

Management System for Tattoo Studio

Julià Medina Velasco

Resum– Crooked Tattoo BCN es un pequeño estudio de tatuaje situado en Ripollet, un pueblo en los alrededores de Barcelona. Este estudio es dirigido por dos hermanos que, por tal de adaptarse a las nuevas tecnologías y a algunos requisitos sugeridos por sus clientes, decidieron que sería beneficioso para su negocio tener un sitio web donde promocionarse. De acuerdo con sus peticiones, ésta web es de hecho un sistema de gestión basado en web para su negocio de manera que exprimir todo el potencial de su negocio y, ser capaces de comprobar el estado de su negocio. MSTS (Management System for Tattoo Studio) es el software que se desarrollará para ser utilizado en Crooked Tattoo por tal de satisfacer las necesidades y requisitos que se podrán ver a lo largo de éste documento. Éste software pretende ser compacto, genérico, robusto y escalable de manera que, en caso de ser necesario, pueda actualizarse fácilmente en un futuro.

Paraules clau– Model-vista-controlador, tatuaje, diseño, arquitectura, patrones, aplicación web, .NET, C#

Abstract– Crooked Tattoo BCN is a small tattoo studio located in Ripollet, a town nearby Barcelona. This studio is managed by two brothers who, in order to adapt to new technologies and to some requirements raised from their clients, have decided that it would be benecial for their business to have a website to promote themselves. According to their requirements, this website is indeed a Web Business Management System (WBMS) so that they are able to squeeze all the potential of their business and also, be able to have a full knowledge of the status of their business. MSTS is the software that it is going to be developed in order to be used by the Crooked Tattoo studio in order to satisfy all the requirements that will be seen in this document. This software will be compact, generic, robust and as scalable so, if needed, could be easily updated in a future.

Keywords– Model-view-controller, tattoo, design, architecture, patterns, web system, .NET, C#

1 INTRODUCTION

18 CENTURY was the time that the word **tattoo** (which is a loanword from the Samoan word *tatau*, meaning "to strike") first appeared in the Western world, thanks to James Cook, as an official name for the practice of marking one's body with ink.

We could say that the modern popularity of tattooing stems from Captain James Cook's three voyages to the South Pacific in the late 18th century. On Cook's first voyage in 1768, one of his officers, Sir Joseph Banks, as well as many others of the crew, returned to England with tattoos.

The first recorded professional tattoo artist in the United

States was a German immigrant, *Martin Hildebrandt*. He opened a shop in New York City in 1846 and quickly became popular during the American Civil War among soldiers and sailors of both Union and Confederate militaries. It was from this moment that tattoo artists began to open their own studios and creating different kind of styles as for example:

- Old School
- New Traditional
- Dotworck
- Ornamental
- Stencil
- Black & Grey

• Contact E-mail: julia.medina@e-campus.uab.cat
 • Specialization realized: Software Engineering
 • Thesis tutored by: Yolanda Benítez Fernández (departament de Ciències de la Computació)
 • Course 2018/19

From that moment, the art of tattooing began to be more popular among the people and more and more tattoo studios began to be opened around Europe, being now a place where are the most tattooed cities in the world, and being a tattoo artist began to be considered as a job.

So, it is here where Crooked Tattoo, a local tattoo studio placed nearby Barcelona, intervenes with the proposal of, through the use of software engineering, carrying out a project in which a web-based management system needs to be designed and developed in order to be able to have a better management of their tattoo studio and also, be able to adapt better to the new technologies.

The execution of this project pretends to help the Crooked Tattoo studio to promote their work among Internet and also, allow them to exercise a better control over their tattoo studio (as for example to check the information about appointments when needed by looking at their "admin panel", etc.)

All being said, this project has as its final goal the development of a scalable software that will be used to make some tattoo artists job a little bit easier and providing them the opportunity of managing their business in an easier way than they are doing today.

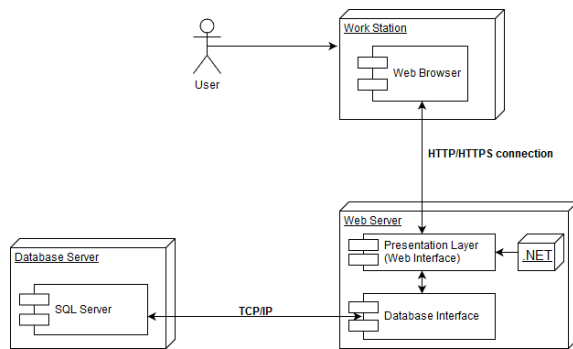


Fig. 1: Deployment Diagram

2 STATE OF THE ART

Developing this web-based application is not something new, the following references are some examples [1], furthermore, this type of tool already has a name *Web Business Management System (WBMS)* it is a software developed to help the owners to run their website with a business mindset. So, starting from the basis that this already exists, why do I develop it again from zero?

Well, the answer is that, from my point of view, this is something really "personal" depending on the business that will use it (different clients, different uses, etc.). For that reason, I preferred to start from scratch so that this software is not just one of those that already exist, but one that is specially designed for the type of business of the Stakeholder, a tattoo studio in this case. In this way both the design of the front-end part and the entire back office part will be focused on the type of users that will use both the web and the business control system, thus offering the best possible experience.

