

BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER
SCIENCE
SPECIALIZATION COMPUTER SCIENCE IN
GERMAN

DIPLOMA THESIS

Vehicle management using OCR for
extracting data from documents

Supervisor

Lecturer Diana Cristea, PhD

Author

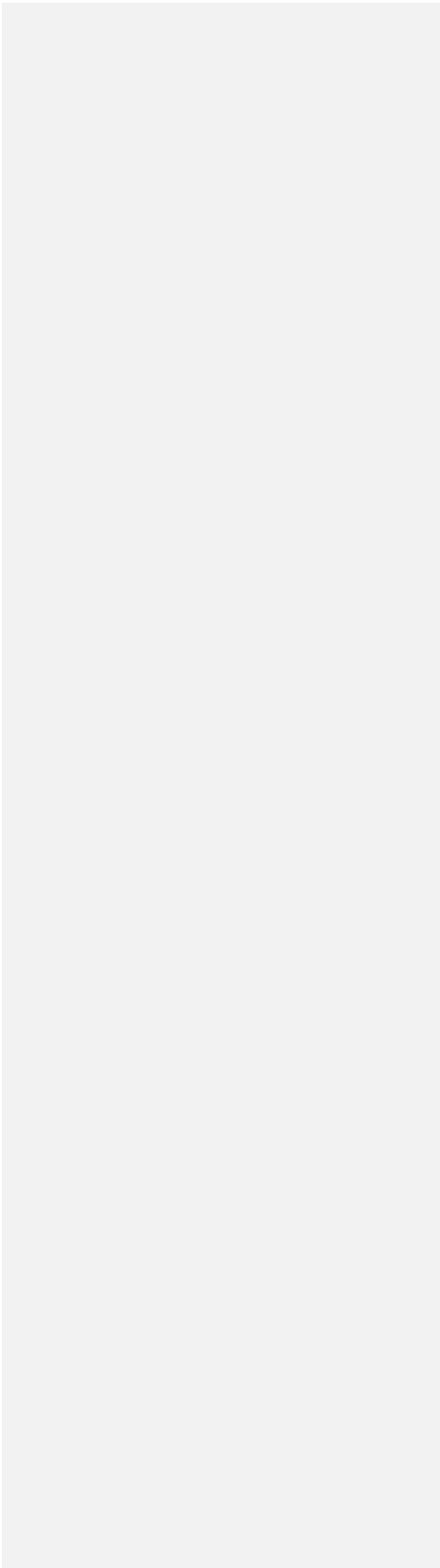
Cîrstea Ștefan-Daniel

2022

Table of Contents

Introduction	4
Scope and motivation	5
Optical Character Recognition (OCR)	6
Web applications.....	8
Server-side technologies	9
Relational Database	12
Client-side technologies	13
Application	15
Application structure and deployment schema.....	15
Optical Character Recognition (OCR)	16
Preprocessing	16
Text extraction	17
Postprocessing	17
Exposing	18
Server-side.....	18
Structure.....	18
Authentication & Authorization	19
Vehicle permissions.....	20
Client-side.....	24
Structure.....	24
Authentication context	24
Database structure	26
References.....	27

Introduction



Scope and motivation

Today, there are more applications that help people remember things, and many ways to store data about your daily activities. But some deadlines are easy to be forgotten especially when it takes more than one or two months to take place. Because of that, I had chosen to create an application which scope is to help people manage all their vehicle information.

Managing vehicles using mobile or web-based applications is not a new idea and it has been done before, an example of this kind is MOVCAR [1]. Because I know that time is very important in our days, anyone have more to do every day and because I am a driver, I like driving and I know I will always be a driver, my application is looking to help people manage their vehicles. Moreover, is looking to help them by extracting data from their documents.

There are more other applications that remind people about their periodical mandatory activities or documents that need to be done in order to be safe as a driver. Table 1 shows a comparison between some of this apps and the app developed by me.

Application / Features	Deadline reminder	Fleet management	Fuel Management	Data extraction	Web solution	Service management	Cloud solution
MvManagement	X	X	X	X	X	X	X
MOVCAR	X	X	X		X		X
MyCar	X						X
Car Alert	X						

Table 1 - Applications comparison

Optical Character Recognition (OCR)

Optical Character Recognition or known as OCR represents a technology used for recognizing and extracting text within a digital source as images and converting information into a machine-readable form in order to use it for data processing. OCR principle seem to be straightforward, but it's implementation may not be that simple because of the variety of fonts and letter spacing formatting. Even then the most difficult job for OCR systems remains recognizing and extracting handwritten text because there are very few writing styles that match, and most offend it will be different.

The process of OCR involves three main objectives that are reached using a series of steps. These objectives are pre-processing of the image, character recognition and post-processing the output. As expected, the first step of OCR is scanning the document. By scanning we can ensure that the document is well aligned and fit a size pattern. This method encourages the correctness and efficiency of text extraction. Preparing the image, this step is focused on removing imperfections from image. This step aims to create a focus on characters and to sharpen them in order to make text clear. The next step is designed to make the document as simple as possible, and it can be called Binarization. Binarization process states in redefining the image document so that it is bi-component build, containing only black and white colors. Black and dark areas are considered to be text zone and the same time the white and light areas are considered background and are ignored, that technique encourage an optimal recognition of the characters. Another phase and the one for which all pre-processing of the image steps were needed is recognizing characters phase. In this phase, the black areas are processed in order to recognize letters or digits. Mainly, OCR focuses on one character or on a block of text. Pattern recognition and feature detection represent the two algorithms mainly used for recognizing characters. Pattern recognition involves training the OCR software in a way that it can learn the font and format. The software is then used for comparing and recognizing characters in the scanned document. Through Feature detection algorithm the OCR software recognize letters and digits by their features in the scanned document. Features in this context can include number of angled lines, curves or crossed lines. A very common use of OCR is vehicle registration plate recognition and now it is implemented in traffic monitoring

camaras. In Figure 1 - OCR Example from PaddleOCR is showed an example found on GitHub at PaddleOCR [2] is showed.



Figure 1 - OCR Example from PaddleOCR [15]

The very specific part of recognizing characters is called OCR Engine. OCR is not a very new technology so there are more OCR Engines available and each of those have advantages and disadvantages. For using some OCR Engines paying a tax is required and taking into count he accuracy of the engine the cost can be higher. Most used solution is Google's Tesseract which can be the best option by the fact that it is free, because it is an open-source OCR Engine. The result of this engine maybe is not as accurate as the ones from ABBY FineReader which is a paid option.

