BABEŞ–BOLYAI UNIVERSITY OF CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Diploma Thesis

E-me, the secure document-handling application

Abstract

This thesis documents the development process of a document-handling application called E-me. It describes the purpose of the application, presents the general and in-detail architecture of the project and the technologies used to develop and deploy the application. The main focus of this project is to present how end-to-end encryption can be integrated into compact projects, utilizing modern encryption standards in the form of AES-256, key derivation algorithms like the Elliptic Curve Diffie-Hellman key exchange, and how these technologies can be used for user privacy and data protection within a mobile (Android) application. This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

2021 BALÁZS MÁRK

ADVISOR:
ASSIST PROF. DR. KOLUMBÁN SÁNDOR

BABEŞ–BOLYAI UNIVERSITY OF CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

Diploma Thesis

E-me, the secure document-handling application

ADVISOR:
ASSIST PROF. DR. KOLUMBÁN SÁNDOR

STUDENT:
BALÁZS MÁRK

2021

UNIVERSITATEA BABEŞ–BOLYAI, CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

E-me, aplicația de gestionare securizată a documentelor

CONDUCĂTOR ŞTIINŢIFIC:
LECTOR DR. KOLUMBÁN SÁNDOR

ABSOLVENT:
BALÁZS MÁRK

2021

BABEŞ–BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
INFORMATIKA SZAK

Szakdolgozat

E-me, a biztonságos dokumentum-kezelő alkalmazás

TÉMAVEZETŐ:

DR. KOLUMBÁN SÁNDOR,
EGYETEMI ADJUNKTUS

SZERZŐ:

BALÁZS MÁRK

2021

Contents

1. Chapter

Introduction

1.1 About E-me

E-me is a platform that serves the purpose of handling various PDF documents and user-related personal information in the form of an Android application.

The main focus of E-me is to provide quick and easy access to documents for its users that otherwise would require a physical paper format to be presented. It aims to replace the traditional physical documents while maintaining validity and data integrity. By means of this replacement, E-me not only creates a collection of a user's documents that can be accessed any time using a mobile device, but also ensures the security of these documents in order to protect its users' privacy.

The application targets people from all walks of life, since every person owns legal documents that often times are misplaced or even lost. Nowadays, the everyday life of an average person is becoming more and more filled with either work or family-related tasks and responsibilities. This increasing number of activities a person has to carry out in a day takes a great amount of effort and more importantly it consumes a substantial part of their time. When it comes to adding new obligations to this heap of duties, that may even depend on various schemes of public institutions and/or authorities, one might have to leave out or even cancel otherwise important daily projects. Acquiring new legal documents takes a great amount of time and patience, since it requires the presence of the owner of the corresponding document, not to mention the transportation and queue time.

E-me intends to tackle this problem by providing its users the ability to request documents based on predefined templates (ex. affidavit). This way a user is able to avoid the cumbersome process of preparing and verifying paperwork. In addition to saving time for a person, the application processes the personal information provided by the user in order to automatically complete the necessary fields of the documents that are being requested. This ensures that the documents created by the application are filled out using the correct information, not giving room for manual/human errors.

1.2 Similar applications in the field

As a document-handling mobile application, E-me has its fair share of competitors in the field. Although there is a staggering number of applications for document management, the feature-set and characteristics of E-me make it unique in multiple ways.

Google Docs

Being one of the most popular document-management application, Google Docs provides a wide variety of features and flexibility in terms of collaboration and document authoring. It can handle multiple types of documents such as PDF, Excel, PowerPoint, Word and many more. One of its key features is document creation and editing. Multiple users are able to edit documents at the same time while seeing each others' changes instantaneously. Google Docs also supports offline editing on multiple platforms and is able to synchronize these documents once the devices come online.

Being a Google product, Google Docs is well-integrated into other commonly used applications (ex. Google Drive) which makes it an effortless collaboration platform for easy cloud-based document access. It has a paid version called Google Workspace which provides additional security features for data protection and privacy.

Documents to Go

Documents to Go is one of the most popular document viewing apps in the market. There is a free as well as paid version available for users. The free version supports viewing file formats including Excel, Word and PowerPoint, however the editing of these files requires the purchase of the full version of the application.

The full version includes other practical features such as creating and editing PDF documents, password-protecting Word and Excel files and it also has Google Docs support, meaning a user can view and edit their files from Google Docs directly in Documents to Go. The application is available on Android as well as on Windows, providing an easy way to transfer documents across multiple devices.

SecureSafe

SecureSafe is an online storage solution. This cloud safe simplifies online file sharing and protects documents and/or passwords, offering a high level of security by means of encryption. The platform relies heavily on cryptography and offers a zero-knowledge protocol for its users. It uses double-ecryption combined with triple redundant data storage in order to prevent security breaches and/or potential data loss.

Being a cloud-hosted application, it enables encrypted file synchronization across multiple devices of a user, but also provides end-to-end encrypted file and pin sharing up to 2 GB in size to any recipients. It also has an integrated password manager and generator for highly secure pins for the online accounts of its users.

1.2.1 Comparison

Similarities

Since E-me itself is a document-handling application, much like the above mentioned applications, it also provides its users the possibility to view, create, edit or even delete their documents within the application. Similarly to *SafeSecure*, the application uses encryption to protect the privacy of its users and uses secure HTTPS-based communication to ensure data protection.

Much like the other applications, E-me has a built-in document sharing mechanism which allows for efficient and secure data transfer between devices of the same or even different users.

Differences

When compared to other file-handling applications, it is important to note that E-me only supports PDF documents. This limitation of file types derives from the fact that, in its core, the application focuses on generating and managing legal documents which are best represented as PDF files when it comes to digitalisation.

Another unique aspect of E-me is allowing the user to provide their personal information. This will be securely stored on the server and processed whenever a user requires a new document. Documents generated by the application are based on predefined templates that represent legal papers. These usually include form fields that are to be completed by the user, however E-me specializes in generating pre-filled documents that require much less or even no further modifications from the user.

Finally, sharing a document via E-me is not permanent. The recepient of the file only has a one-time token when it comes to accessing the corresponding document in a form of a QR code. This code is generated by the owner of the document and it can be interpreted via the built-in scanner of the application in order to retrieve the specified document for a short amount of time.

1.3 Summary

In the following chapters the application will be presented from various perspectives. The second chapter describes the project from a user point of view. It describes the visuals of the

application using illustrations and provides a detailed explanation of the various features and use cases of the app.

In the third and larges chapter are discussed the technical details of the application. It presents the architecture of the project using diagrams, as well as the technologies that were utilized during development. After that, it describes the implementation of some of the major layers and use cases with examples from the actual code. The last part of the chapter discusses the security features of the application, the cryptographic algorithms and methods that were used to ensure user privacy within the app.

The last chapter contains measurements about some of the cryptographic functions that are integrated in the application, furthermore it provides information about the system requirements for both the mobile and the server parts of the project. The second half of the chapter presents some of the major difficulties and obstacles that occured during the development process, as well as possible feature upgrades and expansions for the application.

2. Chapter

User documentation

Summary: *In this chapter E-me is described from a user point of view.*

The building blocks of E-me are pages. The primary role of these pages is allowing the user to manage their PDF documents that were created through the application. This management includes the ability to request a new document, view already owned documents, make modifications to them, delete the ones that are obsolete or contain incorrect data, or even share them with other devices using a QR code.

The secondary role is collecting the data from the user that is necessary for generating their documents. This data is then encrypted, processed and later on added to the documents that the user requests.

2.1 Introductory Pages

For an unauthenticated user there are 3 pages available in the application: Greeting, Login and Registration. Upon opening the application, the user is confronted with the Greeting page which allows them to navigate to the Login and Registration pages using the dedicated buttons.

The Registration page consists of five text fields. Each of these fields are required in order to create a new user, however only two of them are needed for authentication: Login name and Password. The restrictions for these fields are the following:

- The Email and Login Name fields should be unique.
- The Password and Confirm password fields should be identical.
- The Email field should be a valid email address.

Upon successful registration, the application navigates the user to the Login Page in order for them to authenticate themselves using the Login Name and Password they entered. If the authentication was successful, the main shell appears where three tabs can be seen: My Documents, Request Document and Personal Info.

