 5.23: API Gateway testing: Login error

Figure 5.24 shows that when the credentials are valid, login is successful and a token is sent to the client. This token is saved as a Postman variable.

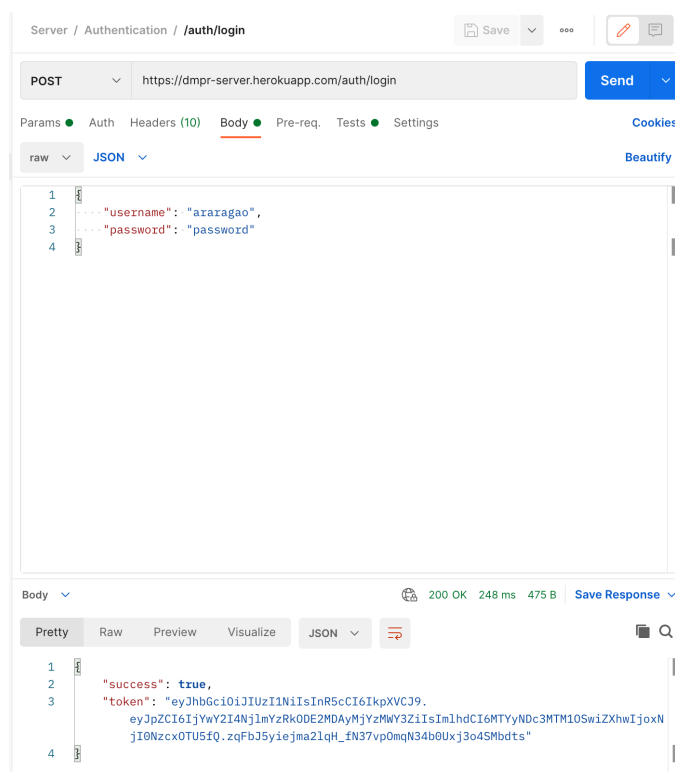


Figure 5.24: API Gateway testing: Login success

Figure 5.25 shows the correct behavior of the function **fetchAllReceipts** that retrieves all receipts associated with the corresponding account. The correct behaviour is only achieved given the use of header authentication, since this route is protected and requires token authentication.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** https://dmp-server.herokuapp.com/receipts
- Status:** 200 OK, 121 ms, 1.08 KB
- Headers:**

KEY	VALUE	DES	Bulk Edit	Presets
Authorization	Bearer {{token}}			
Key	Value	Description		
- Body (JSON):**

```

1  {
2    "success": true,
3    "data": [
4      {
5        "createdAt": "2021-06-19T15:59:34.761Z",
6        "_id": "60ce1470947b800022b33d69",
7        "title": "Jal Gaheer 6",
8        "trip": "André goes on tour",
9        "numberOfPeople": "3",
10       "language": "eng",
11       "comment": "Some beverages are not alcoholic",
12       "picture": true,
13       "pictureKey": "araragao&2021-06-19T15:59:44.557Z",
14       "ocrData": false,
15       "ocrDataKey": "to-be-added",
16       "status": "submitted",
17       "hrResponse": "to-be-added",
18       "associatedAccount": "araragao",
19       "__v": 0
20     },
21     {
22       "createdAt": "2021-06-23T14:01:20.490Z",
23       "_id": "60d33fc5cc3ef90022b58762",
24       "title": "Jal Gaheer 5",
25       "trip": "André goes on tour",
26       "numberOfPeople": "3",
27       "language": "eng",
28       "comment": "Some beverages are not alcoholic",
29       "picture": true,
30       "pictureKey": "araragao&2021-06-23T14:05:57.734Z",
31       "ocrData": false,
32       "ocrDataKey": "to-be-added",
33       "status": "submitted",
34       "hrResponse": "to-be-added",
35       "associatedAccount": "araragao",
36       "__v": 0
37     }
38   ]
39 }

```

Figure 5.25: API Gateway testing: fetchAllReceipts



Figure 5.30: OCR service testing: Hard (1)



Figure 5.31: OCR service testing: Hard (2)

5.4.2.1.1 Processing delay

First and foremost, it should be mentioned that the processing of a receipt with one or multiple workers results in the same output. The only difference is the processing time.

Listing 5.19 shows the code that performed the time measurement. The code works as expected since the function `worker.recognize()` is implemented as an `await` expression that causes the `async` function execution to pause until the promise is settled. Therefore, the service stops until the `worker.recognize()` returns.

Listing 5.19: OCR service testing: time measurement

```

1 const beginningTime = Date.now();
2 const { data: { text } } = await worker.recognize(Buffer.from(picture, 'base64'));
3 const endingTime = Date.now();
4 const processingTime = endingTime - beginningTime;
```

Table 5.1 shows the processing time, in milliseconds, of the OCR service, for both one and five workers.

Receipt	Processing time - single worker (ms)	Processing - five workers (ms)
Easy	7452	4781
Medium (1)	5944	3740
Medium (2)	7955	6395
Medium (3)	8919	8725
Hard (1)	6305	7644
Hard (2)	11583	7940

Table 5.1: OCR service: Processing time of one and five workers

From table 5.1, an intuitive generalization can be confirmed: more workers result in less processing time. However, for receipt *Medium(3)*, this generalization doesn't apply.

In addition, taking processing time as a measure to determine the complexity of the receipt, receipt *Medium(1)* was the fastest one to be processed so it should be labeled as **easy**. This is counter intuitive since there is an apparent need for more pre-processing of the receipt *Medium(1)*, when compared to the receipt *Easy*, as it is not aligned and the text is not as clearly distinct from the receipt's background as it is in receipt *Easy*.

5.4.2.1.2 Percentage of error

Receipt	Total characters	Successful extracted characters	% of error
Easy	234	114	51.29
Medium (1)	413	5	98.79
Medium (2)	227	224	1.32
Medium (3)	291	266	8.6
Hard (1)	166	153	7.83
Hard (2)	260	74	71.54

Table 5.2: OCR service: Percentage of error

Table 5.2 shows the percentage of error for each testing receipt. The most relevant find was that the image quality is by far the most relevant aspect for the correct data extraction, as described by Tafti et al. [8]. Consequently, as receipt *Medium (1)* and *Hard (2)* have inferior quality than the other receipts in analysis, the percentage of error is extremely high for those receipts. For the same reason, but in the opposite spectrum, the processing of the receipts *Medium (2)*, *Medium (3)* and *Hard (1)* resulted in great outputs.

Listing 5.20 shows the result of the OCR processing for receipt *Medium (3)* as means to illustrate the quality of the output given a quality input.

Listing 5.20: OCR service testing: Medium (3) processing result

```
PizzaExpress Biggleswade
36 High Street
Tel 01767 313 232
Ti 3
Abby 20 05t 2019 13:30
Table: 15 c: 2 Aco No: 2522
1 Mix Fruit Cider = 5.95
1 Cappuccino = 2s
1 calzone Nduja = 14.30
1 PEO Courses £19.95 = fags
1 6B & Wozzarella
1 Pollo Pesto
1 Choo Fudge Cake1
38 Courses £13.05
Product Group Summary,
Food 2.25
Drinks 8.50
Total £36.75
You have saved: £6.20
```

6. Discussion and conclusion

6.1 Summary

Digital Management of the Payment Receipts aimed to develop a system based on specific requirements defined by the proponent company, Bosch Car Multimedia, Braga. The requirement list can be found in section 1.3. The successful outcome of this thesis, measured by the considerable number of accomplished requirements, paves the way to the implementation of a solution that undoubtedly adds value to the company.

6.2 Learning outcomes

The technical expertise necessary for the development of this thesis was entirely foreign to the student, considering that the student's technical background, acquired from both BSc and MSc studies, revolves around microelectronics and embedded systems. As a result, the learning payoff was tremendous.