---- object detection ----

Web applications

Web applications are developed software that runs on a web server and can be remotely accessed from other devices through a browser interface. This kind of applications are designed for a variety of scopes from online calculator to web mailing applications, e-commerce shops and even more complex applications. They are usually built from three very important and decoupled parts, frontend or client-side, backend or server-side and database. Client-side represents all the graphical interfaces, directly, what the user sees. Backend is the part in charge of creating the login of the application and the mediator between the frontend and the database, basically its scope is processing data and sending it to the client-side. Database part is the one which is storing the data and is responsible of the correctness of it.

Web applications do not need to be downloaded, in order to run they need a web server, application server host the logic and a database. Web servers are responsible for hosting the client-side of the application and for managing the request that comes from the client. Application server completes a requested task and then returns data to the web server and database is basically used for storing any information needed.

Communication between web application components is also a very important part in application development process. One of the most used interfaces for transferring data bidirectional between software applications are REST APIs (Application Programming Interface). APIs are designed as a bridge that let data travel between two applications. Basically, this is how the backend and frontend most commonly communicate over the HTTP protocol. The HTTP protocol is the most used protocol for transferring information between a Web Browser and a web server, and it represents a text protocol. The entire communication process is illustrated in Figure 2 - Application Rest API communication [3].

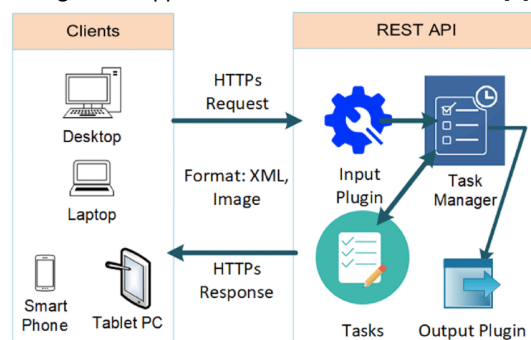


Figure 2 - Application Rest API communication

In terms of technologies, it was a hard decision to be made. And for this application several options were available for the client-side also for the server-side.

For choosing a backend technology more factors were taken into account and more frameworks. The first option for this application as a server-side technology was for sure .NET because it is fast, can be deployed on any platform and I am most familiar with it, but it has the disadvantage that you cannot create AI components that easy. The Second option was Java with Spring framework, because it is open to OOP writeable code, it is a fast-learning framework and it is very used worldwide, but as well as .NET it does not provide an easy AI solution. Another option was Python because code is very easy to write, it has support for AI, APIs are very easy to write, but as a weakness, it does not provide a strong typing convention.

In terms of client-side solutions the following options were considered: Angular because it is a component-based solution, it has some support from ASP.NET Boilerplate and it is the most used web framework with .NET, but it will generate a lot of files in the end; React.Js because of their growth over time and because it uses components and it has some awesome UI kits available, but the disadvantage is that development starts from scratch. Also, I considered using .NET Razor pages because it does not need two projects, it can bind DTOs directly, but it means that classic styling with CSS and JavaScript should be used.

After research, MvManagement (MvM) is using the latest version of technologies available for now. This application is designed on top of .Net Core 6 which was released on November 8, 2021, according to Microsoft official website [4], for the backend of the application. In terms of client-side technology MvM is using the last version of the React.Js at the moment of writing, one of the most known and used java script framework taking into account the statistic done by Stack Overflow [5]. This statistic shows that React.Js is one of the most preferred and most used web frameworks and it will be clear why after next paragraphs.

Server-side technologies

As mentioned, this project is built on one of the most used technologies all over the world. Why .Net? Because .Net is a widely used solution for developing applications, because it is fast, it is maintained by one of the world's giant companies, Microsoft, allows asynchronous programming. It is cross platform allows deploying on any platform not only on Windows; this concept is supported by Microsoft on their website [6]. Last but not least .Net is open source that means that everyone can contribute add their knowledge in order to get the best from

this framework. In terms of programming languages .NET uses C# which is a solution that encourage all the OOP related principles as Encapsulation, Composition, Aggregation, Inheritance, Abstraction and so on.

The version of the .NET was an important decision to be done, and this MvManagement application is built on .NET 6 because it is an LTS (Long term support) release, which means that it benefits from Microsoft support for more time. According to the book “C# 10 and .NET 6 – Modern Cross-Platform Development” by Mark J. Price [7] the LTS versions of .NET are supported since a new LTS version is released and Current versions are supported just for a period after a newer version come out.

Driving without rules is impossible and everything will be a disaster, a lot of crashes, standstills and so on. That can be also applied for coding and there are also some rules and principles for writing. One of the most known and relevant rules and best practices for Object-Oriented programming (OOP) are the five SOLID Principles. Each letter of the word SOLID names one principle. First principle is the Single Responsibility principle, this principle states that a class should do only one thing. The next principle described is the Open-Closed principle, it states that classes from OOP should be open for extension and closed for modification. This means that code inside classes can not be modified but the name class can be extended, more functionalities can be added. The third principle, corresponding to letter L, the Liskov Substitution Principle which states that child classes should be substitutable for the parent classes. Next principle, the Interface Segregation Principle is about separating interfaces. The reason why this principle appeared is because a class should not be forced to implement methods that are not required. The last principle, corresponding to letter D, the Dependency Inversion Principle states that classes should depend on Interfaces and abstraction not on concrete classes.

MvManagement is built on top of ASP.NET Boilerplate framework [8] created by Volosoft and maintained by the entire community because it is an open-source framework, moreover, it also benefits from the support of the .NET Foundation. It is a Framework design on top of SOLID principles that is a very good solution for developing web application with actual practices and tools. What does it provide? ASP.NET Boilerplate (ABP) from my use and according to their documentation it offers a layered architecture based on DDD (Domain Driven Design), modularity, the possibility to build your own modules on top of their modules, multitenancy, a way of storing data about different entities on the same server named by

Gartner Glossary [9] and also by the ABP official documentation. In additions, to this awesome programming features, ASP.NET Boilerplate provides a very useful documentation and a prompt, helpful GitHub community. Some common structures provided by ABP are Dependency Injection, session managing, caching, logging and setting management.