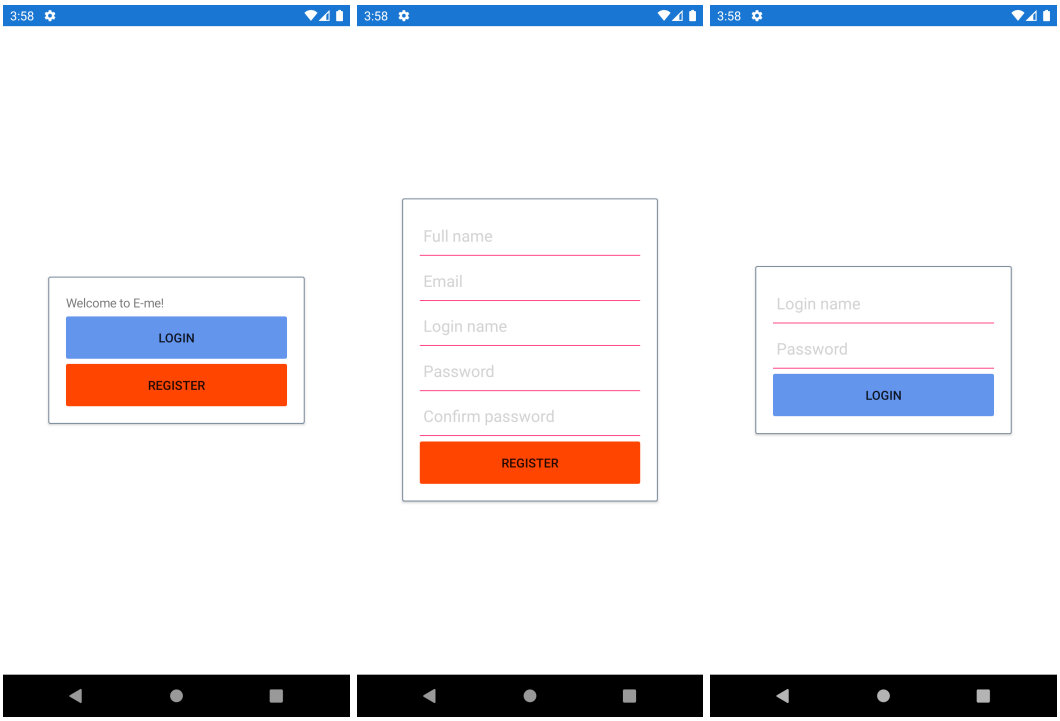

Figure 2.1: Introductory pages (Greeting, Registration, Login)

2.2 My Documents

The My Documents page consists of two main parts: the list of documents and the "Scan QR code" button. The list of documents allows the user to visualize what types of documents they own. On each document two actions can be performed: Share and Remove.

Tapping the Remove button irreversibly deletes the document from the list. After a successful removal the user is able to request a new document of the same type. If the document template was modified since the previous request or the user modified their personal information, the resulting document may be different than the previously deleted one.

The Share button allows the user to safely transfer their selected document to a different device. Upon tapping the button, the application generates a unique code for the document which is then displayed on the screen via a QR code.

The code can be read using the "Scan QR code" button. Tapping this button will attempt to open the device's main camera in order to scan the code. If this feature is accessed for the first time, a prompt appears asking for the user's permission to use the camera. If the permission is granted, the application will open the camera app. Upon successfully reading a QR code generated by E-me, the selected document will appear on the screen (see Document Viewer).

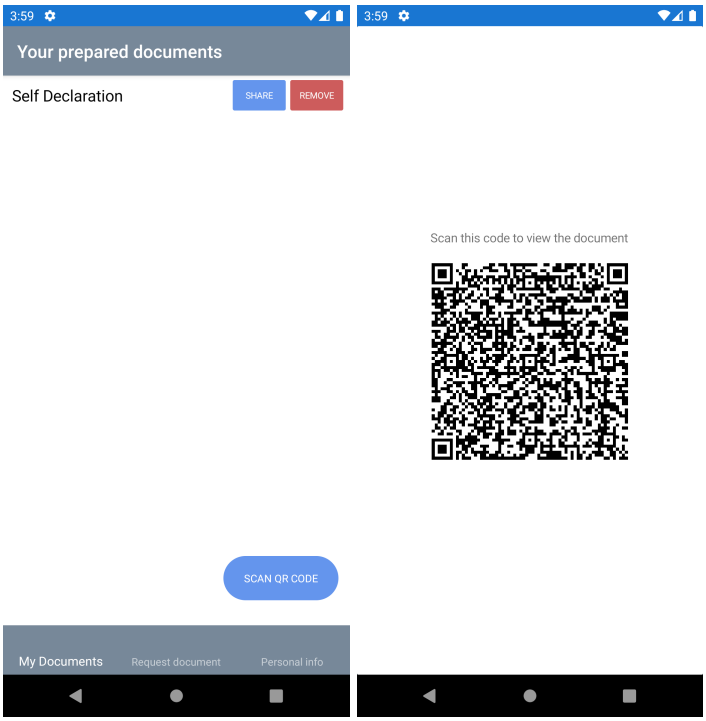

Figure 2.2: Documents and Share Document pages

2.3 Requesting a Document

The Request Document page consists of a list of document types that can be acquired by the user. The list only contains types that are not yet owned by the corresponding user. If the user acquires one of these document types, it will disappear from the list (it will be listed on the My Documents page instead).

Combined with the My Documents page, these two lists contain every document type that can be managed through the application.

The user can request a document by simply tapping on an item from the list. The requested document will be generated and automatically displayed on the screen (see Documents Page).

2.4 Personal Information

This page contains several fields containing the personal details of the user. Neither of these fields are required, however if no information is provided about the user, the application will not be able to automatically fill the requested documents. The information provided by the user on this page will be stored in an encrypted form.

Upon requesting a document, E-me attempts to match the fields of the document with the information of the user and fill them out respectively. Fields that require information that is not

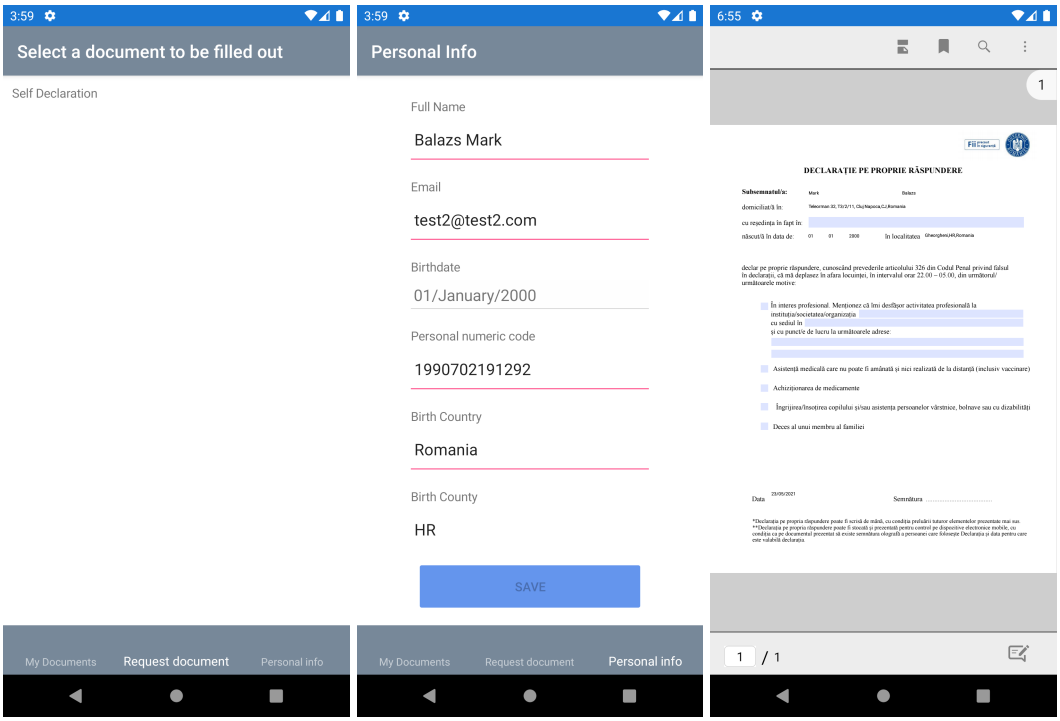

Figure 2.3: Request Document, Personal Information and Document Viewer pages

provided by the user will be left blank and can be filled manually on the Document Viewer.

2.5 Document Viewer

On this page a PDF Viewer can be seen which allows the user to inspect or even edit their documents. This viewer supports text searching, bookmarking, zooming, printing and/or saving a PDF to the local storage of the mobile device. The contents of the PDF are generated by the application, however if the document contains fields that could not be matched with any user data, the user is able to fill them out manually. This page can be closed by pressing the Back button.

Figure 2.3 contains an example of a Romanian COVID-19 self-declaration. The document consists of multiple input types, including checkboxes, dates and text fields. The majority of these inputs refer to general information about the user (name, date of birth etc.) which can be filled out automatically by the application, however some fields require time- or location-dependent data, which is not stored in E-me. In this case, these fields are left blank so that the user is able to complete them manually.

3. Chapter

Technical details

3.1 Architecture

E-me follows a commonly used N-tier architecture with three main parts: data, application (backend) and presentation (frontend) layers. Each of these tiers can be broken down into layers that are defined by their responsibilities within the application. This tier-based architectural approach adds modularity to the application which results in a low cost of change when compared to a single-tier structure.