Firstly, the need to adopt appropriate design methodologies and techniques, validated by the company's field experts, resulted in unforeseen **design skills acquirement**, both conceptually and technically, namely the usage of Sketch, a UI & UX design software.

At the same time, given that it was an unexpected learning outcome, it presented another challenge that, ultimately, culminated in the defiance of the original project planning. Therefore, **time management** was definitely one of the hardest things to properly accomplish due to the lack of provision for setbacks, such as the necessary time to design the UI & UX in accordance to Bosch's design principles, which, in the end, resulted in a reduced time frame for previously planned tasks. Another important factor that should be mentioned is that the required technical skills to successfully develop the project were severely underestimated, on account of the lack of expertise in the required domains. This presented another challenge in regards to the time management, since more time than expected was needed to learn the necessary skills, and only then these could be applied to the development of the project.

In terms of planned technical skills, the student is now familiarized with the **development of mobile applications**, using **React Native**, and, consequently, **JavaScript**, and the **development and deployment of back-end systems**. On the subject of back-end systems, multiple aspects were addressed: **back-end architecture**, in particular the **micro-services architecture**; **cloud services**, specially the deployment of services using PaaS such as **Heroku**, the use of IaaS such as **AWS S3** and the use of SaaS such as the **Twilio Sendgrid** email service; **development and testing of an API**.

On a less impactful level, **security methodologies** were also researched and implemented, although the underlying concepts are not fully understood, given the exceptional complexity of the topic.

6.3 Objective fulfillment

The developed system fulfills more than 85% of the requirements. Therefore, the project can be considered successful, particularly when taking into account the project's scope and the development timeline. Requirement **Requirement 12** was not fulfilled, as the algorithm in use presents technical difficulties for low quality images. Improvements regarding the refinement of the OCR algorithm are presented in 6.4.6. At the same time, requirement **Requirement 13** was not accomplished since REGEX was not implemented due to time constraints.

The optional requirements **Requirement 16** and **Requirement 17** are partially implemented as, during the development, considerations were taken into account regarding both the accessibility, when designing the UI & UX, and the security, when developing both the mobile application and the back-end, although no further verification was done to confirm if the measures taken to fulfill both requirements were enough to satisfy them. The optional requirement **Requirement 18** was not implemented due to the lack of interest from the company's side.

6.4 Further system improvements

6.4.1 Mobile application: Implementation of *pure* React Native

If on one hand, by using the Expo framework, the development cycle is considerably accelerated, on the other, the overhead it introduces is a problem to consider. Having in mind the goal of achieving the best performance and the smallest application size possible, withdrawing the use of the Expo framework is recommended.

It should be noted that Expo libraries can still be used, even though the application is not deployed with the Expo framework.

6.4.2 Mobile application & back-end: Administration page

In order to not depend on the micro-services' available interface, it is advised that an administration page is developed. This would benefit the management of the database, services, and other features.

6.4.3 Mobile application & back-end: Lessen the use of third party libraries

Security measures should be taken into high consideration, given that the system deals with highly sensitive data that shouldn't, on any occasion, be vulnerable. The use of third party libraries defies the system's security since the code can be, at any time, subject to an exploit. It is advised that the number of third party libraries used is reduced to a minimum and, ideally, to none.

6.4.4 Back-end: BaaS implementation

For the sake of efficiently deploying a robust back-end system, BaaS platforms could be used. At first sight, this improvement might seem contradictory to the previously suggested indication of reducing the number of third party libraries. However, companies that provide BaaS platforms, like Firebase, are subjected to excruciating control, as their revenue depends on the quality of the services delivered. This results in transparent and reliable code that promotes a quick way of developing a complete back-end system. Furthermore, regular security updates of the BaaS platforms are extremely beneficial since it relieves the developers of the continuous support of the system.

6.4.5 Back-end: Receipt status service

As stated in section 5.3.1, the OCR service is responsible for the first transition of the receipt's state machine, presented in 5.20, and the others are handled by a request sent to the API Gateway that calls the function *changeReceiptStatus*. This implementation could be more in line with a micro-services architecture, by developing a service that handles the receipt's status. At the same time, this recommendation would result in a easier system management.

6.4.6 OCR service: Trained data & pre-processing

In order to achieve the best possible results, instead of using trained data based in unknown inputs, the data sets used should be based on receipts, as described by Chaudhuri et al. [5]. Moreover, trained data should be created for each supported language.

Another aspect worth mentioning is that pre-processing should take place before running the Tesseract OCR. Although the OCR algorithm performs pre-processing, by using libraries such as **otsu**, that implements a binarization algorithm which can be used to select the optimal threshold between foreground and background quantities based on finding the minimum point of cross variance, improved pre-processing results may be obtained, specially when taking into consideration that this methods are used specifically for photographed documents in odd light conditions [5]. This would result in superior accuracy and precision and in reduced use of memory space and computational effort, as mentioned by Chaudhuri et al. [5].

6.4.7 OCR service: REGEX

As mentioned in subsection 6.3, requirement **Requirement 13** was not fulfilled since REGEX was not implemented due to time constraints. It is advised that REGEX is implemented to accomplish the stated requirement.

6.4.8 OCR service: Translation feature

This feature is suggested having in mind that the standard language used at Bosch, by all associates, is English.

6.4.9 Further testing

The tests detailed in section 5.4 should be improved to guarantee the system is tested under different situations.

6.4.9.1 Mobile application testing

The mobile application was only tested as it was developed, without any kind of formal testing. It is advised that formal testing is performed and documented.

6.4.9.2 OCR Testing

The OCR service testing should be performed in more detail, taking into account different domains:

- The comparison between the Tesseract OCR algorithm with others, proprietary or not, for a matter of performance analysis;
- The Tesseract OCR should be tested with different languages in order to detect particular alphabet problems;
- The Tesseract OCR testing should take into account receipt pictures with different attributes, such as pictures taken from different angles, with multiple light exposure values, etc., and explicitly document the results.