Apart from the aforementioned, another reason to carry out the project from scratch is that in this way it will be easier to fulfill all the requirements obtained from the Stakeholder and to ensure that all parts of the project are unified in the correct way and that work correctly.

3 PROJECT PROPOSAL

Crooked Tattoo BCN is a small tattoo studio located in Ripollet, a town nearby Barcelona. This studio is managed by two brothers who, in order to adapt to new technologies and to some requirements raised from their clients, have decided that it would be beneficial for their business to have a website to promote themselves (this was their main idea, only a web site).

After having some conversations, we could see that, to be able to squeeze the potential of your business and help them to manage it in a more efficient and comfortable way, it would be better to carry out the development of a web-based management system. This is how the web-based Management System for their Tattoo Studio (**MSTS**) was born.

The **MSTS** is the software that will be designed and developed to have a compact, generic and robust solution for the issues that their business do have. This system will need to accomplish, according to the studio managers, the following requirements:

- Forms to request an appointment.
- To homogenize the design for the different devices (mobile, tablet, web).
- Different user roles with different permissions.
- Being able to check the amount of benefits, appointments performed each month and the new users registered.

In order to facilitate the fact, if necessary, of applying changes in the future this **MSTS** software, will be needed to be developed as scalable as possible in order to easily identify the different parts of the system and the whole process must be well documented so that it can be understood. In other words, formalize the entire development process just in case something is needed in a future.

3.1 Project Objectives

To carry out the whole process, the following objectives are considered as those that will allow to cover said project:

Under the nomenclature **MSTS.OBJ.X.Y** the detailed explanation of each objective is as following:

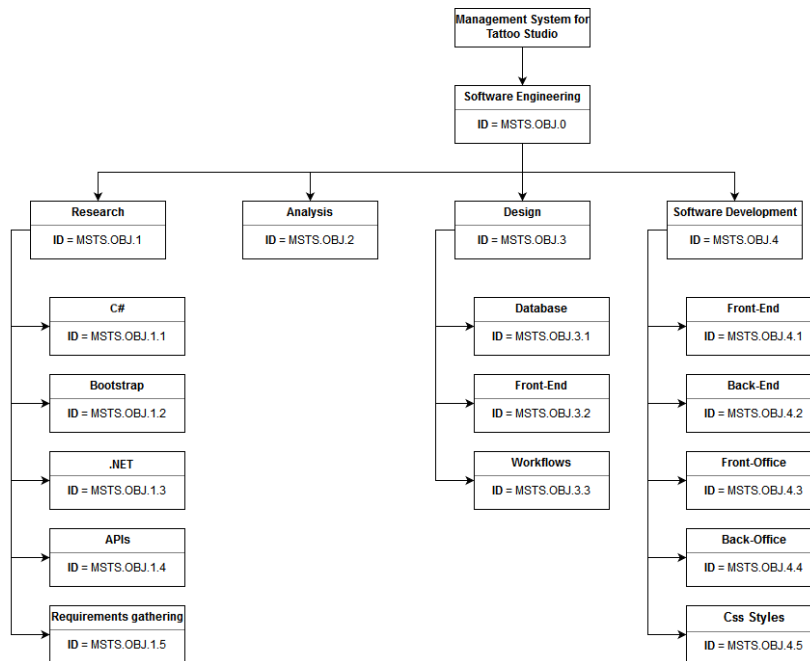


Fig. 2: Objectives Tree

- **MSTS.OBJ.0** – Software Engineering: consists of developing software engineering and apply it to the project in all its facets, starting from an elicitation, or requirements gathering , going through the previous study and analysis, arriving at the development of the project and finally, the validation of the software developed. Starting from this base, the following main objectives appear:
- **MSTS.OBJ.1** – Research: Conduct an initial study to know the environment that will fit the best to execute the project, language, frameworks, etc. To do this, a phase of information gathering on each part must be executed individually, giving rise to the parts shown below:
- **MSTS.OBJ.1.1** – C#: General object-oriented programming language for networking and especially suitable for Web development. With C# it is possible to process the information of forms, generate pages with dynamic content, or send and receive cookies, among many more things, which is really helpful as the MSTs will be a dynamic web application.
- **MSTS.OBJ.1.2** – Bootstrap: Is a free and open-source framework for designing websites and web applications. It contains HTML based design templates for typography, forms, buttons, navigation and others. These provide a modern appearance for formatting text, tables and form elements. This will be extremely useful to develop the front-end.
- **MSTS.OBJ.1.3** – .NET: The .Net framework is a collection of programming support for what are known as Web services, the ability to use the Web rather than our own computer for various services. It can be used to create both - **Form-based** and **Web-based** applications. This framework has a basic architecture that is shown below:

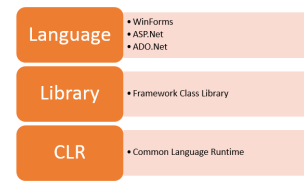


Fig. 3: .net framework architecture diagram

- **CLR**: The *Common Language Infrastructure* or CLI is a platform on which the .Net programs are executed. It has the following key features:
 - * Exception Handling - Exceptions are errors which occur when the application is executed.
 - * Garbage Collection - Garbage collection is the process of removing unwanted resources when they are no longer required.
 - * Working with Various programming languages
- **Class Library**: The .NET Framework includes a set of standard class libraries. A class library is a collection of methods and functions that can be used for the core purpose.
- **Languages**: The types of applications that can be built in the .Net framework is classified broadly into the following categories:
 - * ASP.Net – This is used for developing web-based applications, which are made to run on any browser such as Internet Explorer, Chrome or Firefox.
 - * ADO.Net – This technology is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server.