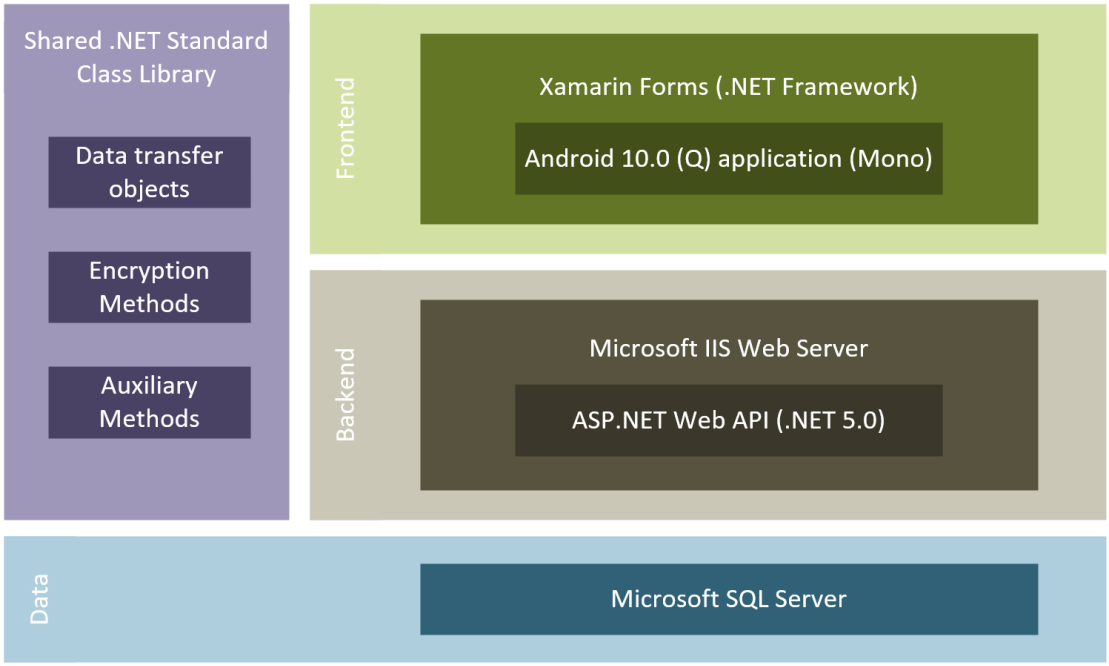

Figure 3.1: General architecture

Along with the low cost of change, the independence of the layers allows for efficient future expansion of the application by means of multiple frontend platforms (ex. web applications, desktop applications), cloud storage/services and additional Web API's that can easily be integrated into the existing application. This, combined with the high compatibility of the .NET

5.0, provides a high level of scalability and maintainability for the application.

The application and presentation layers share a common class library that contains communication-related models and auxiliary methods. This library allows both layers to benefit from the cross-platform nature of .NET 5.0, speeding up the process of development and ensuring there are no discrepancies betweeen the layers in terms of encryption and communication.

3.1.1 Application layer / Backend

The architecture of the application layer has a similar design approach to the general architecture. Consisting of 4 layers, the backend follows the single-responsibility principle in it's core. Because of it's vertical structure, each layer is dependent on one single layer that is directly below it, providing a high level of maintainability.

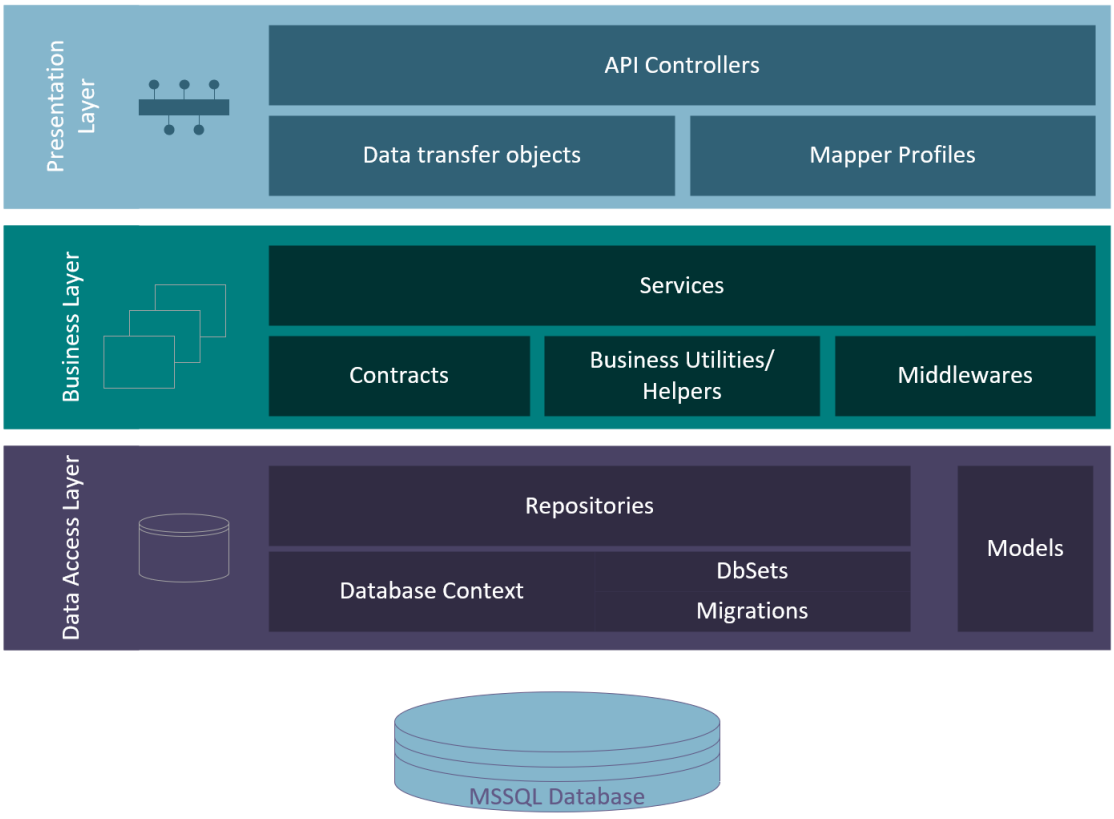

Figure 3.2: Backend architecture

Due to this abstract layout, each layer can independently be replaced or updated. This aspect enables the utilization of external and/or third-party services and components without damaging the integrity of the application.

The API structure of the presentation layer allows for a wide variety of applications or even services in which the backend can be used: desktop applications, WPF, web servers and more.

This characteristic has major role in preserving the flexibility of the application layer.

Another component being responsible for the independece and reusability of the backend is the Data Access Layer. The role of this layer is to provide the data requested by the Business Layer. By reason of abstraction, this layer is independent of the technology of the data source. This enables the utilization of a wide range of data sources, including relational (SQL) and non-relational (NoSQL) databases, cloud services and/or APIs.

Entity relations in the database layer

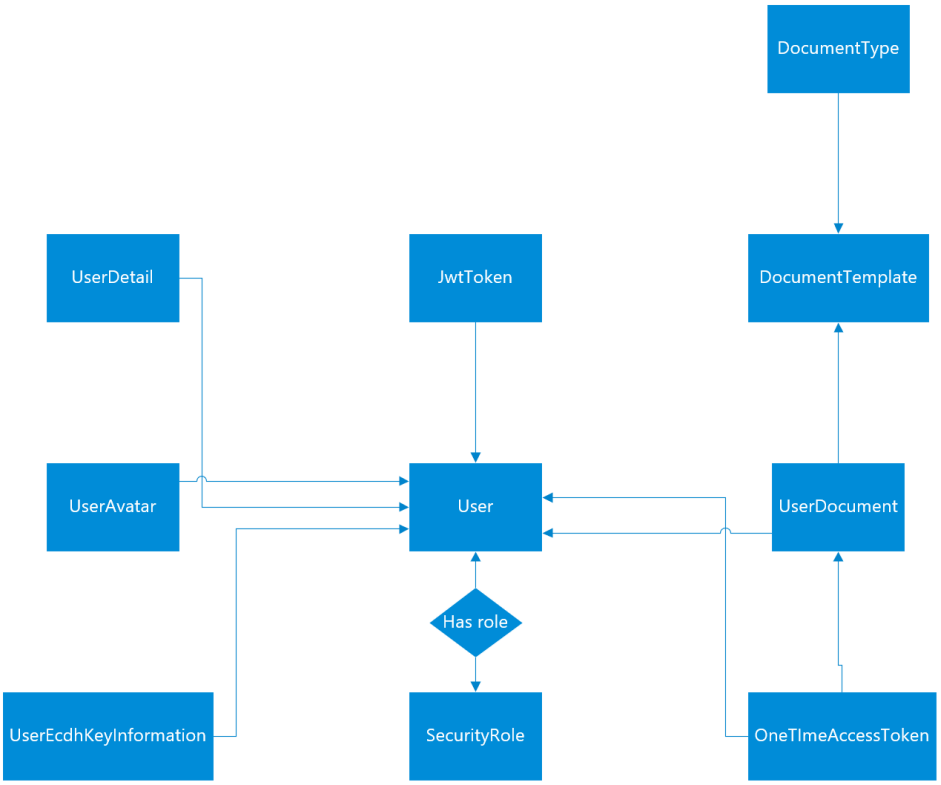

Figure 3.3: Entity-relationship diagram

3.1.2 Presentation layer / Frontend

The frontend of the application consists of three major layers that separate the user interface from the business logic and the data sources. This separation allows for easy horizontal expansion and quick feature development.

The Presentation or UI Layer contains the visuals of the application which are separated into independent pages, however it is also responsible for receiving user events and connecting them with the underlying services. Each page contains it's own event listeners and separate View Model that is responsible for making use of the Business Layer, which has a similar structure and role to the backend's Business Layer: data processing and calculations.

The main components of the Data Layer are Data Stores. These stores are functionally similar to the repositories found in the Backend of the application, although here the data is retrieved using the endpoints of the backend.