Appendices

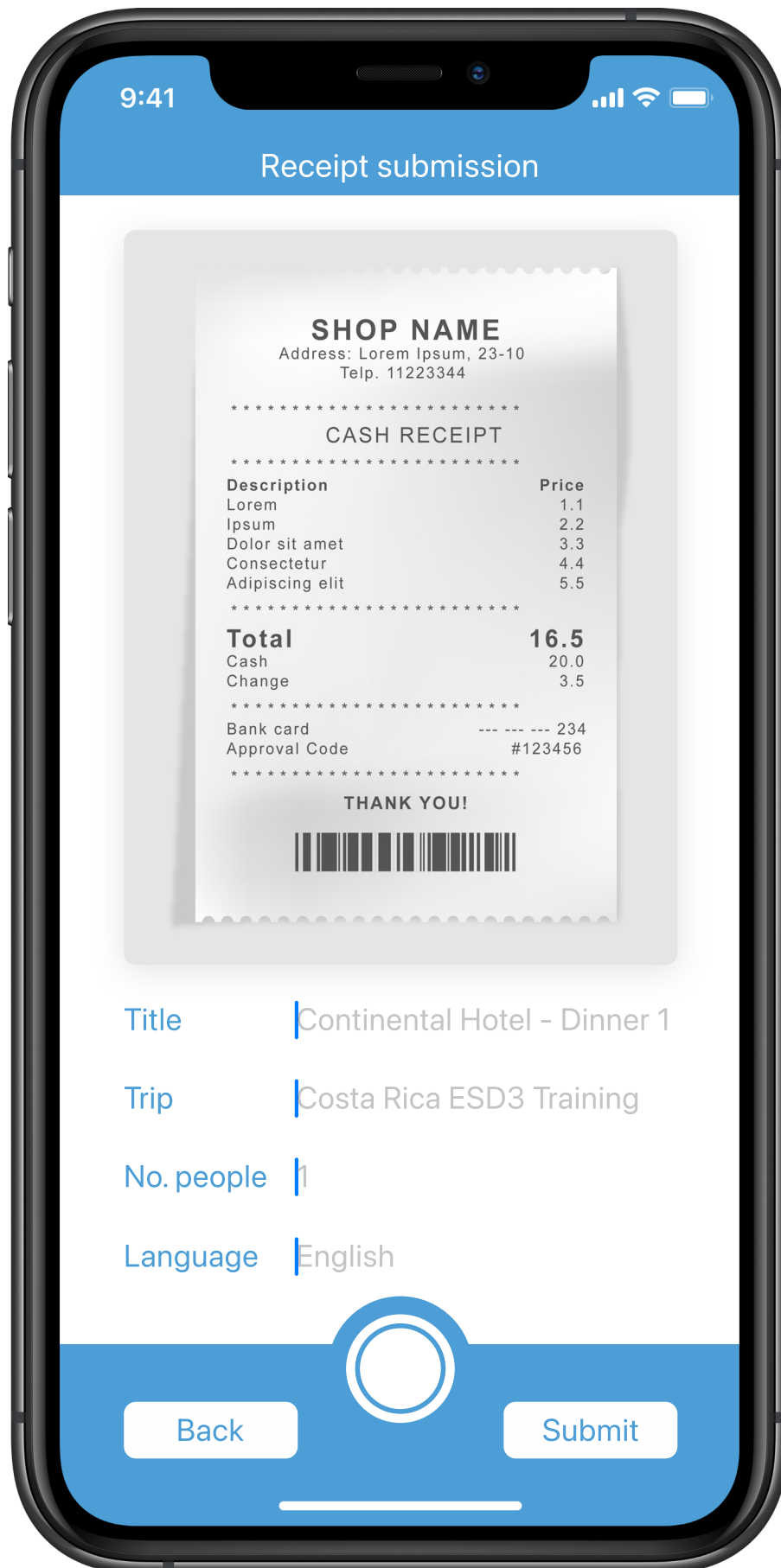
.1 UI Prototypes: Version 1

This section shows the first iteration of the development of the UI prototypes.