Dependency Injection is a software design pattern which aims to separate the declaration of the dependency from the implementation itself. This way by using dependency injection programs manage to be loosely coupled and to follow the dependency inversion and single responsibility principles. In .Net the design of dependency injection is mainly done by creating and interface which contains the structure of the methods, name of the method, returned type, parameters and their type. Interface is implemented by a class or more classes. Further other classes can use the interface methods without knowing how methods are implemented because it is ensured by the interface that the result will be the one expected. Dependency injection is managed inside MvM by ASP.NET Boilerplate framework. It uses the (Castle Windsor [10]), open source, framework for this and some strict naming conventions. By only using Castel Windsor the dependencies should be mapped manually by declaring a Container and then registering each dependency something like:

With the help of the ABP this kind of dependencies are automatically registered. The naming convention is that if there is any class that implements an interface called `IVehicleAppService` and its name contains `VehicleAppService` postfix it will be automatically registered as implementing `IVehicleAppService`. Classes can have other names without following the naming convention and can be registered in dependency injection container, but it must be done manually.

```
Component.For<IVehicleAppService>().ImplementedBy<VehicleAppService>().LifestyleTransient()
```

Session is another very important feature provided by ASP.NET Boilerplate. Basically, it provides an interface `IAbpSession` which can be injected and can be used to obtain information about current user and tenant without using ASP.NET's Session according to their documentation [8]. `AbpSession` defines a few key properties like, `UserId` which represents the id of the current or it can also be null if there is no current user. `TenantId` another key property and one of the most used describes the id of the current tenant or null if current user is not assigned to a tenant.

Relational Database

In terms of databased MvM remains stuck to Microsoft technologies and it uses Microsoft SQL Database. MS SQL represents a relational database. A relational database is a collection of data items with predefined relation between them. These items are organized in a set of tables with columns and rows. Each table has a strongly defined structure which describe specific kind of object. Each column states a specific attribute and has a specific type. Each object stored in the table represent and row in that table. Each entry in a relational database table can be unique identified by a property called primary key. The relation between two rows from deferent tables is described by a foreign key.

SQL states for Structured Query Language. SQL is the mediator used to communicate with a database management system. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform different action on database, such as updating an entry, creating a new entry, querying a specific table.

In order to get an efficient way of creating, mapping and using database models MSSQL is combined with Entity Framework 6 [11]. According to Microsoft, Entity framework is an ORM (Object Resource Mapping) tool that allows CRUD operations without having to write SQL queries. Entity Framework provides two important ways of creating mapping, code-first and database-first. Code-first approach as the name says states that the first done is the code, this means that entities are designed in code using some tools provided by Entity Framework than migrated to database by using EF-cli (CLI = Command Line Interface). After entities are designed migration is added using "Add-Migration Migration-Name", then command "Update-database" in order to apply the migration to the database. The second approach, Database-first, means that firstly the database is designed using specific tools or directly from code and then the mappings are done in code. For the MvManagement the method used is code-first because this way some code can be written, tested, the migration added and then the database updated.

Client-side technologies

For rendering the user interface MvManagement is using React.js library. This library builds Single-Page application. According to the article “Single-page application vs. multiple-page application” written by Neoteric on medium.com [12] single-page application are web solutions that do not need re-rendering a page in order to display new content. And the biggest advantage of using single-page applications is that all the resources (HTML, CSS, Script) are loading once in the app life. Also, this is a component-based JavaScript library, that means it builds component with their unique management context and this results in complex UI.

Single page applications, as React.js builds, need to have an own mechanism to change how elements are displayed inside page. In React applications, this management is done using states. Basically, every component builds in react uses one or more state and all components are rendered dependently form a state. In order to give the developer, the option to manipulate these states, it provides some special methods called Hooks. This Hooks follow a strict convention and all start with the prefix “use”. Some of the most common used states are useState, useEffect, useRef. Each of these Hook is used for very specific action. useState hook is developed to create a local state inside a React component, it provides a getter and a setter for the state created. Another important hook, useEffect, as the name said it react to an effect mainly a state change. If the state is not specified the actions inside hook take place at component render event. A bit different to the others, useRef hook, provides a way to refer another component and its properties.

An advantage of this library is that it is open-source, and you can easily find support. Facebook from 2021 named Meta, decided to use React as an open to development project from 2013 since now, according to their GitHub release history [13]. Now is one of the most used and preferred web framework by developers taking into count the study done by Stack Overflow [5].

In addition to React.js for this application uses an UI kit named Chakra UI [14]. This UI library was relatively recent released, in 2020, and is very popular among developers because is easy to use and is lined up with react actual standards. Chakra is offering a way of writing fewer and cleaner code, but also let developers extend component in order to get the best for them.

Components represent the most useful part of using an UI kit and Chakra is providing a lot of them. According to their website at the moment of writing they are offering components from 11 different categories. A few of them had an important role in choosing Chakra UI, layout components that are designed to keep your website appearing on all size of the screen, no matter if people are using it from phone or from a desktop. Another category of components that have a very important role to the aspect of the website is Form components category. Forms components provided by this UI kit are appreciated because they are offering nice looking buttons which can be easily be customizable. Moreover, it provides already design inputs on top of which developer can add its signature in order to make it looking good. There is more other small component have a great impact to your website as Breadcrumbs, Icons, Alerts, Tooltips and so on.

Installing UI kits can sometimes be very annoying, and you must read a lot of rigid documentations. For Chakra UI this is not the case. Installing it is as easy as wrapping the application tag in a “ChakraProvide” tag. This feature was very appreciated by the developer community.

Application

Application structure and deployment schema

My Vehicle Management application has four main parts. Server-side which represents a .NET Application that covers the application logic. The server side is strongly coupled with the SQL database. The third part is the OCR application designed in a FastAPI REST Services application using python and Tesseract. And all of this are collected and offered to user in the client-side. The client side is a React.JS application that consumes the endpoints from backend and OCR application.

Many technologies being involved means many places to host. The application environment is hosted on 3 different platforms. The best solution found in terms of performance and costs for backend application is Azure App service. It that is also an advantage that both .Net and Azure are maintained by the same company, Microsoft. For MS SQL database the best hosting solution comes also from Microsoft, namely SQL database basic plan from Azure. Cost, performance and fast deploying option make Firebase hosting the best solution for the React.JS frontend application. Last but not least, the OCR application, a python developed app fits the best on Heroku hosting. A clear vision about deployment schema is shown in Figure 3 - Deployment diagram.