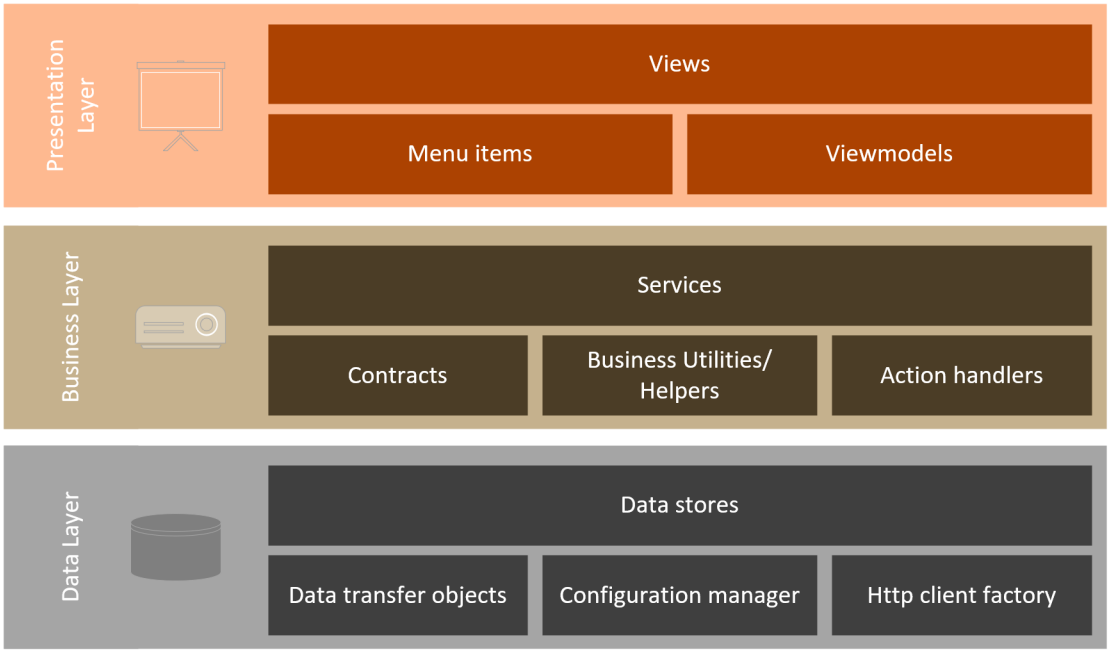

Figure 3.4: Frontend architecture

3.2 Technologies

E-me makes great use of the cross-platform nature of .NET, meaning both the frontend and the backend of the application are implemented using the same programming language: C# 9. The flexibility that is provided by the language and the wide variety of frameworks that make use of it allows for a cleaner codebase and a simpler project structure when compared to a multilingual application design.

3.2.1 Backend

In terms of frameworks, the backend relies on .NET 5.0. Being the next major release of .NET Core following 3.1, this open-source cross-platform software framework unifies a wide variety of frameworks .NET has to offer, including Windows Forms, ASP.NET Core, Azure and more. This unification made the framework a perfect candidate for the backend of an application like E-me, since it can be used on a wide variety of platforms and ensures long-term support.

E-me uses Internet Information Services (Microsoft IIS), which is an extensible web server software, as it's backend server. It allows the application to communicate through HTTPS and

provides it's separate development-time SSL certificate in order to secure the transport layer. With the *Development-time IIS support* feature, the web server allows *hot reloading*, which speeds up the development and debugging processes.

The main constituent of the backend itself is the ASP.NET Core web framework. Using it's built-in IoC, this framework allows for a high level of abstraction in order to build well-structured enterprise level applications using dependency injection [1] [2]. This framework is responsible for registering and configuring any other dependencies that may appear in the application backend.

Alongside with ASP.NET Core, one of the main components of the backend is the Entity Framework Core, which is the base of the *Data Access Layer*. In terms of building and maintaining the data access layer, E-me uses the code-first approach through *EF Migrations*. This approach provides flexibility in case of a potential database change and/or creation, but more importantly it allows the entire structure of the entities to be managed based on the models and datasets present in the code.

E-me relies heavily on encryption and security, which demands the utilization of advanced cryptographic libraries. For this purpose, the application uses the *Windows CNG (Cryptographic Next Generation) API*, which enables exchanging documents and other data in a secure environment. In terms of cryptography, this API provides a full implementation of the *Elliptic-curve Diffie-Hellman* key agreement protocol, alongside with many other cryptographic protocols and algorithms such as RSA, AES (Advanced Encryption Standard) or DSA (Digital Signature Algorithm) and hash functions like SHA-2 or MD5.

The documents handled by the application are created and managed by the *Telerik Document Processing Core* framework, which provides flexibility and efficiency in terms of document management. It enables processing the most commonly used text, PDF and spreadsheet file formats, allowing the application to create, import and export documents without relying on other external dependencies.

3.2.2 Frontend

Being a .NET-based application, E-me uses Microsoft's open-source mobile UI framework, *Xamarin Forms* as a base for it's frontend. This library allows developing native mobile applications on multiple platforms, such as iOS, Android or UWP using *.NET Standard* [3]. The multi-platform capability allows for a shared codebase and logic across all applications, which speed up the development process immensely. Xamarin uses an XML-based interface designer in order to create the visuals for the application. Alongside with the commonly used Xamarin UI components, E-me uses the *Telerik UI for Xamarin, GoogleVision API and Syncfusion* UI libraries for additional components and controls in it's user interface.

Although the frameworks used by the different layers of the application vary, these can have a shared codebase in the form of a *.NET Standard class library*, which can be used by both the backend and the frontend, allowing for closely integrated layers.

3.3 Implementation

Being cross-platform and having a wide range of possibilities in terms of future expansion and upgrades, E-me focuses heavily on scalability and maintainability. In order to achieve these characteristics, multiple factors have to be considered during development, such as project structure, dependencies and code reusability. For this purpose, both the backend and the frontend of the application relies on *Dependency Injection*. This technique allows for a service-based structure within the application and it is an elegant way of implementing loosely coupled classes.

3.3.1 Backend

The backend of E-me is separated into 5 projects that are defined by their role within the application. These projects are targeting the *.NET 5.0* framework, one exception being the *Shared* project, which is used as the common codebase for the backend and the frontend. The majority of the projects are *Class Libraries*, which are collections of pre-coded object-oriented templates and classes. These libraries revolve around the main entry-point of the application, the *MVC project*.

The MVC project

As the main entry-point of the application backend, this project is responsible for configuring and registering all dependencies and services E-me requires and utilizes. This configuration of the *IoC (inversion of control)* and the registration of dependencies take place in the *Startup* class. This class is registered to the *Host Builder* as soon as the application starts.

```
1  public static IHostBuilder CreateHostBuilder(string[] args)
   {
         return Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
6                       webBuilder.UseStartup<Startup>();
                });
   }
```

Representing the presentation layer, this project defines the *API Controllers* which are representing the endpoints of the backend. These controllers are the connection points through which the backend is able to communicate with other layers of the application. The endpoints of

these controllers follow the same route structure: *[base-address]/api/[controller-name]/[action-name]*, for example: *https://e-me.Mvc/api/auth/login*. All endpoints of the application require *HTTPS* to be accessed and send response in *JSON* format to ensure that the receiving party is able to interpret the message.

One of the most important of these controllers is the *AuthController*, which is responsible for registering, authenticating and de-authenticating the users of the application. This is the only controller that can be accessed by an unauthenticated user, since the rest of them handle more sensitive data.

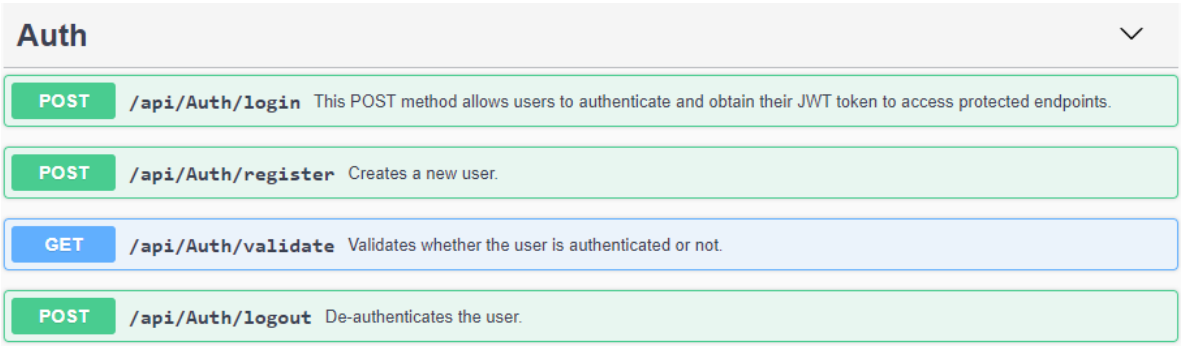

Figure 3.5: AuthController endpoints

Once a user is authenticated, they are able to access the protected endpoints. A controller's or and endpoint's protected nature is marked by the *Authorize* declarative attribute. When accessed, the endpoints marked with this attribute automatically verify the authorization header of the request and the role of the current user if necessary.

```
[Authorize]
[ApiController]
public class UserDetailsController : Controller
```

Two of the main protected controllers are the *UserDocumentsController* and the *DocumentTemplatesController*. These two combined are responsible for handling any document-related user requests, such as reading or deleting by id, getting available or owned documents etc. All of these actions contain an extra layer of validation in order to ensure that the current user is eligible for viewing, modifying or removing the specified documents or types.