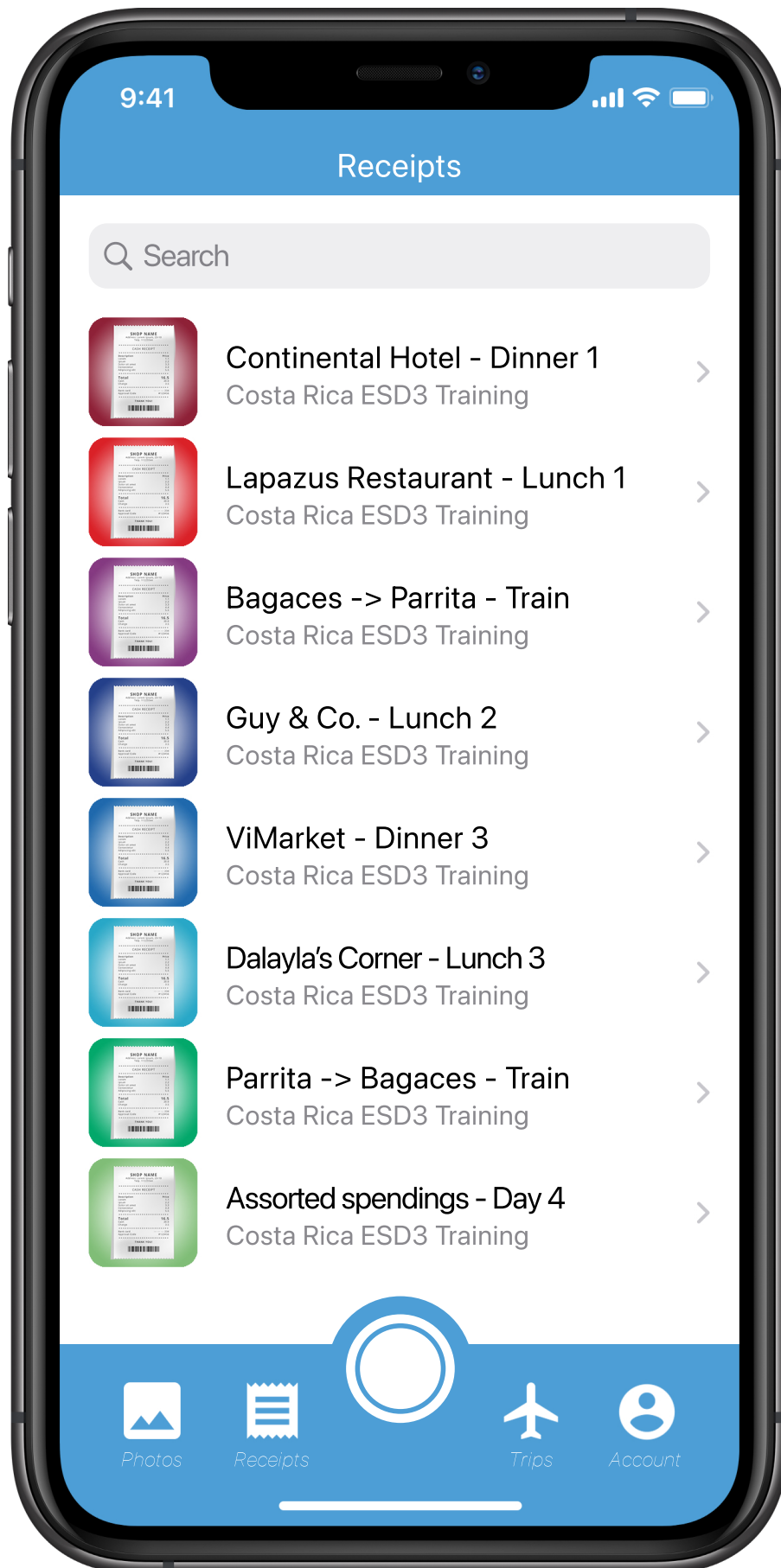
.1.1 UI Prototype - V1: Camera screen



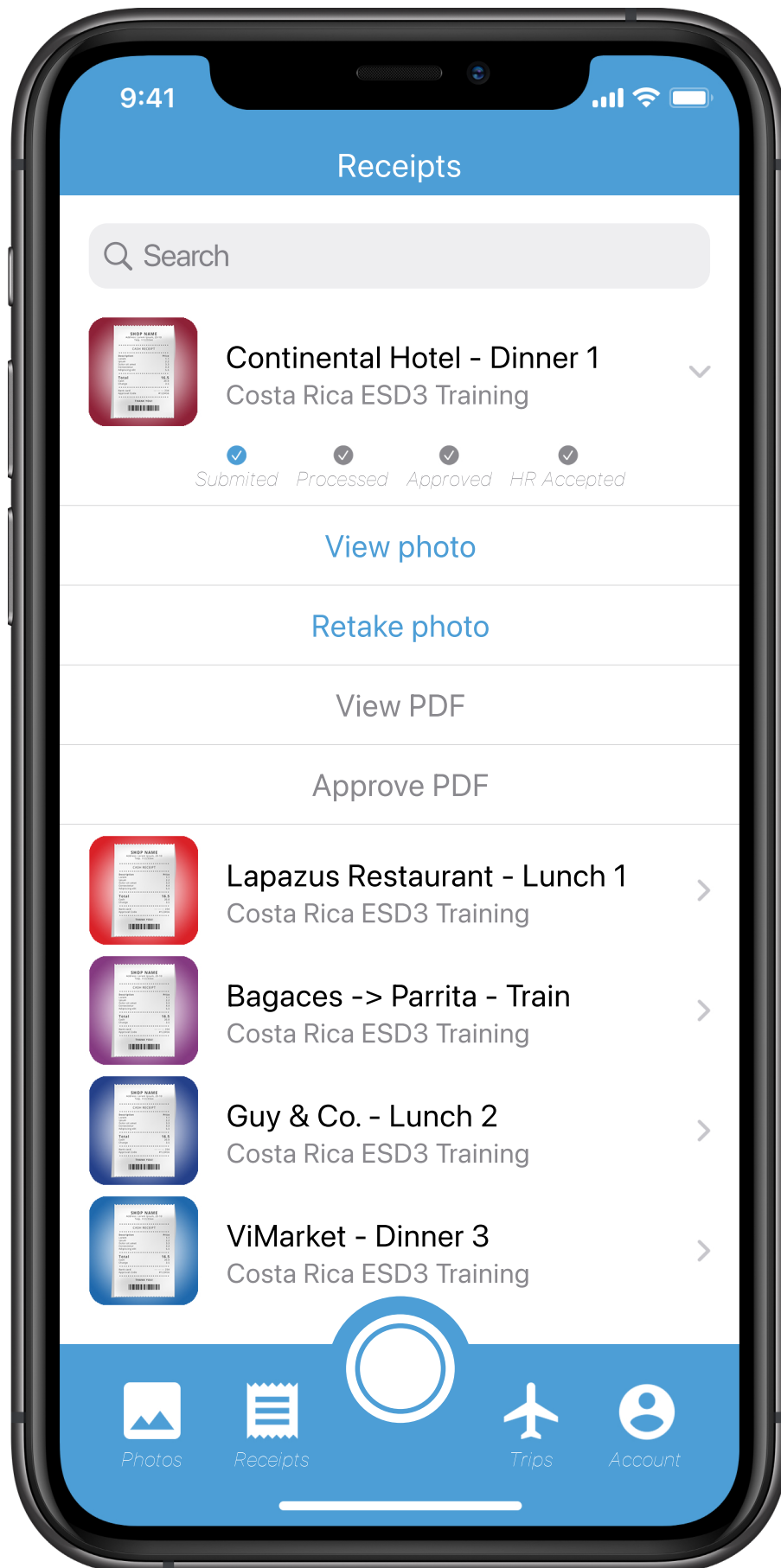
.1.2 UI Prototype - V1: Receipt submission screen