- **MSTS.OBJ.1.4** – APIs: An application programming interface (API) is a set of subroutine definitions, communication protocols, and tools for building software. Within this project, the use of APIs will be needed in order to proceed with a complete & easy solution.
- **MSTS.OBJ.1.5** – Requirements Gathering: In order to have a global vision of the desired final system, a requirement gathering must be carried out with the Stakeholder to understand the needs that impelled them to request said system and, to discover the requirements for the system, the scope and the restrictions of the system.
- **MSTS.OBJ.2** – Analysis: A correct realization of the reason of being of this project is carried out and it must be recorded in a detailed documentation the point from which this project starts, the desired scope as well as how the project's execution is going to be managed.
- **MSTS.OBJ.3** – Design: This objective focuses on how the software that implements the project will be developed.
- **MSTS.OBJ.3.1** – Database: The database (**DB**) is one of the most important parts of the MSTS since having an organized and well-structured database is key for this project as the web application will have different sections and, we will need to store in an orderly manner all the necessary information.
In order to create a structured DB, an Entity-relationship model has been created to specify there all the parts of the DB as well as the relations between the tables.
- **MSTS.OBJ.3.2** – Front-End: To develop the front-end part of the web application, it will first be necessary to design it and have it approved by the Stakeholder. The design of this part will be based mainly on offering the best User Experience possible.
- **MSTS.OBJ.4** – Software Development: Based on the objective *MSTS.OBJ.3* this objective, must comply with the above information. In this way, the development of the code will be carried out respecting the optimal procedure for any software engineering project as well as respecting the guidelines and the documentation established to carry it out.
- **MSTS.OBJ.4.1** – Front-End: Development of the necessary software to, based on the designs, to create all the views that will be displayed in the application.
- **MSTS.OBJ.4.2** – Back-End: Development of the necessary software to, based on the work-flows designs, provide internal logic to all the processes of the web application.
- **MSTS.OBJ.4.3** – Back-Office: Development of the necessary software to provide functionality for internal operations. It will be developed as easy to use as possible so that the administrator will be able to apply any change in an intuitive manner.
- **MSTS.OBJ.4.4** – Css styles: Development of the necessary software to create all the styles that will be used on all the views.

4 WORKING METHODOLOGY

The methodology that has been followed to carry out this project, has been an agile working methodology known as SCRUM. But, it is not pure SCRUM, since this methodology is more focused on carrying out group work instead of individual work, but an adaptation of it as this project is being carried out just by one person. That means that myself is the one who will be planning the activities, the sprints and the meetings with the Stakeholders (the Tattoo studio owners and my tutor).

SCRUM consists on planning out small iterations of time in which objectives are defined and work is carried out in order to meet them. To achieve these objectives, a daily monitoring of the work done the previous day is carried out so that, if necessary, changes can be applied in the planning and specify the tasks that will be carried out throughout the day. Said iterations of time, within the scope of SCRUM, are called Sprints, which, as we have mentioned before, is the period in which the work itself is executed. Using this methodology, I plan 2 weeks sprints, that could change if needed to 4 weeks sprints, so that it coincide with the meetings I hold with my project tutor.

In order to have a good planning I use the tool "Trello" that allows establishing a list with different tasks to be carried out and distributing them between tasks "in progress" or "finished". So we have a simple way to visualize the tasks that remain to be done, those that have already been done or the ones that are being carried out. It is ideal to keep a good control of the work and also fits very well to the needs proposed by this project.

5 PLANNING

In order to perform this project from beginning to end, a planning has been prepared which mentions the tasks that must be completed at all times. These tasks have been divided between activities and sub-activities that together make up this project. To carry out this planning, a Gantt Chart has been created where these activities are established, the time dedicated to each activity and the relationship between them. Apart from this, the deadlines established in the project itself have been taken into account. Those deadlines are the following:

- **Nov.11th 2018** – First progress report.
- **Dec.23rd 2018** – Second progress report.
- **Jan.27th 2018** – Final report proposal.
- **Feb.10th 2018** – Presentation proposal.
- **Feb.11th 2018** – Final delivery.
- **Feb.17th 2018** – Poster delivery.
- **Feb.18th to 21st 2018** – Defense of the project .

Broadly speaking, the Gantt Chart consists of three main blocks which encompasses almost all the activities. These blocks are:

- **Research:** Carry out a preliminary study on the project to look for other similar applications and websites and how investigate how were realized, in order to have a preliminary information and, be able to prepare a planning the realization of the project.
- **Design:** Realize an initial design based on the desirable properties of all software and the restrictions set by the Stakeholder, such as the execution of a modular and scalable software.
- **Development:** Create the software, by following the designs that had been already done.