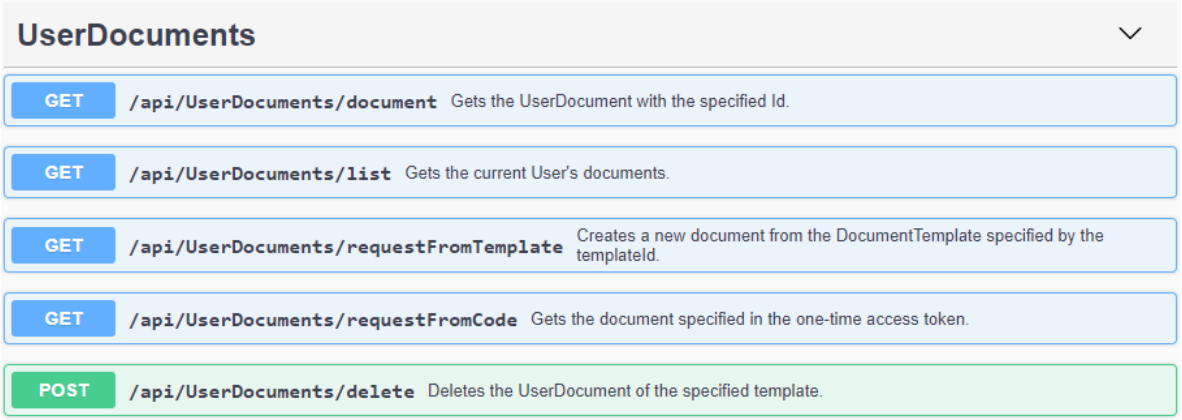

Figure 3.6: UserDocumentsController endpoints

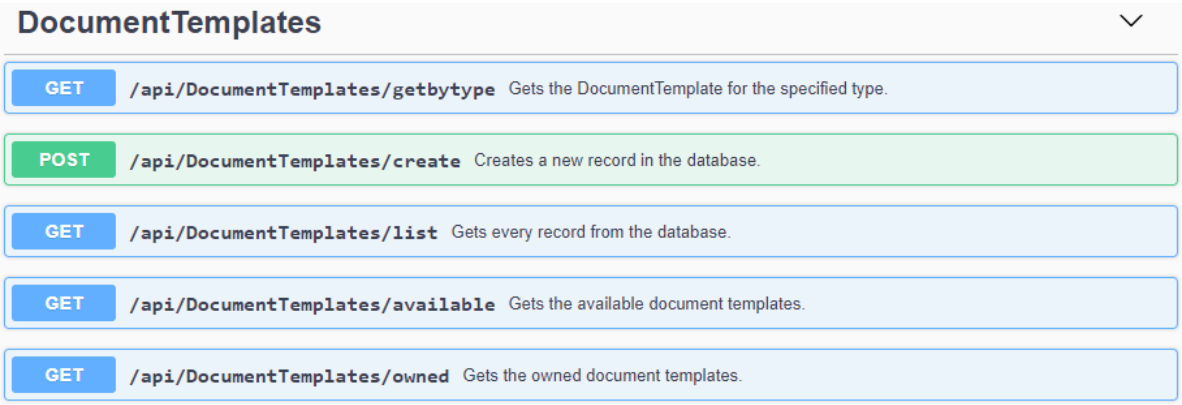

Figure 3.7: DocumentTemplatesController endpoints

Apart from these three major controllers, there are three that are significantly more compact and have much fewer endpoints: *DocumentTypesController, OneTimeAccessTokensController* and *UserDetailsController*. These three cover the non-document-related actions within the application, which include modifications of the personal information of the user, requesting a *document-sharing token* (represented as a QR code in the frontend) or listing the document types that can be managed through the application.

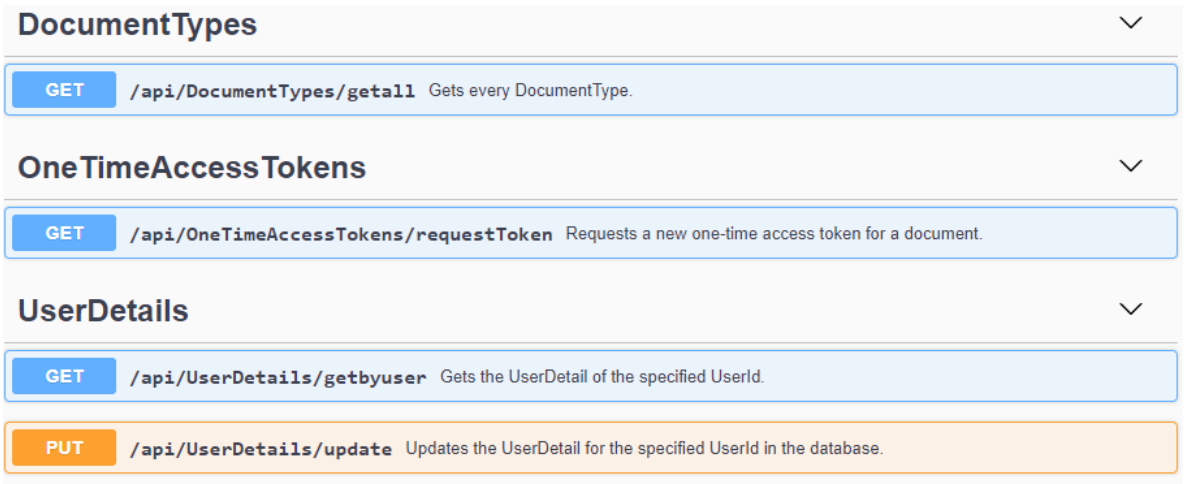

Figure 3.8: Endpoints of the minor controllers

The Business project

As the name suggests, this intermediate project represents the Business Layer of the application backend. The project itself consists of service contracts (interfaces), factories and implementations that make use of the data-access objects from the data layer. These services perform calculations, alterations and complex operations on the data provided by the underlying layer and are utilized by the above mentioned MVC project as the source of processed data. They are registered in the *Startup* class and injected in the constructors of the controllers by the *IoC container* upon accessing the endpoints of the corresponding controller:

```
public AuthController(IAuthService authService, IUserService userService)
{
    _authService = authService;
    _userService = userService;
}
```

This loose class coupling allows for multiple implementations of the services. These implementations follow the same structure (contract), but don't necessarily have matching outputs and/or dependencies. The implementation is selected either using the *factory pattern* or in the moment of registering the service to the *IoC container*. Example for a service contract:

```
public interface ITokenGeneratorService
{
    string Generate(string userName, DateTime validTo, string role);
    string GeneratePasswordResetToken(string username, DateTime validTo
        );
    string GenerateOneTimeAccessToken(Guid userDocumentId, DateTime
        validTo);
}
```

The Model project

One other major building block of the backend is the *Model* project. As the *MVC* represented the presentation layer of the application backend, the *Model* implements the data-access layer. Having said that, this project is responsible for data persistence-related operations via *Entity Framework Core*. In order to manage the data on the server, the project has to model the structure of the database in the form of classes (*Models*) and *DbSets* which represent tables and the underlying data respectively.

The framework allows for multiple directions in managing the database structure (ex. database-first or code-first), out of which E-me uses the code-first option. This kind of implementation denotes that the database is automatically generated by the framework using the pre-coded models and defined constraints. For representing various versions of the data structure that may appear during development, database migrations are used which contain the alterations that occured after the previous version of the database was generated. These migrations can be reverted and/or modified, providing flexibility when designing the data-layer and also ensuring that the structure of the database is correctly representing the model-structure from the code at all times.

The outermost layer of the Model project consists of *repositories*. These classes represent the data-access objects (*DAO*), which are used by the services in the Business Layer to retrieve raw or pre-processed data. Each data model has its own repository that allows for performing the basic operations on the corresponding table from the database. These operations include deletion, insertion and querying. Each repository is inherited from the *BaseRepository* abstract class, which implements the above mentioned basic operations:

```
public abstract class BaseRepository<TEntity> : BaseRepository,
    IBaseRepository<TEntity> where TEntity : Models.BaseModel
```

The more complex operations and non-generic queries are implemented individually in each repository depending on the needs of the corresponding service that uses the DAO. This can be seen in the following example:

```
public class OneTimeAccessTokenRepository : BaseRepository<
    OneTimeAccessToken>, IOneTimeAccessTokenRepository
{
        ...
    public async Task<OneTimeAccessToken> FindByUserDocumentIdAsync(Guid
        userDocumentId)
        {
                return await All.FirstOrDefaultAsync(p => p.UserDocumentId
                    == userDocumentId);
        }
}
public interface IOneTimeAccessTokenRepository : IBaseRepository<
    OneTimeAccessToken>
{
        Task<OneTimeAccessToken> FindByUserDocumentIdAsync(Guid
            userDocumentId);
```

```
}
```

3.3.2 Frontend

In terms of volume, the frontend of the application is substantially smaller than the backend. It consists of 3 projects, one of them being automatically generated as a result of using *Xamarin Forms*. The main entrypoint here is the *Mobile* projects which contains the majority of the frontend implementation and can be separated into 4 major building blocks: services, models, views and view models (MVVM).