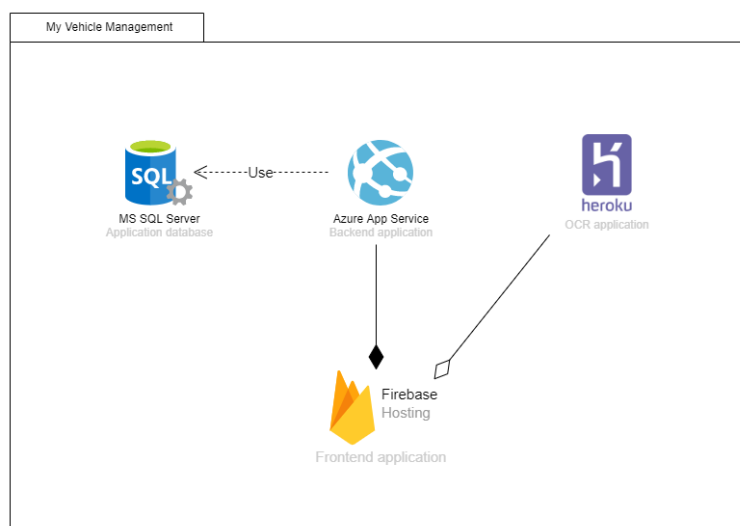


Figure 3 - Deployment diagram

Optical Character Recognition (OCR)

Reading text from image covers more just recognizing characters. To have an accurate result preprocessing is necessary. After preprocessing comes into place the text recognition itself using the existing engines, free or paid. Then the last step in transforming text inserted in image into a computer useable text is postprocessing. These all steps are required for an efficient OCR process. For achieving the text extraction, we chose to use python together with some external libraries. The libraries used are FastApi for creating endpoints, OpenCV for image processing, numpy for boxing image, and Tesseract python library for extracting the data.

Preprocessing

Preprocessing is the most complex step of OCR process. For the case described this application the preprocessing steps are resizing the original image, gray scaling, removing the imperfections and not needed lines and extracting the paper or document from original image. Figure 4 shows the original image after resizing, the end result and the steps between.

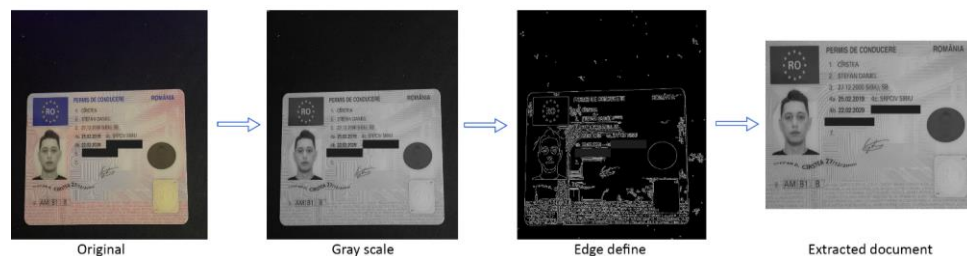


Figure 4 - Image processing steps for driving license

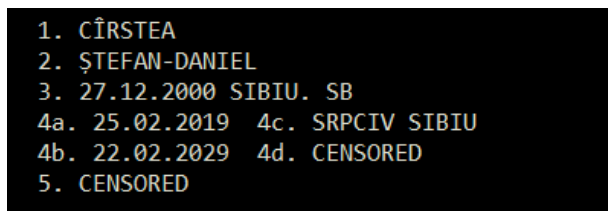
First step in preprocessing the image for scanning the driving license is to resize the image, in this case the image was resized to 900 pixels width and 1600px height assuming that the image was done in portrait mode. Next step is removing the colors from the image because the OCR engines recognize dark text highlighted on white paper and doing this the OpenCV library was used, namely the `cvtColor` method. Another OpenCV method needed for performing the line detection, it is called Canny and the result is the one in Edge define image from figure 4. In this step was also added a gaussian blur effect to image in order to remove the fine line from the driving license card. The last step on preprocessing level is capturing the driving license card in this case. This extraction was also done using OpenCV methods.

Text extraction

After the image preparation it can be accurately extracted by the OCR engine. The one chosen for performing this step involved the Tesseract OCR. Because the recognition set up was done for a Romanian document the option for the Tesseract language is Romanian. Another configuration option chosen for Tesseract OCR engine is to set it up on automatic page segmentation, this is because image input with preprocessing can have some imperfections.

Postprocessing

After the text is successfully extracted from image it is time for the preparation in order to be delivered to the end user. For a clear vision about how the data is structured the elements were transformed in a python dictionary. The dictionary components are last name, first name, date from when the license is available and the date when the license expires. Because the text is extracted as a page, that means it will have the same structure as in the original document. In the best case when all characters are recognized correctly the end result of the OCR process will look like in Figure 5.



```
1. CÎRSTEA
2. ȘTEFAN-DANIEL
3. 27.12.2000 SIBIU. SB
4a. 25.02.2019 4c. SRPCIV SIBIU
4b. 22.02.2029 4d. CENSORED
5. CENSORED
```

Figure 5 - Extracted text for a Romanian driving license

To format the text into a python dictionary the text needs to be sliced by type. This way the post processing is designed to split the text received after OCR by lines first. Then for each line the words are extracted. For name extraction pattern the first word is ignored and the others are marked as first name, respectively last name. Another pattern for this case is the one for lines containing date. Lines are divided by word, the first word is again ignored and the next is considered to be a date. After date extraction the word containing date follows a recognition process. The recognition process represents a regex for date that should match in the extracted word. If the recognition process does not match at first try a replace is performed in order to replace the commas with a dot because is a very common mistake the OCR engine makes on parsing the text.

Exposing

For creating a really usable feature the OCR should be accessed remotely. The solution for this issue is to create an endpoint which receive an image and then triggers all the OCR parsing steps. The API was created using the FastAPI python library. Exposing endpoints throw internet means that anyone can access it and you need to secure your endpoint to be sure that just the allowed one can access the endpoint. In this case the text extraction endpoint was restricted by the DNS. This restriction was done using an allow CORS origins restriction. This Allow CORS origins means that just the whitelisted origins can access the endpoint. In this case the whitelisted server is just the frontend server which consumes the resource.