There are three activities that are not covered by the blocks mentioned before as they cover the whole project. Those activities are: **Management System for tattoo studio**(the project itself), **Software Engineering**(the process used to develop the project) & **Documentation**.

If it is desired to take a look at this Gantt Chart, it can be found in the **11 Section**

6 DEVELOPMENT CYCLE

6.1 Requirements Gathering

Before giving way to the design and implementation stages, it is necessary to carry out a preliminary and exhaustive analysis to determine the requirements that the project must meet. With this stage it is possible to reflect with clarity and precision the different characteristics of the system (requirements) classified between functional requirements (what the system must include) and non-functional requirements (system constraints, performance requirements, etc.).

For this, a Software Requirements Specification document has been made with which it is tried to obtain a better understanding of the problem and the specific aspects that must be developed to solve said problem as well as the main functionalities to develop and what is it expected for the software product to do and not do.. This document specifies the requirements that the software must comply with, as well as a priority and impact of each requirement.

6.2 Design

Talking about the design, it is possible to distinguish three main parts:

- Model Design
- Front-End Design
- Application logic Design

6.2.1 Model Design

The model was created from the requirements gathering stage since, within them it is possible to understand the different parts that will form the system. The model is the most important part of this project, since it is the extraction of real life to an application so that we are managing to represent a physical store and the management of its business logic in a web application.

The model consists of two different types of main entities:

- **Main entities:** These entities represent items that are present in the same physical store, such as users, products, appointments, etc. That is, are those tangible things that we can extract from a real business model to our web model.
- **Support entities:** These entities represent items of relationship, grouping, distinction, etc. that help to make sense of the main entities. For example: a table that relates users to their respective roles. In real life we know that a user is a tattoo artist, but when interpreting this information in an application we need this type of "supplement" to be able to discern the artists among the rest of users.

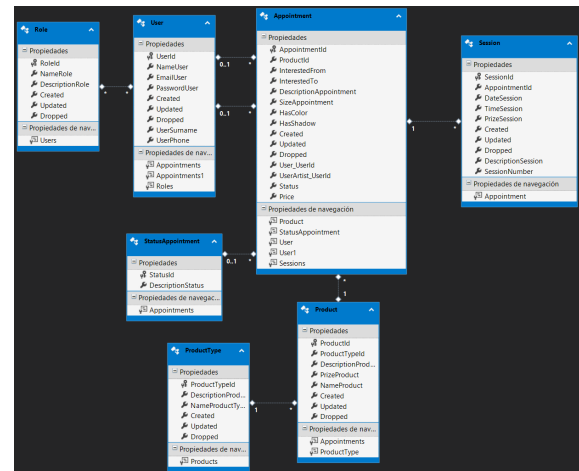


Fig. 4: E-R Diagram

6.2.2 Front-End Design

With the design of the visual interface of the application, some requirements came up when conversing with the Stakeholders:

- The interface must be responsive
- The design shall be striking but clean, without too many things on it
- The interface shall be intuitive and easy to use

In order to create the design I worked side by side with the Stakeholders taking profit of their ideas and artistic skills. From that point we all started to do some freehanded draws on a blank paper and that way I captured their main idea of the design and also, I commented on some of the defects that the designs could have and made several recommendations until finally we got a design that fitted all their requirements and seemed to please everyone. Once this was finished, I took a couple of days to study this design until our next meeting within two days where I commented them some problems that might appear and finally, we ended up with a final-design that was approved by the Stakeholders and that was realistic at the same time that fitted the requirements.

6.2.3 Application logic Design

This type of design refers to the usability and the flow of information that allow to carry out the use of the application. In this, all the functionalities of the actors that intervene in the application are defined.

Below are the functions that make up the system, depending on the type of user that is using the web application.