The Mobile project

Being a *Xamarin Forms* project, the Mobile part of the frontend is a *.NET Standard 2.1* class library which contains the shared codebase across all the mobile platforms that are part of the application. This project serves as a template for Xamarin to generate the native code for each individual platform (Android, iOS or UWP). In case of E-me, the Android platform is targeted which denotes that the generated libraries and binaries are using *Mono runtime* in order to access all of the native Android APIs.

The setup of the project is similar to the *MVC* project from the backend of E-me, since it uses the same concepts when it comes to inversion of control and dependency injection. Similarly, a *Startup* class is used to register services, visual components (views) and view models to the *IoC container* for loose coupling.

The main component of the presentation layer is the *AppShell*. This shell is a tab-based wrapper for the views of the frontend. The shell's visibility alters depending on the currently rendered view; some views are not required to be seen among the tabs of the application shell (ex. Login view, Registration view).

The visual components/views of the frontend are refferred to as *pages*. Each page consists of two parts: an XML file representing the visuals of the page and a C# class that provides the functionality for these visuals. The various view models and services are injected into the constructors of these underlying classes:

```
public RegisterPage(RegisterViewModel registerViewModel, IUserService
    userService, IMapper mapper, INavigationService navigationService)
{
        _userService = userService;
        _mapper = mapper;
        _registerViewModel = registerViewModel;
        _navigationService = navigationService;
        InitializeComponent();
        BindingContext = registerViewModel;
}
```

The services of the frontend include *cryptographic, communicaiton and storage services*. The cryptographic service makes use of the shared key-derivation and encryption methods implemented in the *Shared* project. It is used in the decryption of documents received from the application backend and upon generating keys for the *Diffie-Hellman key exchange*(see End-to-end encryption).

Communication services are responsible for establishing connection with the backend of the application. This connection is made possible by *HttpClients* which are retrieved from a *HttpClientFactory*. The role of this factory is to create clients that already include the authorization header (if the user is authenticated) and base address of the backend server. These services create request messages based on the needs of the user and handle the interpretation of the response from the server (deserialisation).

E-me makes use of the *Android Keystore system* in order to securely store sensitive information (keys, tokens etc.) via the storage services. These services are responsable for reliable data retrieval either from the keystore or from configuration files depending on the needs of the user.

3.3.3 Use cases

The use cases in E-me can be categorised by their actors, since they don't have overlapping instances.

Unauthenticated user

As for a unauthenticated user, there are two possible actions that can be perfomed: Login and Registration. The detailed description of the login process can be found in the Security section.

The process of registration starts at the *Registration page* on the frontend. Upon submitting the registration form, a *UserRegistrationDto* object is created containing all the necessary information for creating a new user. After a successful backend validation via the *UserService*, a new record in the User table is created, followed by new records in the *UserSecurityRole* and *UserDetail* tables. If the creation of the new records was successful, the backend server sends an adequate response (200 OK) in order to signal the client about the successful registration.

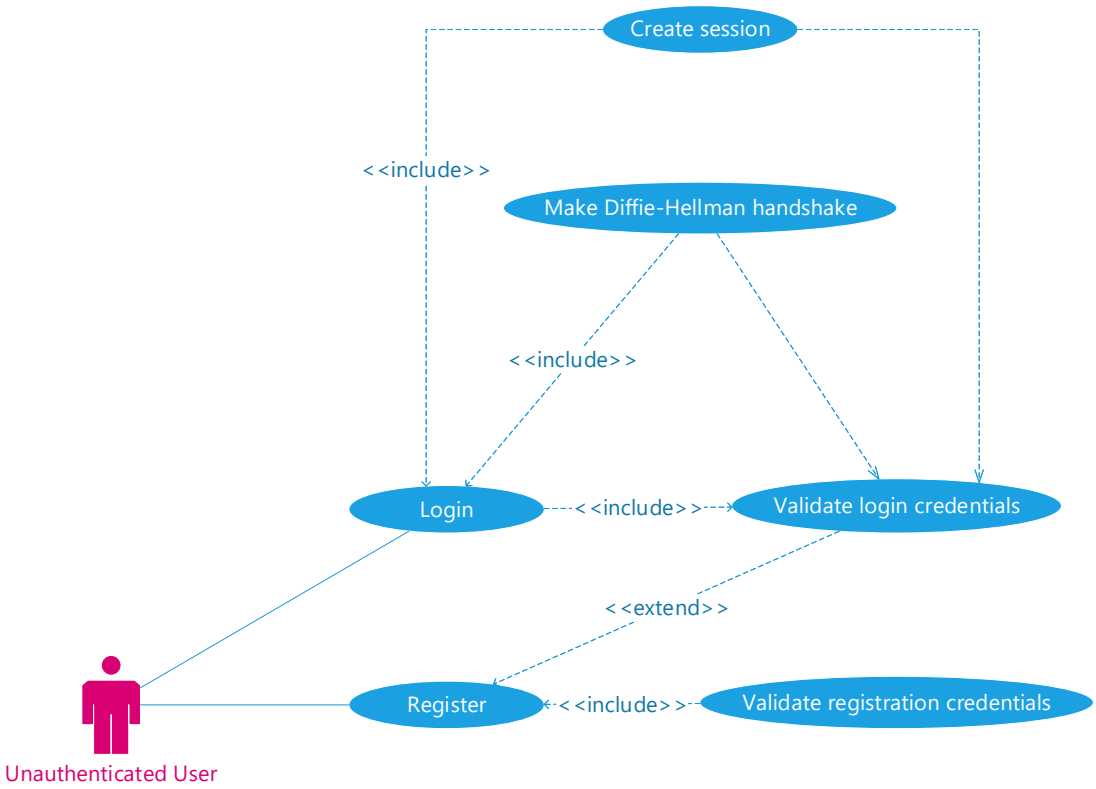

Figure 3.9: Use-case diagram (unauthenticated user)

Authenticated user

Following a successful login, a user is able to perform their document-related actions, which include viewing, sharing, deleting and/or requesting documents, and also have access to their personal information.

One of the main features of E-me is the automatic document creation using the personal details of the user. The process of requesting a document starts at selecting a document type (template) from the list of available documents. Using the *DocumentService*, the frontend proceeds to send the id (GUID) of the template to the *RequestFromTemplate* endpoint of the server. Upon receiving a valid template ID, the server loads the reuqested template document from the database. Using the *Telerik Document Processing* library, the *DocumentProcessorService* identifies the form fields of the corresponding document and looks for matching fields in the personal details of the user. Upon mapping the fields, the service proceeds to fill the ones that have matching information about the user.

```
public UserDocument GetDocumentFromTemplate(DocumentTemplate
    documentTemplate, UserDetail userDetail)
{
        var userData = userDetail.MapToDictionary();
        var radDocumentTemplate = DocumentProcessing.
            GetFixedDocumentFromBytes(documentTemplate.File);
        DocumentProcessing.FillDocumentFormFieldsByFieldNames(
            radDocumentTemplate, userData);
        var filledDocument = DocumentProcessing.GetBytesFromFixedDocument(
            radDocumentTemplate);
        return filledDocument;
}
```

Upon successfully filling the forms of the document, the newly created instance is saved in the database. After successfully creating the necessary instances in the database, the server retrieves the encryption key information about the user from the database to encrypt and hash the newly created document. If the encryption was successful, a *UserDocumentDto* is created which contains the encrypted file in *base64* format and the hash of the file to ensure data integrity and authenticity.

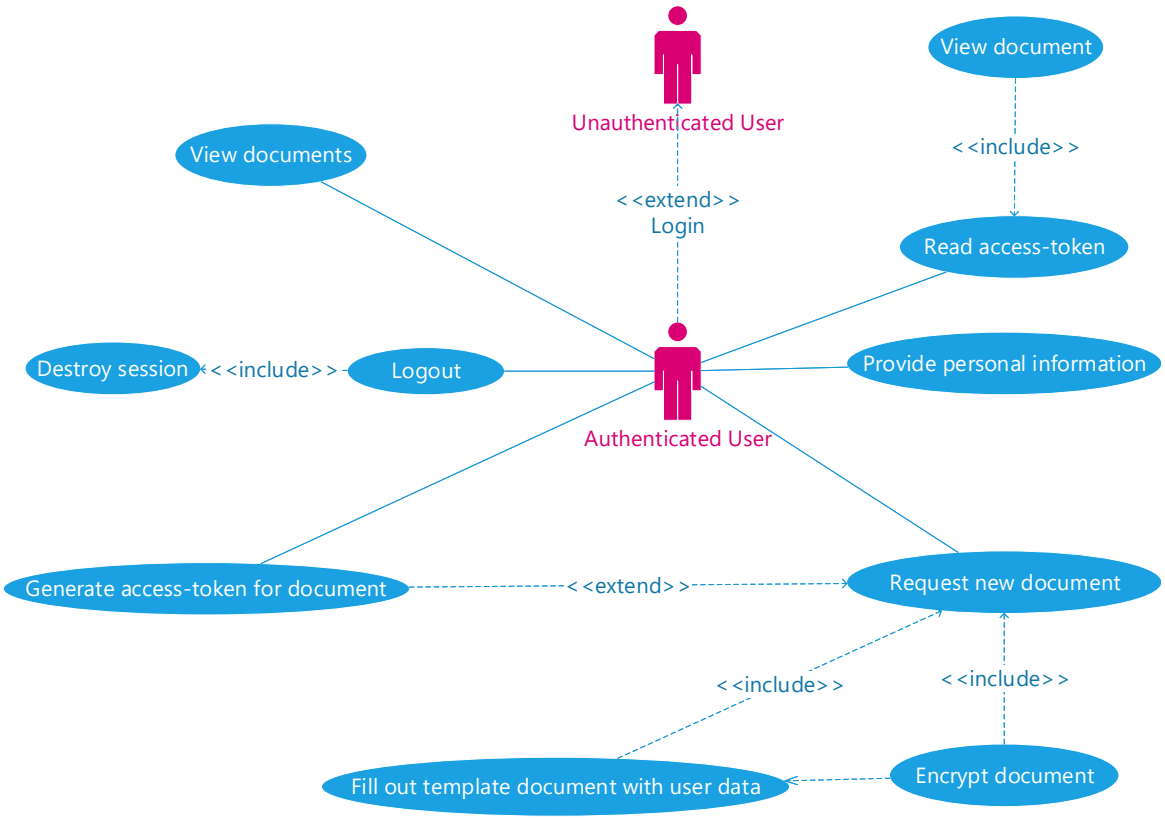

Figure 3.10: Use-case diagram (authenticated user)