Server-side Structure

Server side or the kernel of the application has a multi-module and multi-project structure, where each module and project have its role. Main application has as component six projects, namely, Application, Core, EntityFrameworkCore, Migrator, Web.Core and Web.Host and this projects also have reference between them. Except from main application, there are other decoupled modules. There is a module for catalogues containing auto related catalogue project, documents related catalogue project and insurance companies related catalogue. Another module contains the Abp.SendGrid module for SendGrid mail provider. And the module that ensure us that backend is working as we expect is the tests module. The backend application structure is also described in Figure 6 and offers a clearer vision about the above described projects and how they are linked one to each other.

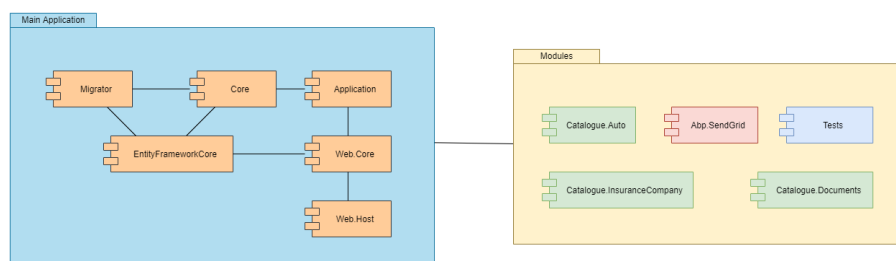


Figure 6 - Backend application projects structure

As previously mentioned, each project from backend application has its own scope and role. Entity Framework (EF) is used in the application through the EntityFrameworkCore project. That means that this project represents our middleware to communicate with the SQL database. Because the applications are designed on a code-first approach for EF the migrations added through EF should be reflected also in database. The project responsible for take the database up to date and to apply EF migrations is the Migrator project. An important project is the Core project, it stores the business logic, entities, helpers, extensions, authorization, and other data that should not be exposed to consumer. Application project is the biggest Core project consumer and is strongly dependent of it. This project stores the application services, DTOs (Data Transfer Objects) and all the logic that is meant to serve the consumer needs. A very important project is the Web.Host one, because it represents the communication with the end consumer of the APIs. Representing the communication middle with the consumer the Web.Host project main role is to store the endpoints definition. Another middleware project is the Web.Core which as expected represent the mediator between controllers from Web.Host and the application services from the Application project.

Authentication & Authorization

In order to keep user information safe and secured, the server-side application is designed to expose information only if the user is authenticated in the application. For this application authentication is done using JwtBearer authentication which generates and validation token using the endpoint “/api/TokenAuth/Authenticate” that provides user access to secured endpoints. This token is unique and user specific and is randomly generated based on security key. The secured endpoints are marked using Authorized annotation.

There are also some endpoints that are meant to be used only if the user has a specific authorization, for example to see the organization users. This is done using permission and roles. Endpoints that need specific permission to be used this are marked with an annotation as in Figure 7. This authorization can be applied also for classes and the effect is applied for all exposed methods inside class.

```
[AbpAuthorize(params permissions: PermissionNames.Pages_Users)]  
0 references | Stefan Cirstea, 22 days ago | 1 author, 2 changes  
public override async Task DeleteAsync(EntityDto<long> input)
```

Figure 7 - Permission authorization for endpoints

This token is used as user signature, because based on this token it is obtained the current logged in user or whether the user is authenticated. Based on this token we have the certitude that we can provide user information about his account because it is certain that is his account.

Excepting the endpoint permission itself in application are also implemented some vehicle specific permissions described in the next chapter.

Vehicle permissions

The part that makes the backend of the application different is that the app is design for both companies and individuals. This is because companies have the option to have custom roles, both for company entity inside application and for one vehicle itself. For example, a transport company, there are many drivers, as a company owner you want drivers to be aware if the vehicle, they are driving has an insurance or technical inspection that is about to expire.

<pre>[Table(Name = "tblVehiclePermission", Schema = "veh")] 59+ references Stefan Cirstea, 61 days ago 1 author, 5 changes public class VehiclePermission : Entity<long>, ICreationAudited { [Key] [Required] [Column(Name = "IdVehiclePermission")] public override long Id { get; set; } 56 references Stefan Cirstea, 63 days ago 1 author, 2 changes public override long Id { get; set; } 0 references Stefan Cirstea, 63 days ago 1 author, 1 change public DateTime CreationTime { get; set; } 0 references Stefan Cirstea, 63 days ago 1 author, 1 change public long? CreatorUserId { get; set; } [Required] [Column(Name = "Name")] 59+ references Stefan Cirstea, 63 days ago 1 author, 2 changes public string Name { get; set; } [CanBeNull] [Column(Name = "PermissionDescription")] 4 references Stefan Cirstea, 67 days ago 1 author, 1 change public string Description { get; set; } [ForeignKey(Nameof(Vehicle))] [Column(Name = "IdVehicle")] 20 references Stefan Cirstea, 61 days ago 1 author, 3 changes public long? IdVehicle { get; set; } 1 reference Stefan Cirstea, 63 days ago 1 author, 1 change public Vehicle Vehicle { get; set; } [CanBeNull] [ForeignKey(Nameof(VehicleRole))] [Column(Name = "IdVehicleRole")] 20+ references Stefan Cirstea, 63 days ago 1 author, 1 change public int? IdRole { get; set; } 1 reference Stefan Cirstea, 63 days ago 1 author, 1 change public VehicleRole VehicleRole { get; set; } [CanBeNull] [ForeignKey(Nameof(User))] [Column(Name = "UserId")] 0 references Stefan Cirstea, 63 days ago 1 author, 2 changes public long? UserId { get; set; } 1 reference Stefan Cirstea, 63 days ago 1 author, 1 change public User User { get; set; } }</pre>	<pre>[Table(Name = "tblVehicleRole", Schema = "veh")] 50 references Stefan Cirstea, 51 days ago 1 author, 4 changes public class VehicleRole : FullAuditedEntity, IHaveTenant { [Key] [Required] [Column(Name = "IdVehicleRole")] public override int Id { get; set; } 12 references Stefan Cirstea, 63 days ago 1 author, 2 changes [Required] [Column(Name = "Name")] 18 references Stefan Cirstea, 61 days ago 1 author, 2 changes public string Name { get; set; } [CanBeNull] [Column(Name = "Description")] 10 references Stefan Cirstea, 64 days ago 1 author, 2 changes public string Description { get; set; } 23 references Stefan Cirstea, 64 days ago 1 author, 1 change public int? TenantId { get; set; } } [Table(Name = "tblVehicleRoleUser", Schema = "veh")] 9 references Stefan Cirstea, 63 days ago 1 author, 3 changes public class VehicleRoleUser : Entity<long>, ICreationAudited { [Required] [Column(Name = "Id")] 0 references Stefan Cirstea, 63 days ago 1 author, 2 changes public override long Id { get; set; } 0 references Stefan Cirstea, 64 days ago 1 author, 1 change public DateTime CreationTime { get; set; } 0 references Stefan Cirstea, 64 days ago 1 author, 1 change public long? CreatorUserId { get; set; } [Required] [Column(Name = "IdVehicleRole")] 6 references Stefan Cirstea, 64 days ago 1 author, 1 change public int IdRole { get; set; } [Required] [Column(Name = "IdVehicle")] 3 references Stefan Cirstea, 63 days ago 1 author, 2 changes public long IdVehicle { get; set; } [ForeignKey(Nameof(User))] [Column(Name = "UserId")] 7 references Stefan Cirstea, 63 days ago 1 author, 2 changes public long? UserId { get; set; } 1 reference Stefan Cirstea, 64 days ago 1 author, 1 change public User User { get; set; } }</pre>
---	--