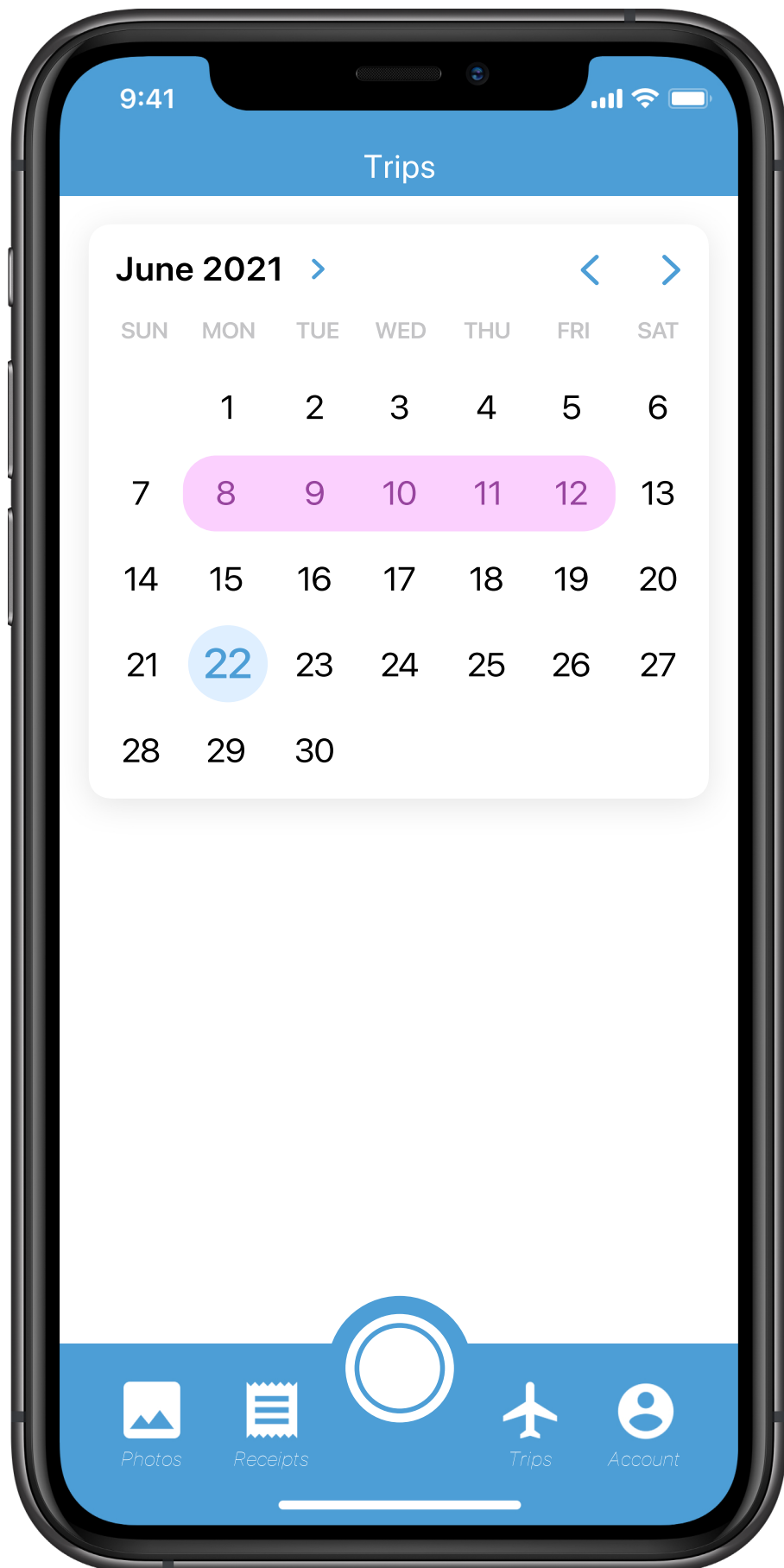
.1.3 UI Prototype - V1: Receipts screen



.1.4 UI Prototype - V1: Receipt options screen



.1.5 UI Prototype - V1: Trips screen



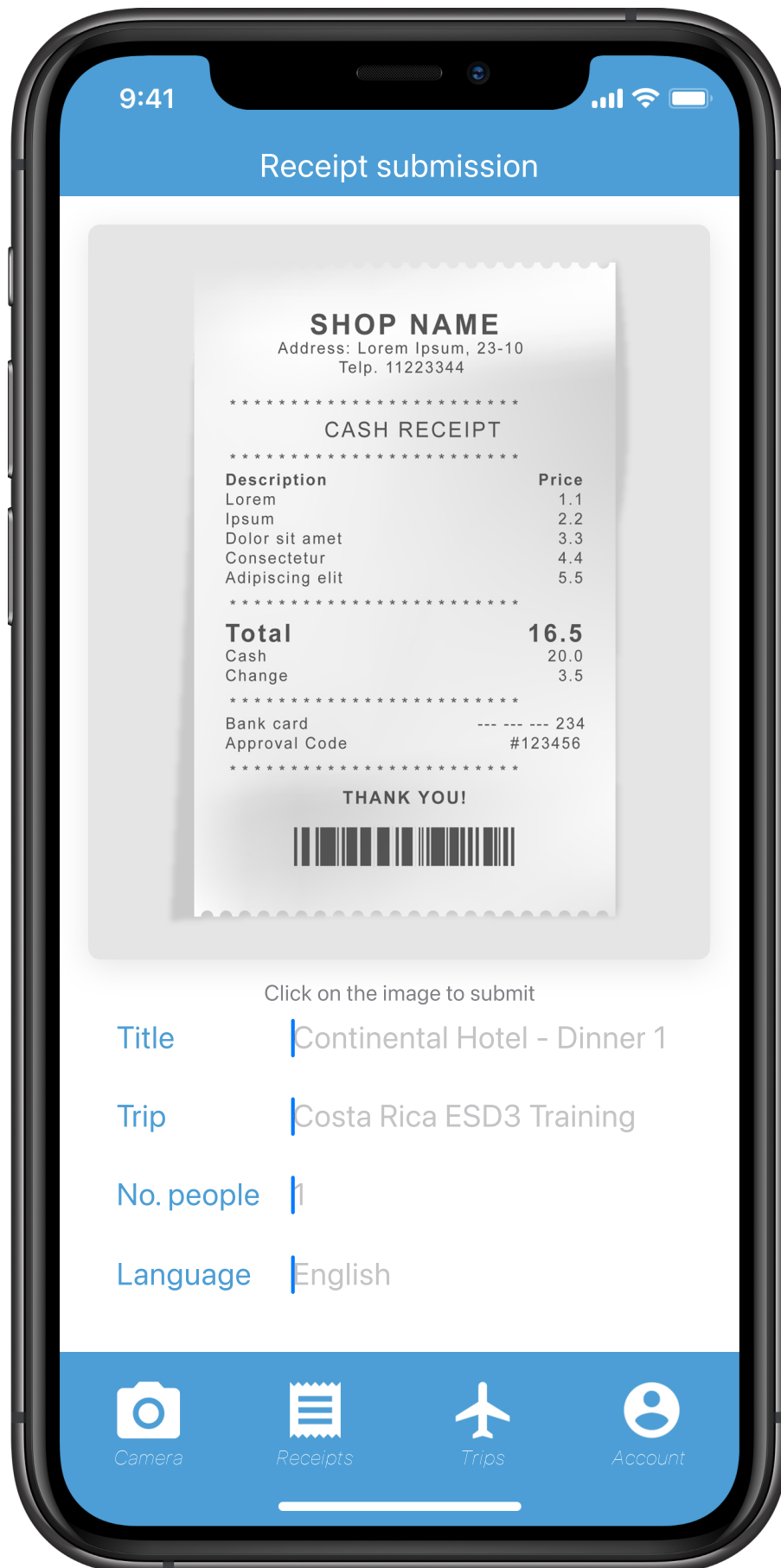
.2 UI Prototypes: Version 3

This section shows the third iteration of the development of the UI prototypes.