After receiving the DTO from the server, the frontend client validates the hash of the document and decrypts it using the shared encryption key stored in the *Android Keystore*. If the

decryption was successful, the client presents the PDF document to the user via the *Syncfusion PDF Viewer*.

When sharing a document, the document ID is sent to the server by the *DocumentService* from the frontend. Upon receiving a correct document ID and performing validations on the user's eligibility, the backend generates a token containing information about the document which is available for *one hour* after being generated. After this token is received on the client side, it is represented as a QR code.

When reading the QR code using the built-in scanner, the mobile client retrieves the above mentioned token and sends it to the backend. In case of a valid token, the server perform an encryption and hashing of the corresponding document identical to the one performed when creating a new one. After the client receives the encrypted document, it performs the same actions as mentioned above to present the document.

3.4 Security

As a document-handling application, E-me relies heavily on secure communication, data storage and encryption. In order to achieve the required level of protection, the application uses a handful of libraries that target various layers within the backend and the frontend.

3.4.1 Data-layer security

First and foremost, E-me utilizes *Entity Framework Core Data Encryption*, which is a plugin for the above mentioned *Entity Framework Core*, to add support of encrypted fields using built-in or custom encryption providers. This library ensures the protected storage of the data via encryption. This way the data stored in the database cannot be interpreted by other parties, not even in a case of a security breach or database failure. In the case of E-me, a built-in encryption provider is used which enables *AES-256* encryption configured with a CBC (cipher-block chaining) cipher. The algorithm uses a secret key stored in the configuration of the application.

```
private AesProvider CreateAesProvider()
{
    var key = Encoding.ASCII.GetBytes(_authSettings.SecretKey);
    var provider = new AesProvider(key, CipherMode.CBC,
        PaddingMode.Zeros);
    return provider;
}
```

The data encryption library allows for independent protection of model attributes, providing flexibility for the developers in selecting sensitive fields that require encryption. This selection can greatly reduce the time consumption of the encryption and is preformed via the *Encrypted* field attribute:

```
[Encrypted]
public string FullName { get; set; }
```

3.4.2 End-to-end encryption (E2EE)

The second layer of security is located in the *Business Layer* of the application backend. One of the main features of E-me is the end-to-end encrypted (E2EE) document transfer which is achieved by this second layer. In order to successfully perform an encryption on the backend followed by a decryption on the frontend, the two sides must complete a secure key exchange beforehand. This exchange occurs upon a user's successful login to the system.

Upon receiving the user's credentials, a key pair (private and public keys) is generated on the frontend of the application. The key generation and derivation takes place in the *EcdhKeyStore* class which uses a password-based key derivation functon (*PBKDF2*) with a *SHA512* hash function as the private key generator and *X25519 Elliptic Curve Diffie-Hellman* to derive a public key from the generated private key [4].

```
public EcdhKeyStore()
{
        using var rngCsp = new RNGCryptoServiceProvider();
        var salt = new byte[SaltSize];
        var rawKey = new byte[RawKeySize];
        rngCsp.GetBytes(salt);
        rngCsp.GetBytes(rawKey);
        PrivateKey = new Rfc2898DeriveBytes(rawKey, salt, Iterations,
            HashAlgorithmName.SHA512).GetBytes(KeySize);
        PublicKey = Curve25519.GetPublicKey(PrivateKey);
}
```

The public part of the key pair is included in the login request in order to be processed on the backend server. Upon a successful verification of the login credentials, the backend's authentication service generates it's own private and public key pairs. These — combined with the client's public key — enable the derivation of a third key (shared secret) and a shared *AES Initialization Vector (IV)* which can be used for encryption and hashing in the future [5]. This key derivation process is followed by the storage of the keys and hash parameters in an encrypted form in the database [6].

```
public void SetPeerPublicKey(byte[] peerPublicKey)
{
        PeerPublicKey = peerPublicKey;
        SharedKey = Curve25519.GetSharedSecret(PrivateKey, peerPublicKey);
        HmacKey = SharedKey;
        DerivedHmacKey = Curve25519.GetSharedSecret(HmacKey, HmacKey);
        using var aesProvider = new AesCryptoServiceProvider
        {
                Key = SharedKey
        };
        aesProvider.GenerateIV();
        IV = aesProvider.IV;
```

```
}
```

The server's public key alongside with the *IV* is then included in the response of the login request. This way the client is able to generate an identical shared secret using the same methodology. In the end of the authentication process both sides will possess a secret encryption key and hash parameters despite not sharing them directly. These secret values are used whenever a document is shared between the two parties. In addition to the encryption of these documents, the application makes use of the hash-based message authentication (*HMAC*) in order to ensure data integrity and authenticity.

Transport Layer Security (TLS)

The third and most general layer of security can be found on the transport layer in the form of *HTTPS*. This layer is responsible for providing communications security over the network using a self-signed *SSL Certificate*. It enables communication bewteen parties in an encrypted form and it is used in all of the application's requests.

Authentication and authorization

The majority of E-me's endpoints require authorization to be accessed. In order to authenticate and authorize a user, the application utilizes *JSON Web Tokens (JWT)* [7]. These tokens are generated by the authentication service upon providing the correct login credentials from the application frontend and are to be included in the header of the HTTP requests by the client. These tokens contain information about the user, the generator of the token and timestamps which are used to verify a token's validity. On the server-side, the *JwtTokenValidatorMiddleware* is responsible for processing the request header and validating the incoming tokens for each request.

4. Chapter

Results and evaluation

4.1 Metrics

4.1.1 Encryption/decryption speed

Since the major features of the application revolve around data security and encryption, it is important to select the correct methodology for data protection in terms of both efficiency and security [8]. *AES-256* is a flexible, yet extremely secure option in terms of information protection. Although it is virtually impenetrable using brute-force methods, in terms of resources it is less efficient than the *DES (data encryption standard) algorithms*.

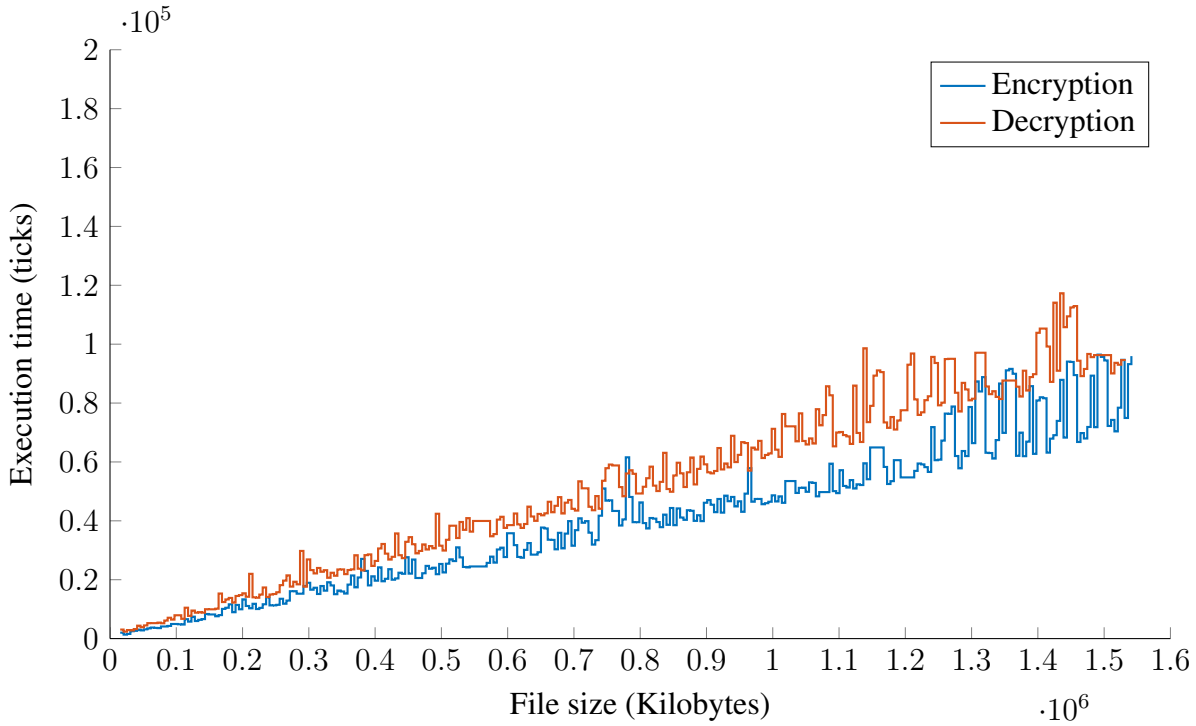

Figure 4.1: Encryption/Decryption duration over file size

As the above graph illustrates, the cryptographic operations have a linear (O(n)) time com-

plexity, the decryption being slightly more demanding in general. In real-life scenarios the majority of the documents handled by the application do not reach 10 MB in size. This denotes that generally the cryptographic operations for a single document would not cross 15 milliseconds. *Note: the measurements were taken using an Intel®Core™i7-4790K CPU @ 4.00 GHz processor and 16 GB of DDR3 RAM on a desktop PC.*

4.2 System requirements

From a hardware point of view, the application falls into the ligh-weight category, since it is not using algorithms with high time- or space-complexity. A major factor regarding the physical capabilities of the server is the size of the userbase, the scale of the application. Although the server can run on machines with fairly low performance with ease, a high number of consecutive requests can drastically influence performance or even cause server outage or failure. Taking this into consideration, the recommended specifications are estimated to be capable of handling approximately *10 000 users*.