Figure 8 - VehiclePermission, VehicleRole and VehicleRoleUser entities

Vehicle permission management feature is based on three important entities, VehiclePermission, VehicleRole and VehicleRoleUser. VehicleRole represents the roles that are defined and can be assigned to a person referencing a vehicle. VehicleRoleUser is the linker entity and contains information about what role, on which vehicle the user has. Finally, the VehiclePermission entity, which is more complex, represent the permission itself with its

description and can be assigned both for role and for individual user. These entities are illustrated in Figure 8.

The extended classes are provided by ASP.Net boilerplate. Entity class ensures that the class that inherit it has the Id property to be transformed in a database table. FullAuditedEntity also implements the Entity class, but it also has some specific properties. This class provides some properties that keep track of creation time, creator user id, last modification time, deletion time and more that are automatically handled by ASP.Net Boilerplate at runtime. More this class offers the soft delete option. Soft delete is basically a column named IsDeleted, if its value is positive than the database entry is ignored.

As an aside, it is important to point out that Figure 6 shows how Entity Framework code first approach is implemented. The idea is that mainly the database models are creating by using annotations. Table annotation is mandatory for Entity Framework to what is the name of the creating table and optionally the schema can be added as in the case showed. Key annotation is used to mention that the property is a primary key in the future table. The consequence of Required annotation is that the property will have a constraint and it can not have the null value in the database. The opposite of the Required annotation is CanBeNull which allows the null value for the specific attribute. Giving other name than the C# attribute name to the database column can be done by using Column annotation on the attribute. The link in the database is done by creating foreign keys, in order to create a foreign key with Entity Framework the ForeignKey annotation is used and as parameter is passed the name of the associated property.

Back to vehicle permissions, to use this permission in application services a custom manager was designed. This custom vehicle manager has also an interface that is implemented by the manager. This interface is showed in the Figure 9, and it is injected using dependency injection into application services that needs access to vehicle permissions.

```

9 references | Stefan Cristea, 58 days ago | 1 author, 3 changes
public interface IVehiclePermissionManager
{
    /// <summary>
    /// Method <c>CheckCurrentUserPermissionAsync</c> checks if current user have specified permission
    /// </summary>
    /// <exception cref="AuthenticationException">Thrown when user not logged in</exception>
    11 references | Stefan Cristea, 61 days ago | 1 author, 1 change
    Task<bool> CheckCurrentUserPermissionAsync(long vehicleId, string vehiclePermission);
    /// <summary>
    /// Method <c>CheckPermissionAsync</c> checks if user have specified permission
    /// </summary>
    /// <exception cref="AuthenticationException">Thrown when user not logged in</exception>
    5 references | Stefan Cristea, 61 days ago | 1 author, 1 change
    Task<bool> CheckPermissionAsync(long userId, long vehicleId, string vehiclePermission);
    /// <summary>
    /// Method <c>AssignPermissionAndGetIdAsync</c> assign a permission to a role or to a user for a specific vehicle
    /// </summary>
    /// <param name="input"> If is not null the permission will be applied to userId</param>
    ///
    /// <returns>The Id of the inserted permission</returns>
    ///
    /// <exception cref="AbpException">Thrown when permission does not exist</exception>
    /// <exception cref="AbpException">Thrown when user already have this permissions</exception>
    /// <exception cref="AbpException">Thrown when both UserId and IdRole are null</exception>
    10 references | Stefan Cristea, 61 days ago | 1 author, 1 change
    Task<long> AssignPermissionAndGetIdAsync(PermissionAssign input);
    /// <summary>
    /// Method <c>CreateRoleAndGetIdAsync</c> create a new role with given permissions.
    /// </summary>
    ///
    /// <exception cref="AbpException">Thrown when one of the give permission is not defined</exception>
    /// <returns></returns>
    3 references | Stefan Cristea, 61 days ago | 1 author, 1 change
    Task<int> CreateRoleAndGetIdAsync(VehicleRole vehicleRole, long idVehicle, List<string> vehicleRolePermissions);
    /// <summary>
    /// Method <c>DeletePermissionFromRoleAsync</c> delete permission from selected role on specified vehicle
    /// </summary>
    3 references | Stefan Cristea, 61 days ago | 1 author, 1 change
    Task DeletePermissionFromRoleAsync(int idRole, long idVehicle, string permissionName);
    /// <summary>
    /// Method <c>DeletePermissionFromUserAsync</c> delete permission from selected user on specified vehicle
    /// </summary>
    2 references | Stefan Cristea, 61 days ago | 1 author, 1 change
    Task DeletePermissionFromUserAsync(int idUser, long idVehicle, string permissionName);
    /// <summary>
    /// Method <c>GetCurrentUserPermissions</c> retrieves all the permissions for current user both on role and individual
    /// </summary>
    /// <param name="idVehicle">id of the vehicle on which permission are assigned</param>
    /// <returns></returns>
    1 reference | Stefan Cristea, 61 days ago | 1 author, 1 change
    Task<IEnumerable<VehiclePermission>> GetCurrentUserPermissions(long idVehicle);
    /// <summary>
    /// Method <c>GetRolePermissions</c> retrieves all the permissions for specified role
    /// </summary>
    /// <param name="idRole">id of the role to be checked</param>
    /// <param name="idVehicle">id of the vehicle on which permission are assigned</param>
    /// <returns></returns>
    1 reference | Stefan Cristea, 61 days ago | 1 author, 1 change
    Task<IEnumerable<VehiclePermission>> GetRolePermissions(int idRole, long idVehicle);
    /// <summary>
    /// Method <c>AssignUserRoleAsync</c> assign the specified user role for the vehicle.
    /// </summary>
    /// <param name="idUser">id of the assigned user</param>
    /// <param name="roleName">name of the described role</param>
    /// <param name="idVehicle">vehicle id</param>
    /// <returns></returns>
    4 references | Stefan Cristea, 58 days ago | 1 author, 1 change
    Task AssignUserRoleAsync(long idUser, string roleName, long idVehicle);
    /// <summary>
    /// Method <c>AssignUserRoleAsync</c> assign current user the specified user role for the vehicle.
    /// </summary>
    /// <param name="idUser">id of the assigned user</param>
    /// <param name="roleName">name of the described role</param>
    /// <param name="idVehicle">vehicle id</param>
    /// <returns></returns>
    2 references | Stefan Cristea, 58 days ago | 1 author, 1 change
    Task AssignCurrentUserRoleAsync(string roleName, long idVehicle);
}