Anonymous User

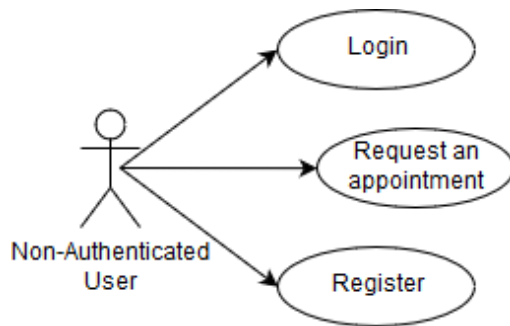


Fig. 5: Anonymous User Use Case

Registered User

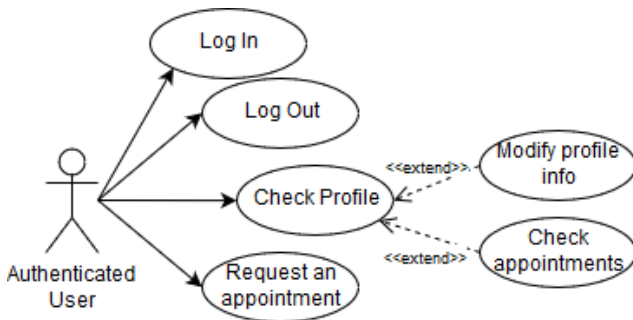


Fig. 6: Registered User Use Case

Artists

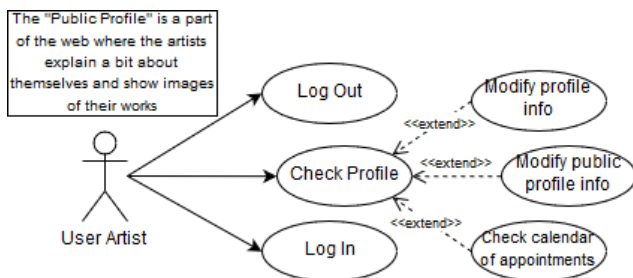


Fig. 7: Artist Use Case

Administrator

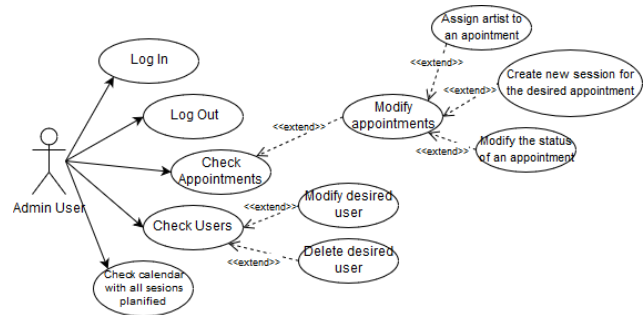


Fig. 8: Administrator User Case

6.3 Implementation

The implementation of the project has been performed by following the flow explained below: First of all, the model was created in a database hosted on a local SQL Server. To translate the Transact-SQL code and adapt it to the model, I have opted to use the Entity Framework (EF) that is offered by Microsoft[2]. This tool is an Object-Relational-Mapping (ORM), which allows us to relate a model of persistence data with our model at a higher level in our application using instances of our classes. That is, it allows us to relate the data model of a database with the data model in C# classes. This method of work is more commonly known as Database First, that is to say, from a database model, the classes of the C# model are automatically generated and already internally related to each other.

From this point is where the ViewModels appear [3]. As the name indicates, they are in charge of relating necessary information from the model and the view, in such a way that they will pick up what they need from the fields of the model to show it in the view to the user. It represents the data that it is wanted to be displayed on a view/page, whether it will be used for static text or for input values (like textboxes and dropdown lists) that can be added to the database (or edited). A ViewModel can have two main modification flows:

- It can be modified in the view by the user through a form and end in the data model in a nagged manner.
- It can be populated in the data access layer (DAO / Generic Repository (the explanation of this pattern can be found in **Section 7**) and shown informatively in the view.

The development process has been systematic for each of the functionalities or use cases previously explained.

1. Create the ViewModels by defining the "fields" that will be necessary to obtain from the users. When doing this, in order to provide some "logic" to those forms, data annotations have been used in order to manage the data definition in a single place and do not need rewrite the same rules in multiple places. Those tags are helpful to enforce validation rules, specify how data from a class /member is displayed in the UI and specify the intended use of class member and the relationship between classes.

2. Create the way to retrieve / modify / insert the information in the data access layer (DAO / Generic Repository).
3. Create the view without logic, just the items that are desired to be shown and used in a future. Taking profit from the data annotations used in the ViewModels, here, in the views, the *Unobtrusive Validation* has been used. This type of validation uses HTML5-compatible "data-" attributes to store all of the information it needs to perform validation so, it is possible to implement simple client-side validation without writing a ton of validation code, and that the user experience can be improved simply by adding the appropriate attributes and including the appropriate script files.
4. Develop the business logic in the controller. For example, in order to create a new session for a tattoo, the following steps are done:
 - First, the appointment under which a new session is desired to be created is retrieved.
 - A new session is created and related to the appointment by using the unique Id of it.
 - After that, each property of the new session is populated with the data received from the view in the ViewModel.
 - The, it is needed to fill out the audit or historical fields(date of registration, modification).
 - Finally, the ORM is used to persist the information in the Database.
5. Knot the information following the MVC data flow.
6. Test at a functional level the possible error cases (eg: null values, incorrect information, etc.)

7 DESIGN PATTERNS

- **Data Acces Object (DAO):** [4] is a design pattern that has the purpose of separating the access to the tables of the DB in a decoupled way from the rest of the app. Basically it allows us to have a class that does not depend on anything other than itself and retrieves the information from the database.

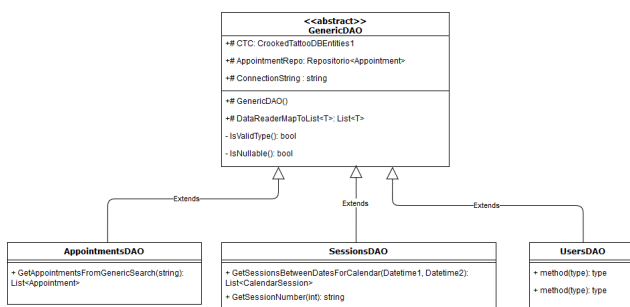


Fig. 9: DAO architecture

As possible to see in the class diagram of the figure 9, simply instantiating a class **SessionsDAO**, it is possible to access to any type of information from

the database without worrying about the connection, the context, and other configuration methods that are being already inherited from GenericDAO.