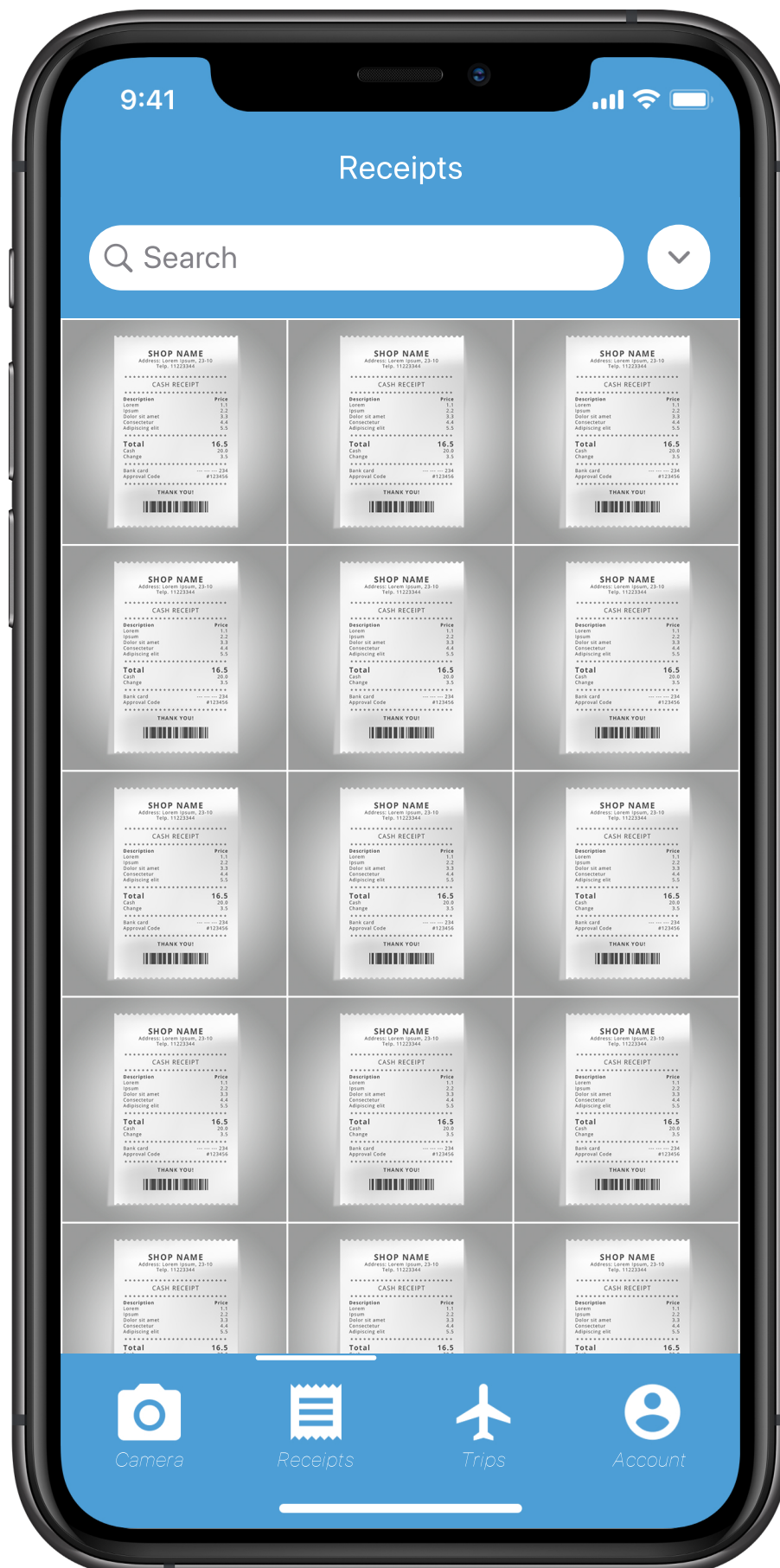
.2.1 UI Prototype - V3: Camera screen



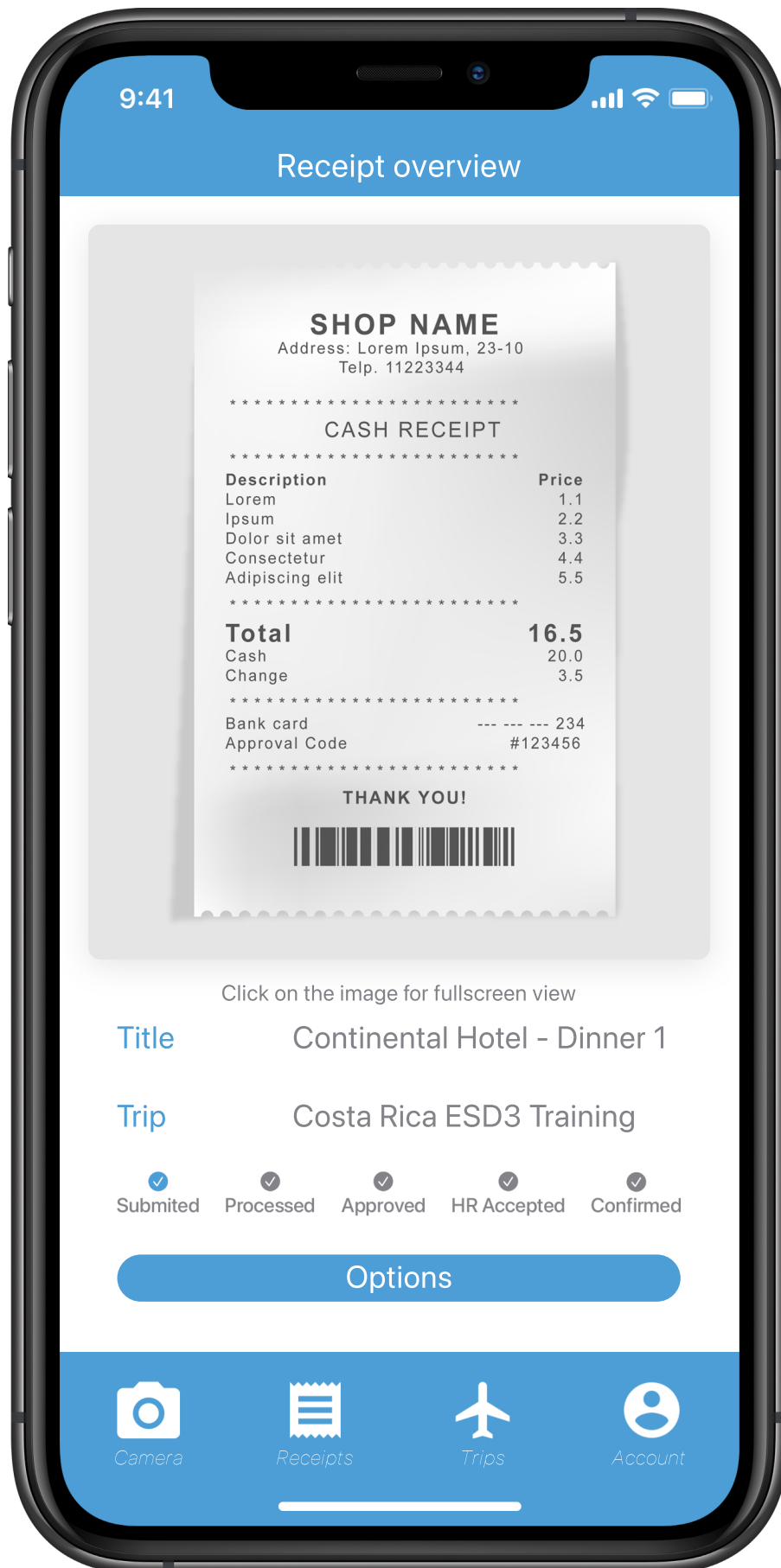
.2.2 UI Prototype - V3: Receipt submission screen