```

Figure 9 - Vehicle permission manager interface

Figure 9 also describes the methods provided by the interface and their signature. These methods signature also have its proper documentation, which scope is to help the programmer use them. Vehicle permission manager provides method for: check if the user has a permission, assign permission to user and to specific role, create role, remove permission from user and from specific role, retrieve all user's permissions for specific vehicle, retrieve role permissions and assign user to a role.

This methods implementation as the other implementations from application are done using the repository interfaces provided by Entity Framework and LINQ expressions for queries. One of the most used methods, CheckPermissionAsync is shown in Figure 10.

```
5 references | Stefan Gheata, 51 days ago | 1 author, 2 changes
public async Task<bool> CheckPermissionAsync(long userId, long vehicleId, string vehiclePermission)
{
    var idRoleOfUserPerCar = await _vehicleRoleUserRepository.GetAll()
        .Where(r => r.IdVehicle == vehicleId && r.UserId == userId) // IQueryable<VehicleRoleUser>
        .Select(r => r.IdRole) // IQueryable<int>
        .FirstOrDefaultAsync(); // Task<int>

    if (idRoleOfUserPerCar == 0)
    {
        var userPermission = await _vehiclePermissionRepository.GetAll() // IQueryable<VehiclePermission>
            .FirstOrDefaultAsync(p => p.UserId == userId && p.IdVehicle == vehicleId && p.Name.Equals(vehiclePermission)); // Task<VehiclePermission?>

        return userPermission != null;
    }

    var rolePermission = await _vehiclePermissionRepository.GetAll() // IQueryable<VehiclePermission>
        .Join(
            inner: _vehicleRoleUserRepository.GetAll(),
            outerKeySelector: p => p.IdRole,
            innerKeySelector: vp => vp.IdRole,
            resultSelector: (p, vp) => new { permission = p, userClaim = vp }
        )
        .Where(p => p.permission.IdRole == idRoleOfUserPerCar &&
            p.userClaim.IdVehicle == vehicleId &&
            p.userClaim.UserId == userId &&
            p.permission.Name.Equals(vehiclePermission)) // IQueryable<{permission, userClaim}>
        .FirstOrDefaultAsync(); // Task<{permission, userClaim}>
    return rolePermission != null;
}
```

Figure 10 - Method implementation for checking user permission

This method implementation is somehow tree structured. This tree structuring comes from the fact that firstly is checked if there is any role assigned to the user. After the role check follows the permission check. The permission check depending on the role check result is searched by user individually or for both the role assigned to user and the user himself. Then for simplicity in my implementation this method is also used in CheckCurrentUserPermissionAsync which basically has the same scope, but it checks the permission for the logged in user.

Client-side Structure

Visual part of the application as the backend part is structured in modules and basically each folder from root represents a module. There are two root modules, namely public and src, and each of them is tiled in sub-modules. Public contains basically data that is public, like localization module and assets. And the src module contain more sub-modules that are used in creating the application interface. The structure and containing modules are described in Figure 11.

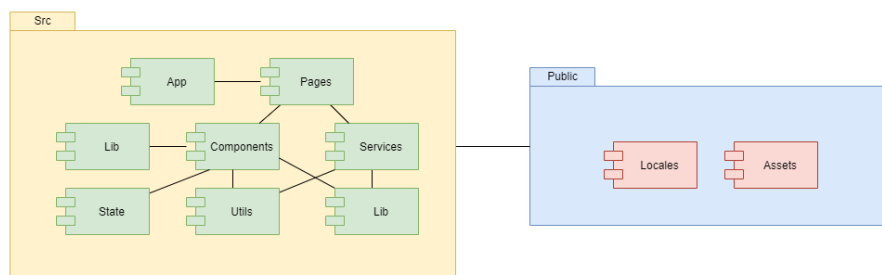


Figure 11 - Client-side application modules structure

Each sub-module has its own scope. App module is used to store the general application related features as the custom styling and routing. In components module are store the custom-made elements that are next used in pages. Intuitive, pages module stores the pages components. In the lib module covers the constants and application logic files that are needed in more than one component farther. Utils module stores the utility components as localization configuration and custom hooks. State module is reserved for redux components, namely slices, thunks and store. The linker module between frontend and backend is the services module.