To apply this pattern correctly, the methods that appear in each class will be to extract information related to the class in which they are found, for example:

if at any point it is wanted to obtain the number of registered users in 2019, we would add a new method "GetRegisteredUsers (int year)" to the Users-DAO class. In this way, we are encapsulating and decoupling information by layers and by models.

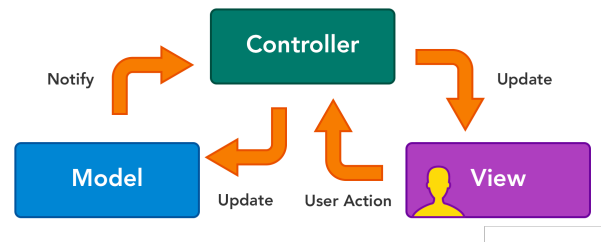
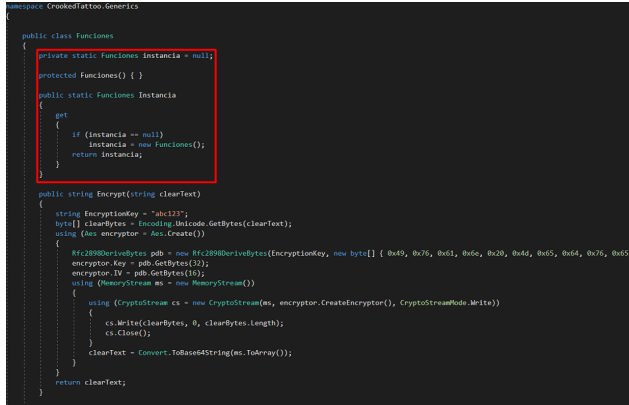


Fig. 10: Model-View-Controller Pattern

- **Model - View - Controller (MVC):** This pattern is one of the most popular in the world of web development. It is based on 3 main pillars:
 - **Model:** It contains the classes that represent the model of a real world of flat objects. These objects would be users, tattoo sessions, products, etc. In addition to them, it will also contain classes to help the real model to understand it in a computerized way, for example, the type of product offered, the user role , etc.
 - **View:** It is the visual representation of the data, everything that has to do with the graphical interface goes here. It only has to contain visual information to be able to interact with the user, therefore, it will never apply business logic, but logic of user experience.
 - **Controller:** The controller is responsible for managing the flow of information between the view (user) and the model (classes and BD). In this way, the controller will receive information generated by the user, it will interpret it and will tell what to do with it: check errors, redirect to another view, call a method of data insertion in BD, etc.
 - **Singleton:** A singleton is a class that allows only a single instance of itself to be created and gives access to that created instance. It contains static variables that can accommodate unique and private instances of itself. It is used in scenarios when a user wants to restrict instantiation of a class to only one object. This is helpful usually when a single object is required to coordinate actions across a system.
- The reason for using this pattern in the application is to avoid that methods that never vary and that are general

throughout the application are executed in different instances. In this way we get, for example, that there is not a different instance to encrypt a password for each connected user, but that there will be a single instance in the entire application for all users. It is a good method to optimize the performance of the application in terms of memory.



```

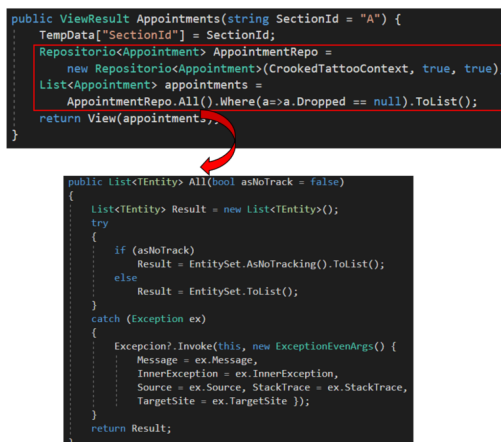
public class Funciones
{
    private static Funciones instancia = null;
    protected Funciones() { }
    public static Funciones Instancia
    {
        get
        {
            if (instancia == null)
            {
                instancia = new Funciones();
            }
            return instancia;
        }
    }

    public string Encrypt(string clearText)
    {
        string EncryptionKey = "abc123";
        byte[] clearBytes = Encoding.Unicode.GetBytes(clearText);
        using (Aes encryptor = Aes.Create())
        {
            Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, new byte[] { 0x49, 0x76, 0x01, 0x0c, 0x20, 0x0d, 0x05, 0x04, 0x76, 0x05, 0x01, 0x0c, 0x20, 0x0d, 0x05, 0x04 });
            encryptor.Key = pdb.GetBytes(32);
            encryptor.IV = pdb.GetBytes(16);
            using (MemoryStream ms = new MemoryStream())
            {
                using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateEncryptor(), CryptoStreamMode.Write))
                {
                    cs.Write(clearBytes, 0, clearBytes.Length);
                    cs.Close();
                }
                clearText = Convert.ToBase64String(ms.ToArray());
            }
        }
        return clearText;
    }
}