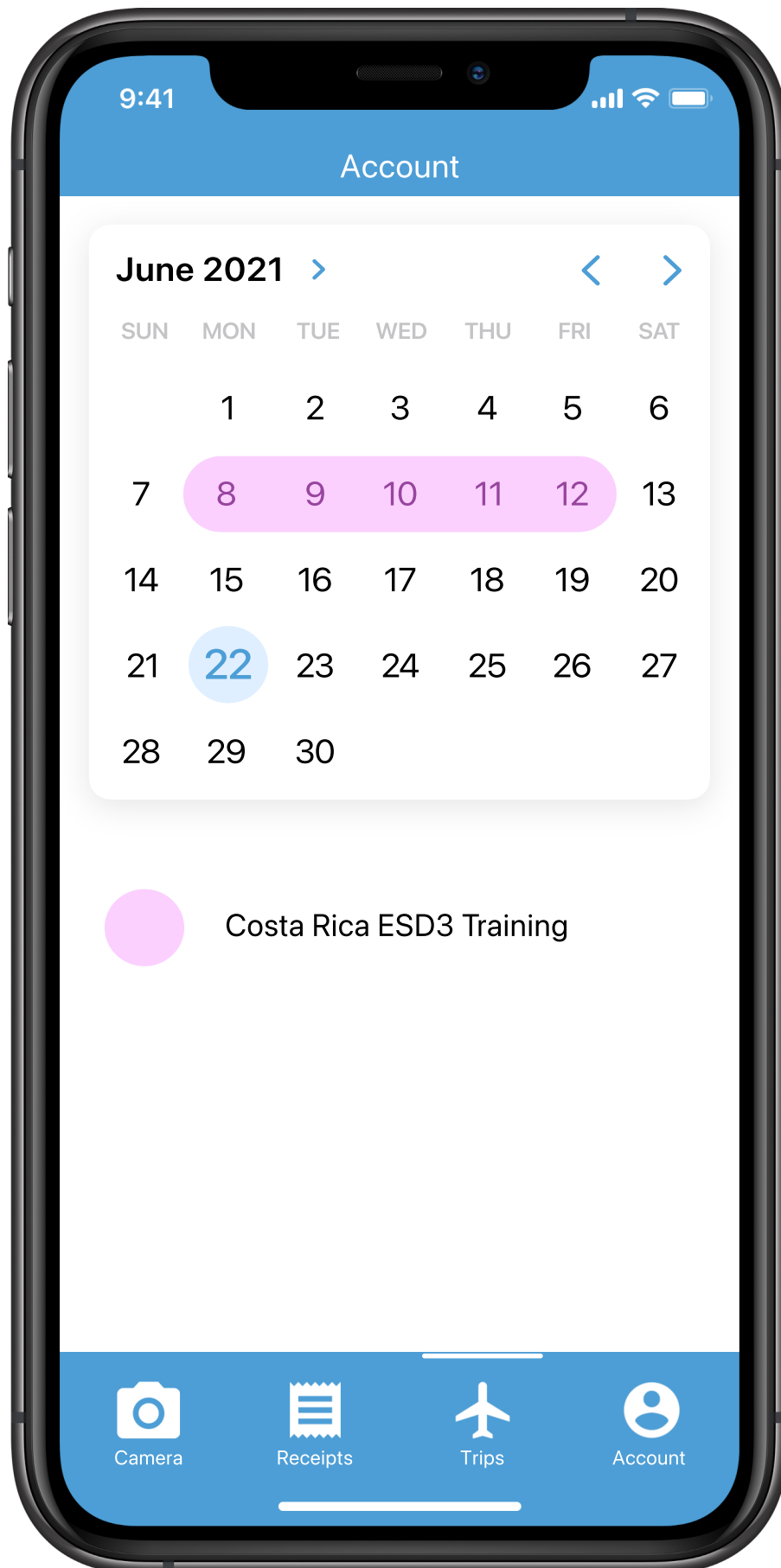
.2.3 UI Prototype - V3: Receipts screen



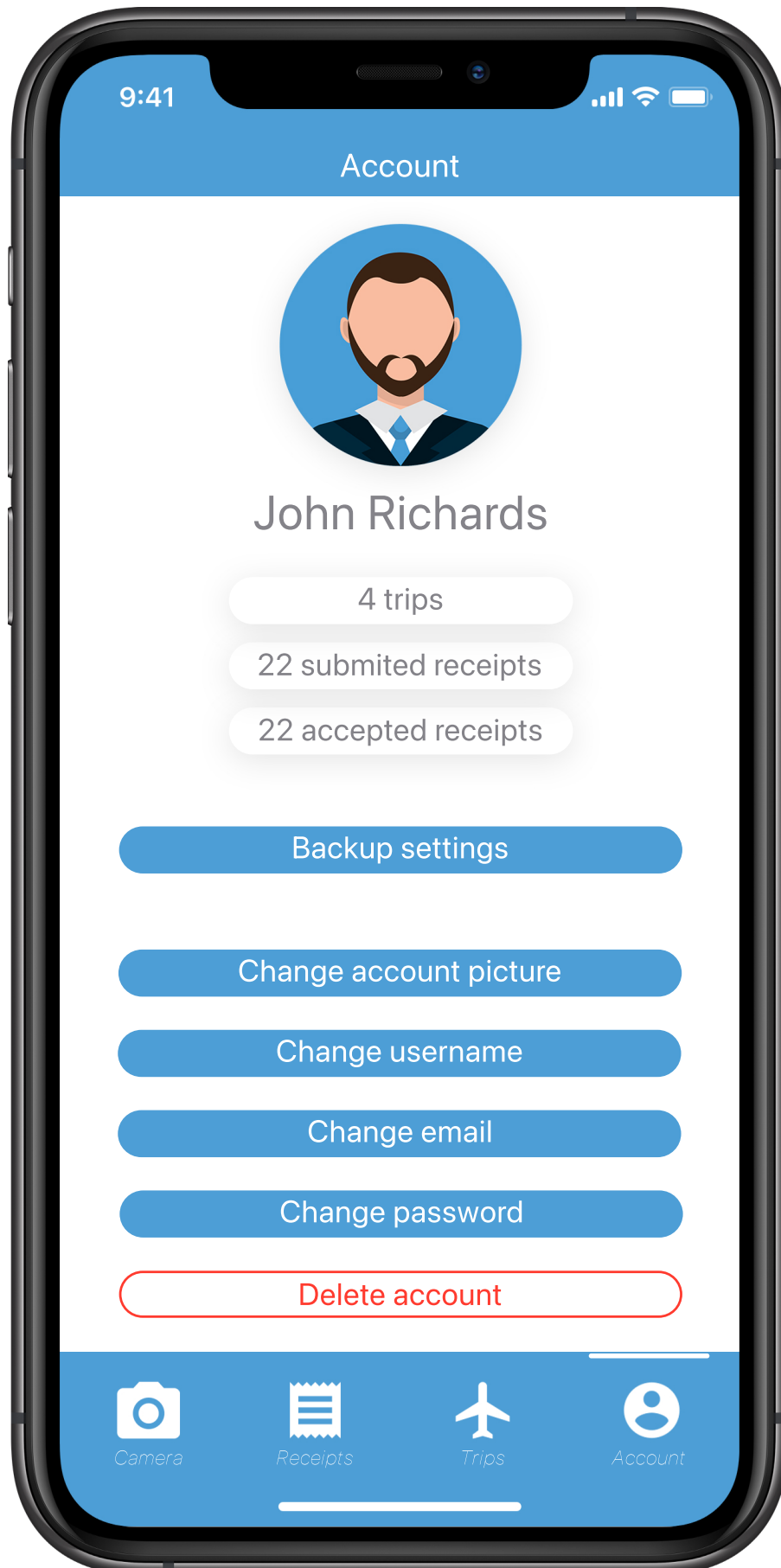
.2.4 UI Prototype - V3: Receipt overview screen



.2.5 UI Prototype - V3: Trips screen



.2.6 UI Prototype - V3: Account screen



.3 Bosch section presentation

This section shows the *powerpoint* that was developed for an internal presentation.



ANDRÉ ARAGÃO

Microelectronics and Embedded Systems
Intern @ Engineering Software Department (XC-CI2/ESD5)
Bosch Car Multimedia Portugal, S.A.



Internal | CM-CISECS-Eng | 2020-07-04

© Robert Bosch GmbH 2020. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



Agenda

1. Problem overview
 1. Bosch's reimbursement protocol
 2. Proposed solution
2. System overview
 1. System Overview
 2. System Overview – Technical aspects
3. Mobile application
 1. Application UI design
 2. Application development stage
4. Back-end server

Internal | CM-CISECS-Eng | 2020-07-04

© Robert Bosch GmbH 2020. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



PROBLEM OVERVIEW



<https://www.overleaf.com/project/60ccb1befc8a8ddef86ac4b9>

Problem overview Bosch's reimbursement protocol



Fig.1: Bosch's reimbursement protocol diagram

Internal | CM-025EC50-8kg | 2020-07-04

© Robert Bosch GmbH 2020. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



Problem overview Proposed solution



Fig.2: Proposed solution diagram

Internal | CM-025EC50-8kg | 2020-07-04

© Robert Bosch GmbH 2020. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



SYSTEM OVERVIEW



System overview

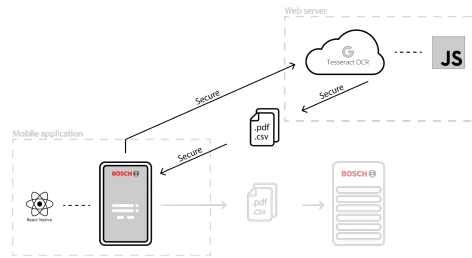


Fig.3: System overview diagram

Internal | CM-CISECS-00g | 2020-07-14

All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



System overview

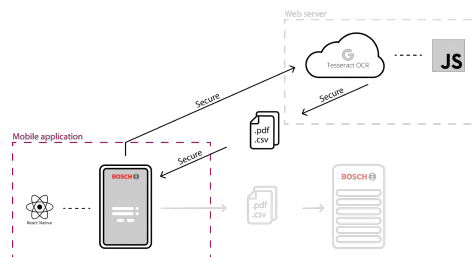


Fig.3: System overview diagram

Internal | CM-CISECS-00g | 2020-07-14

All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



System overview

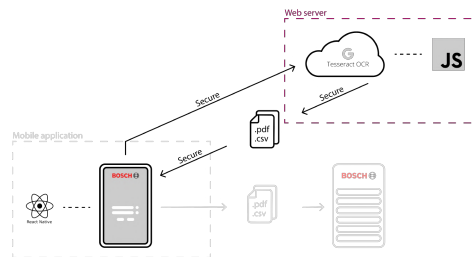


Fig.3: System overview diagram

Internal | CM-CISECS-00g | 2020-07-14

All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



System overview

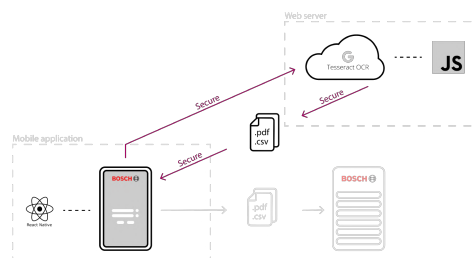


Fig.3: System overview diagram

Internal | CM-CISECS-00g | 2020-07-14

All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



System overview

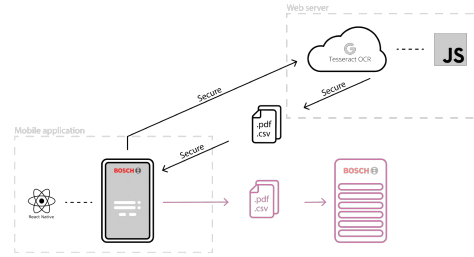


Fig.3: System overview diagram

Internal | CM-CSECS-00g | 2020-07-14

Copyright Bosch System-Office. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