4.2.1 Recommended hardware specifications

Web Server (*including database server*)

- CPU: Octa-Core 2.8 GHz CPU or higher
- RAM: 16 GB available RAM (DDR4) or higher
- Storage: 50 GB available hard disk space

Mobile application

- CPU: Qualcomm®Snapdragon™765 or newer
- RAM: 2 GB available RAM
- Storage: 4 GB of non-volatile storage
- Rear camera: 720p resolution or higher

4.2.2 Software requirements

Web Server

- Microsoft Windows Server 2012 or 2012 R2 operating system

- .NET 5 runtime + Hosting Bundle for IIS Support
- IIS 10.0
- Microsoft OLE DB Driver 18 for SQL Server
- Microsoft SQL Server 2019
- Microsoft SQL Server Management Studio (or optionally HeidiSQL)

Mobile application

- Android 10.0 (Q)

4.3 Obstacles and difficulties

As expected, no application development process exists without its intricacies and E-me is no exception. In the course of one year, there were multiple obstacles and difficulties that had to be tackled in order to create the current version of the application, some of them slowing down the development process by days or even weeks.

These impediments provided a great deal of knowledge when it comes to preemtive thinking and the organization of a work process that requires multiple steps and layers to complete. In my opinion, the majority of the following obstacles are easily avoidable with in-depth analysis of the technologies and processes that build up the application.

4.3.1 HTTPS on Android

When it comes to networking, the mobile operating systems are extremely strict and do not provide the same level of flexibility in terms of communication with the backend as regular web applications. This characteristic is a consequence of the closed design of the mobile environments which serves the purpose of personal data security and user protection. Although being more secure, this strict structure is a disadvantage when it comes to application development on a mobile device.

Since the backend of the application requires HTTPS to be accessed, the frontend had to follow this rule and set up its requests accordingly. In order to use HTTPS, the server needs to have a *SSL Certificate* to enable connecting devices to verify the server's identity and legitimacy. At first, E-me's backend used the *Development-time SSL certificate* provided by IIS to achieve this. However, this is not a globally valid certificate and it is to be used only in a development

environment, and when it comes to Android, it does not allow connections to these types of servers on HTTPS.

The first approach to the problem was to create a self-signed certificate either using *OpenSSL* or *IIS*. These solutions seemed reasonable for development-time support for HTTPS, however both attemps failed miserably when it came to installing the certificates on the Android Emulator. The authority (*CA*) of these newly generated certificates was not a universally accepted Certificate Authority (hence the *self* part), which means they were not accepted by Android 10.

After some research, I came across *Certbot*, which is a free, open source software tool for generating *Let's Encrypt* certificates on manually-administered websites to enable HTTPS. It is made by the *Electronic Frontier Foundation (EFF)* which is a nonprofit that stands for (amongst other) digital privacy. Using this piece of technology, I was able to request a free SSL certificate for the application which solved the issue of connecting the two major layers of the application using HTTPS.

4.3.2 Windows CNG and the Mono runtime

Since cryptography and the *Diffie-Hellman key exchange* are major components for both the backend and the frontend of the application, their implementation required a great amount of resources. For these components a separate solution was created in order to implement proofs of concept for functionalities such as key derivation, document encryption and decryption and automatic document form-filling. These proofs of concept were implemented using *.NET 5.0* which includes the implementation for the *Windows Cryptographic Next Generation API* that also provides interfaces and implementations for the *Elliptic Curve Diffie Hellman* and other key derivation methods and algorithms.

In order to use these functionalities on the frontend, these had to be implemented using *.NET Standard*, since the *Xamarin Forms* project cannot have dependencies on *.NET 5.0* libraries. This transition between the different frameworks was a smooth process without major difficulties. The *Shared* project was created, containing the cryptographic algorithms that were functioning perfectly on the backend.

When it comes to the frontend, after adding the *Shared* project as a dependency and running the generated native Android application, a runtime error occured. After some investigation, I realized that although *Xamarin Forms* allows the utilization of the *Windows CNG API*, *Mono runtime* does not have the implementation for it. This issue had no workarounds, the libraries that depended on the *Windows CNG* had to be replaced.

The first approach was to implement these cryptographic libraries, including key derivation and encryption, however this would have required an extreme amount of resources in terms of research and time of development. After days of investigation, I've discovered an open-source

implementation of the *X25519 Elliptic Curve Diffie-Hellman* by Hans Wolff. Using this implementation, combined with the cross-platform *AesCryptoServiceProvider* I was able to replace the earlier dismissed Windows-dependent library.

4.4 Possibilities for future upgrades

Since E-me is a fairly simple mobile application, there are multiple directions in which it can be improved and expanded. In terms of availability, the application can be easily implemented on multiple platforms (ex. iOS, UWP, web applications and desktop applications), since it has a cross-platform framework as its base.

As for additional features, the application has a wide variety of possible upgrades to choose from. Since E-me requires the authentication of the user, adding *biometric authentication methods* would greatly increase the practicality of the application. These authentication methods can include fingerprint scanning and face- and/or voice-recognition. This way the user would be able to skip the process of logging in when they open the application, enabling for a higher level of comfort while using it.

Being a document-handling app, a possible way of improvement for E-me is enabling digital signatures. With this feature, the users would be able to verify and sign official documents within the application. This would provide a new layer of security for the application, but also would enable handling more sensitive legal documents that would otherwise require a signed paper form.

In general, documents are dependent on already existing official papers that contain certain personal information, state contractual relationships and/or grant some rights. Representing these dependencies in the application would not only provide transparency about the structure of how legal documents are in relationship with each other, but it would provide practical features in terms of requesting documents or restricting certain document types for users. These dependencies can be treated as a directed graph (not tree, since a document can have multiple *parents*) where each node represents a document and the paths between nodes specify the dependencies themselves. A convenient way of utilizing this graph representation is requesting documents *in bulk* which would simplify the process for the user, since it allows for requesting a document and all of its dependencies while performing one single action.

In parrallel with the digital signature, adding a separate *Administration* layer through which certified professionals would be able to manage and verify documents and user information validity (ex. government), could greatly increase the potential of the application in terms of every-day use. This administrative layer, combined with the digitally signed documents, would not only mean high level of data validity and privacy, but the documents handled through the application could be accepted by institutions and authorities of the state.

In order to correctly identify and complete forms within the documents, the application requires pre-processed document templates. At the current state of E-me, these templates are to be prepared and uploaded individually. This manual processing can often be time consuming and leaves room for human error which could result in faulty documents or unnecessary data exposure. With the aim of tackling this shortcoming, the application would be able to use *Machine Learning*. The main concept of this AI-based approach would be to identify and interpret the forms of documents based on how the users of the application fill them out. E-me would learn how to complete certain documents based on real-life examples and this way it would be able to make predictions for newly added document templates. This addition would greatly reduce the time cost of maintaining the templates of the application and it would help in reducing the possibilities of human blunders.

Bibliography

[1] M. Seemann, *Dependency injection in. NET*. Manning New York, 2012.

[2] B. Joshi, "Asp. net core web api," in *Beginning Database Programming Using ASP. NET Core 3*. Springer, 2019, pp. 175–226.

[3] D. Hermes, *Xamarin mobile application development: Cross-platform c# and xamarin. forms fundamentals*. Apress, 2015.

[4] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 207–228.

[5] A. Abdullah, "Advanced encryption standard (aes) algorithm to encrypt and decrypt data," *Cryptography and Network Security*, vol. 16, 2017.

[6] B. L. Gorman, "Encryption of data," in *Practical Entity Framework*. Springer, 2020, pp. 399–448.

[7] O. Ethelbert, F. F. Moghaddam, P. Wieder, and R. Yahyapour, "A json token-based authentication and access management schema for cloud saas applications," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. IEEE, 2017, pp. 47–53.

[8] G. Singh, "A study of encryption algorithms (rsa, des, 3des and aes) for information security," *International Journal of Computer Applications*, vol. 67, no. 19, 2013.