```

Fig. 11: Singleton Pattern

- Factory Pattern:** This pattern allows us to have a "factory" of objects. In the project there is an interface called "IRepository" that offers a CRUD functionality via Entity Framework to the respective DBSets. A class "Repository" has been implemented as a "factory" that implements said interface. An adaptation of this pattern has been used to avoid having so many classes that implement the main interface as model classes. In order to do this, an EntitySet is defined within the class that admits as a type of input a model of our data model. In this way we allow any model mapped to the BD to be accessed, modified, retrieved, and eliminated through a same class, simply indicating to it the model type when instantiating it and calling the method that is necessary. So, if we want, for example, to recover all the current Appointments, we would only need two lines of code as we can see in **Figure 12**.



```

public IActionResult Appointments(string SectionId = "A") {
    TempData["SectionId"] = SectionId;
    Repository<Appointment> AppointmentRepo =
        new Repository<Appointment>(CrookedTattooContext, true, true);
    List<Appointment> appointments =
        AppointmentRepo.All().Where(a => a.Dropped == null).ToList();
    return View(appointments);
}

public List<Entity> All(bool asNoTrack = false)
{
    List<Entity> Result = new List<Entity>();
    try
    {
        if (asNoTrack)
            Result = EntitySet.AsNoTracking().ToList();
        else
            Result = EntitySet.ToList();
    }
    catch (Exception ex)
    {
        Exception? Invoke(this, new ExceptionEventArgs {
            Message = ex.Message,
            InnerException = ex.InnerException,
            Source = ex.Source, StackTrace = ex.StackTrace,
            TargetSite = ex.TargetSite });
    }
    return Result;
}

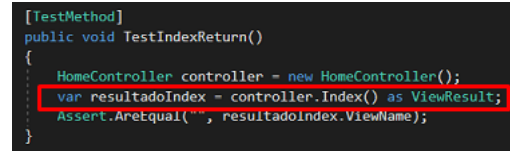
```

Fig. 12: Repository Usage Example

8 TEST

In order to realize a testing within the parts of this project, different black box tests have been carried out to check basic functionalities such as, for example, the result of database-based queries, controllers functioning, etc. Next, a small example will be detailed.

This is an example of one of the use cases made. The purpose is to verify that the *Index* method of the controller, returns a view with an empty content, since this way it would be redirecting to the url whose name has its method of action, in this case "Index".



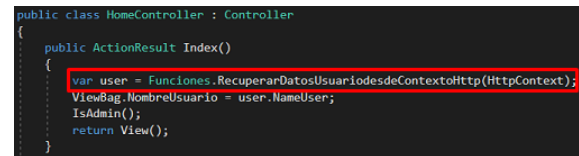
```

[TestMethod]
public void TestIndexReturn()
{
    HomeController controller = new HomeController();
    var resultadoIndex = controller.Index() as ViewResult;
    Assert.AreEqual("", resultadoIndex.ViewName);
}

```

Fig. 13: Use Case defined

This process originates a call to perform a data collection of the http context.



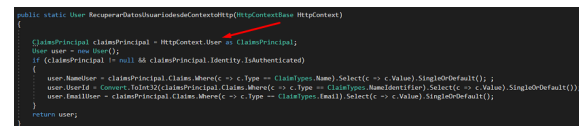
```

public class HomeController : Controller
{
    public IActionResult Index()
    {
        var user = Funciones.RecuperarDatosUsuarioDesdeContextoHttp(HttpContext);
        ViewBag.NombreUsuario = user.NameUser;
        IsAdmin();
        return View();
    }
}

```

Fig. 14: Index method

Initially, during this data collection, it was being assumed that the context is always correct, but in reality, it could have failed and caused the web application to stop.



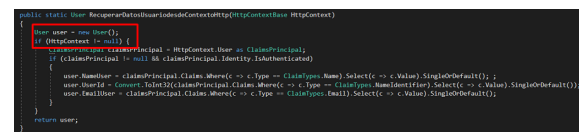
```

public static User RecuperarDatosUsuarioDesdeContextoHttp(HttpContextBase HttpContext)
{
    ClaimPrincipal claimPrincipal = HttpContext.User as ClaimPrincipal;
    User user = new User();
    if (claimPrincipal != null && claimPrincipal.Identity.IsAuthenticated)
    {
        user.Number = claimPrincipal.Claims.Where(c => c.Type == ClaimTypes.Name).Select(c => c.Value).SingleOrDefault();
        user.UserId = Convert.ToInt32(claimPrincipal.Claims.Where(c => c.Type == ClaimTypes.NameIdentifier).Select(c => c.Value).SingleOrDefault());
        user.EmailUser = claimPrincipal.Claims.Where(c => c.Type == ClaimTypes.Email).Select(c => c.Value).SingleOrDefault();
    }
    return user;
}

```

Fig. 15: Initial function

So, by doing this simple check I have been able to realize the mistake and modify the method so that it also takes into account null values in the controller and in access to user data within the http context



```

public static User RecuperarDatosUsuarioDesdeContextoHttp(HttpContextBase HttpContext)
{
    User user = new User();
    if (HttpContext != null)
    {
        ClaimPrincipal claimPrincipal = HttpContext.User as ClaimPrincipal;
        if (claimPrincipal != null && claimPrincipal.Identity.IsAuthenticated)
        {
            user.Number = claimPrincipal.Claims.Where(c => c.Type == ClaimTypes.Name).Select(c => c.Value).SingleOrDefault();
            user.UserId = Convert.ToInt32(claimPrincipal.Claims.Where(c => c.Type == ClaimTypes.NameIdentifier).Select(c => c.Value).SingleOrDefault());
            user.EmailUser = claimPrincipal.Claims.Where(c => c.Type == ClaimTypes.Email).Select(c => c.Value).SingleOrDefault();
        }
    }
    return user;
}

```

Fig. 16: Corrected function

In order to carry out these tests, NUnit has been used. An open source framework of Unit Tests for Microsoft .NET that serves the same purpose as JUnit in java, which has been used during this bachelor.