SYSTEM OVERVIEW: TECHNICAL ASPECTS



System overview Stack

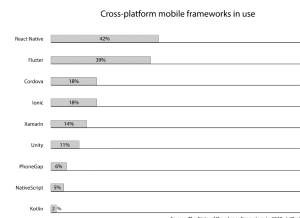


Fig.4: Cross-platform mobile frameworks in use

Internal | CM-CSECS-00g | 2020-07-14

Copyright Bosch System-Office. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



System overview Stack

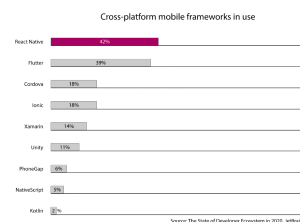


Fig.4: Cross-platform mobile frameworks in use

Internal | CM-CSECS-00g | 2020-07-14

Copyright Bosch System-Office. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



System overview

Stack

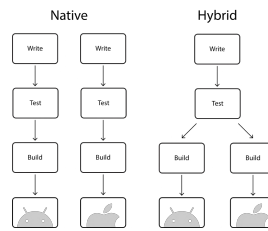


Fig.5: Native vs. Hybrid

Internal | CM-CD/ECSS-Eng | 2020-07-14

Copyright Bosch System (2020). All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



System overview

Stack

MERN

Internal | CM-CD/ECSS-Eng | 2020-07-14

Copyright Bosch System (2020). All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



System overview

OCR Software



Internal | CM-CD/ECSS-Eng | 2020-07-14

Copyright Bosch System (2020). All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



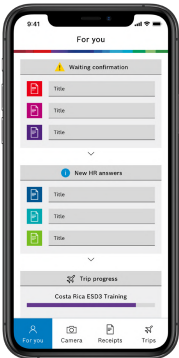
MOBILE APPLICATION



Mobile application
App logo



Mobile application
Dashboard



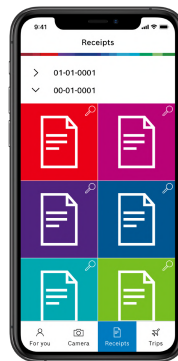
Mobile application
Camera



Mobile application
Camera



Mobile application Receipts



Internal | CM-CECCB-00g | 2020-07-14

© Bosch Power Tools GmbH. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



Mobile application Receipts

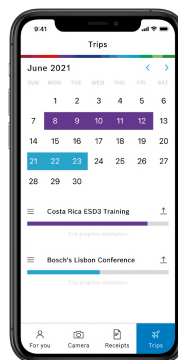


Internal | CM-CECCB-00g | 2020-07-14

© Bosch Power Tools GmbH. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



Mobile application Trips

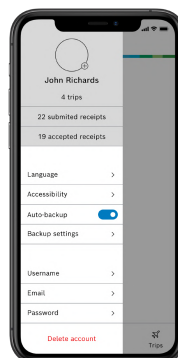


Internal | CM-CECCB-00g | 2020-07-14

© Bosch Power Tools GmbH. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



Mobile application Side menu

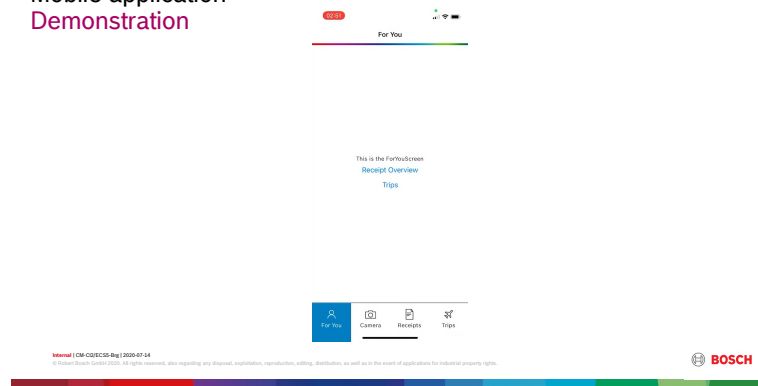


Internal | CM-CECCB-00g | 2020-07-14

© Bosch Power Tools GmbH. All rights reserved. Also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.



Mobile application Demonstration



BACK-END SERVER



Back-end server Preliminary OCR results



POPPIES
55-59 OLD COMPTON STREET
SOHO
LONDON W1D 6HW
YELIZ REG 1081
SUNDAY 31 JULY 2018 18:34 000053
TTUORDER NO: 44
1 CHIPS LRG £2.50
2 CODREG £13.00
1 CALAMRI £6.50
TOTAL £22.00
cASH £22.00
VAT No

30 Internal | CM-CECCSO-Reg | 2020-07-14



Future milestones

► App development	02/06/21
► Back-end finalization and integration	09/06/21
► Performance analysis	23/06/21
► Thesis writing	30/06/21
► Further improvements	30/07/21

31 Internal | CM-CECCSO-Reg | 2020-07-14



QUESTIONS?
THANK YOU!



Bibliography

- [1] *Web Content Accessibility Guidelines (WCAG) 2.1*. Version 2.1. URL: <https://www.w3.org/TR/WCAG21/>.
- [2] *How to Meet WCAG (Quick Reference)*. Version 2.1. URL: <https://www.w3.org/WAI/WCAG21/quickref/>.
- [3] *OWASP Web Security Testing Guide*. Version 4.2. URL: <https://owasp.org/www-project-web-security-testing-guide/>.
- [4] *OWASP Cheat Sheet Series*. Version 2.0. URL: <https://owasp.org/www-project-cheat-sheets/>.
- [5] Arindam Chaudhuri et al. *Optical Character Recognition Systems for Different Languages with Soft Computing*. Vol. 352. Studies in Fuzziness and Soft Computing. Springer International Publishing, 2017.
- [6] Arindam Chaudhuri. “Studies in Applications of Soft Computing to some Optimization Problems.” PhD thesis. 2010.
- [7] Francis T. S. Yu and Suganda Jutamulia. *Optical Pattern Recognition*. Cambridge University Press, 1998.
- [8] Ahmad P Tafti et al. “OCR as a service: an experimental evaluation of Google Docs OCR, Tesseract, ABBYY FineReader, and Transym.” In: *International Symposium on Visual Computing*. Springer. 2016, pp. 735–746.
- [9] P. Que, X. Guo, and M. Zhu. “A Comprehensive Comparison between Hybrid and Native App Paradigms.” In: (2016), pp. 611–614.
- [10] *The State of Developer Ecosystem in 2020 Infographic*. 2020. URL: <https://www.jetbrains.com/lp/devecosystem-2020/>.
- [11] Wenhao Wu. “React Native vs Flutter, Cross-platforms mobile application frameworks.” In: (2018).
- [12] Jakub Jagiello. “Performance comparison between React Native and Flutter.” PhD thesis. 2019. URL: <https://www.diva-portal.org/smash/get/diva2:1349917/FULLTEXT01.pdf>.
- [13] William Danielsson. “React Native application development – A comparison between native Android and React Native.” PhD thesis. 2016. URL: <https://www.diva-portal.org/smash/get/diva2:998793/FULLTEXT02.pdf>.
- [14] H. Brito et al. “JavaScript in mobile applications: React native vs ionic vs NativeScript vs native development.” In: (2018), pp. 1–6.
- [15] Dennis Bilberg. “Comparing performance between React Native and natively developed smartphone applications in Swift.” PhD thesis. 2018. URL: <http://www.diva-portal.org/smash/get/diva2:1215717/FULLTEXT01.pdf>.
- [16] Teddy Mantoro, Abdul Muis Sobri, and Wendi Usino. “Optical character recognition (OCR) performance in server-based mobile environment.” In: *2013 International Conference on Advanced Computer Science Applications and Technologies*. IEEE. 2013, pp. 423–428.
- [17] *Tesseract Open Source OCR Engine (main repository)*. Version 4.1.1. URL: <https://github.com/tesseract-ocr/tesseract>.