Authentication context

React.JS is based on states, but states have sense only inside a component and at re-render the state comes back to its default value. Sometimes, managing state globally in entire application is needed and that is achieved using Redux. One scenario for using Redux is for managing the authentication status and current logged in user. User state stored using redux is composed of authentication status, current user object retrieved from backend, tenant (company) information also received from backend on authentication, user permissions inside application and a loading status. For accessing the application user over application and for

creating custom rendering based on authentication status a custom context is defined that wrap the application component. Figure 12 shows in the right side how AuthProvider component is used in the right side and its implementation in the left side.



```

import React, { createContext } from 'react';
import { useSelector } from 'react-redux';
import useAuth from '../utils/auth/useAuth';

const AuthContext = createContext({
  isAuthenticated: false,
  currentUser: null,
  currentTenant: null,
  isLoading: true,
  permissions: []
});

const AuthProvider = ({children}) => {
  useAuth();
  const value = useSelector((state)=>state.user.value);

  return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>
};

export {AuthProvider, AuthContext};

```

```

ReactDOM.render(
  <React.StrictMode>
    <Suspense fallback={<GlobalLoading/>}>
      <Provider store={store}>
        <CookiesProvider>
          <ChakraProvider theme={theme}>
            <AuthProvider>
              <App />
            </AuthProvider>
          </ChakraProvider>
        </CookiesProvider>
      </Provider>
    </Suspense>
  </React.StrictMode>,
  document.getElementById('root')
);

```

Figure 12 - AuthProvider implementation and usage

Context implementation uses also a custom defined hook named useAuth which scope is to check session status and to set the Redux state. Inside provider the value for user state is retrieved using useSelector hook and the value is set to context provider. Setting the user state value is done using useDispatch hook. Custom, useAuth hook implementation is showed in Figure 13.



```

function useAuth() {
  const dispatch = useDispatch();

  useEffect(() => {
    const fetch = async () => {
      var request = await getSessionInfo();
      var currentSession = request.data.result;
      if(currentSession.user){
        var userPermissionsRequest = await getCurrentUserPermissions();
        var permissions = userPermissionsRequest.data.result.items;
        dispatch(login({isAuthenticated:true, currentUser:currentSession.user, currentTenant:currentSession.tenant, isLoading:false, permissions:permissions}));
      }
      else{
        dispatch(login({isAuthenticated:false, currentUser:null, currentTenant:null, isLoading:false, permissions:[]}));
      }
    }
    fetch();
  }, [dispatch]);

  return null;
}

export default useAuth;

```

Figure 13 - useAuth custom hook implementation

Because the AuthProvider wrap the application component the authentication context can be use everywhere inside application component and its child components. Accessing the AuthContext is done by using useContext hook.

Database structure

Application persistence is ensured by a relational database, namely Microsoft SQL Database. As the name said this kind of database is defined by relations. Application database is quite complex in terms of relations. Database diagram is present in Figure 14 and covers all types of relations between two tables.

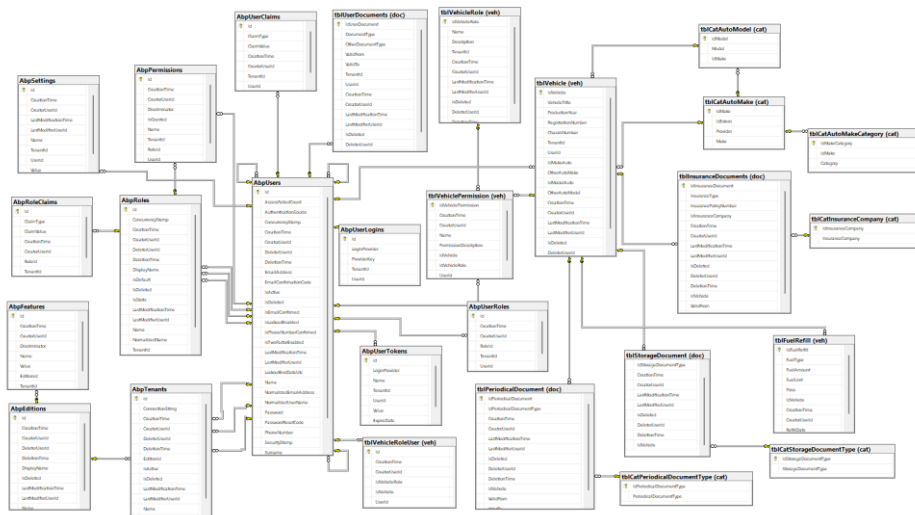


Figure 14 - Database table diagram

For a clear workflow database is design to have different schema for different application zones. That means that for application related tables the schema used is dbo, the default one, for vehicle related tables the schema chosen is the veh one and another schema, named cat, is used for catalogues.

Commented [DC1]: Ar fi bine sa adaugi si referinte car
sau articole, de ex pt partea de OCR.

References

- [1] Movcar SRL, "MOVCAR Website," 2021. [Online]. Available: <https://movcar.app/>.
- [2] PaddlePaddle, "PaddleOCR," p. <https://github.com/PaddlePaddle/PaddleOCR>.
- [3] "REST-API-Communication Figure," pp. https://www.researchgate.net/figure/The-REST-API-communication_fig3_355708737.
- [4] ".NET and .NET Core Support Policy," Microsoft, 2021. [Online]. Available: <https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core>.
- [5] "Stack Overflow insights," Stack Overflow, 2021. [Online]. Available: <https://insights.stackoverflow.com/survey/2021#most-popular-technologies-webframe>.
- [6] "Microsoft - What is .NET?," Microsoft, [Online]. Available: <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet>.
- [7] M. J. Price, "C# 10 and .NET 6 – Modern Cross-Platform Development," in *C# 10 and .NET 6 – Modern Cross-Platform Development*, 2021.
- [8] "ASP.NET Boilerplate," Volo Soft, [Online]. Available: <https://aspnetboilerplate.com/>.
- [9] "Gartner Glossary - Multitenancy," [Online]. Available: <https://www.gartner.com/en/information-technology/glossary/multitenancy>.
- [10] C. Project, "Windsor," p. <https://github.com/castleproject/Windsor/blob/master/docs/README.md>.
- [11] "Entity Framework 6," [Online]. Available: <https://entityframework.net/>.
- [12] Neoteric, "Mediu.com - Single-page application vs. multiple-page application," [Online]. Available: <https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58>.
- [13] facebook, "React repository," Meta, [Online]. Available: <https://github.com/facebook/react/releases>.
- [14] "Chakra UI," [Online]. Available: <https://chakra-ui.com/>.
- [15] PaddleOCR, "PaddleOCR - Exemple," p. https://github.com/PaddlePaddle/PaddleOCR/blob/release/2.5/doc/imgs_results/multi_lang/img_01.jpg.

Commented [SC2]: De gasit isbn

Commented [DC3]: [5] - Aici ai repetat titlul de 2 ori..

Commented [DC4]: [1] - Aici un singur titlu trebuie, nu
are doua denumiri

Commented [DC5]: Titlul nu se pune in ghilimele in
bibliografie.