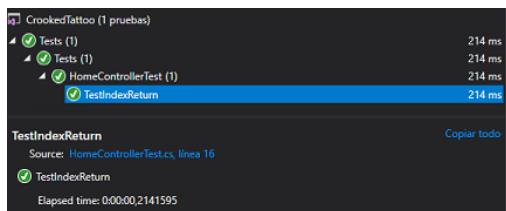


Fig. 17: NUnit

In summary, as in this case has happened, the testing has helped to correct different possible errors that exist during the intermediate processes that a method executes. That is, the simple fact of wanting to verify that what a method returns is correct, has helped to verify that the processes that are executed to return that result are also correct and controlled.

9 RESULTS

Now, The results obtained from the development of this project will be exposed. Since displaying those results requires a considerable amount of images, in order not to load this document much, a part of the results will be shown in this section and the rest in the appendix section.

Navigation Bar - Within the Home Page, we have the navigation bar that it is used for the users to navigate through the web page. Here, two different pictures will be exposed:

- How a normal user would see the navigation bar
- How an Admin user would see the navigation bar

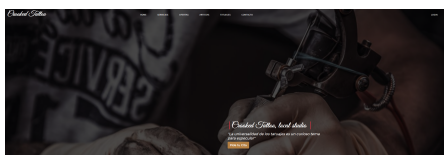


Fig. 18: Normal User view

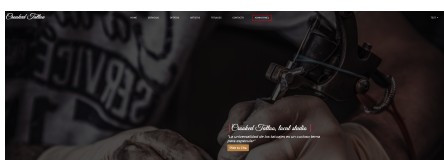


Fig. 19: Admin User view

So, if the user is has the Admin role, he will be able to access to the Admin Panel of the web system.

Forms - Forms are used by the users to apply for an appointment, create/modify users, create new sessions for an appointment, etc. All the forms used in this web application are created from ViewModels. This helps to define all data types and add an unobtrusive validation so that the user sees if something is wrong in the form before sending it.

Fig. 20: Apply for an Appointment

When an Admin user wants to apply a modification (in an appointment or user) the information of the form comes already filled with the information that contains in the DB.

Fig. 21: Modify forms

Instagram API - One of the requirements made by the Stakeholder was that the web application shall be easy to manage. As they are a tattoo studio, it is needed that their work is exposed in the web page. For that reason, Instagram API has been used so that they do not need to upload the pictures but, those will be automatically captured from their Instagram account.

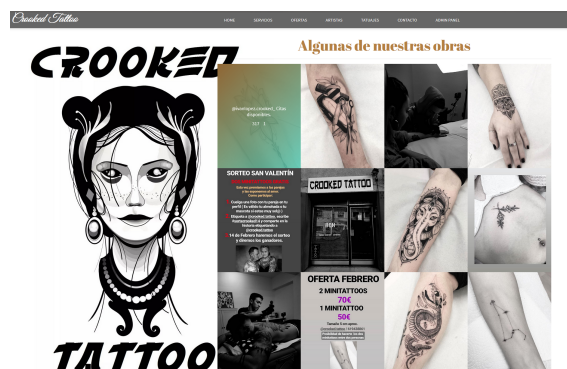


Fig. 22: Instagram Feed

Admin Panel This is the section where they will find information about the status of the business and where they will manage their appointments, users, etc. There is some information that is displayed directly from the DB by using queries so that they know their revenues, number of tattoos, and some more information required by them.

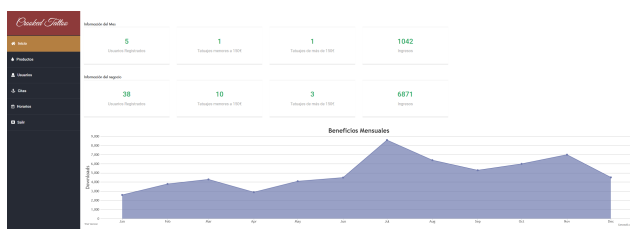


Fig. 23: Admin Panel

In order for them to look for users/appointments, there is a search bar in each of the views that instantly searches for the word/letter that the user is entering in each of the columns.

The screenshot shows the Admin Panel - Users section. It features a search bar at the top and a table of users with the following columns: Nombre, Apellido, Teléfono de contacto, Email de contacto, and Fecha de alta. The table contains several rows of user data.

Fig. 24: Admin Panel - Users section

10 CONCLUSIONS & FUTURE WORK

10.1 Conclusions

Since the beginning of this project, I believe that the development has been quite positive. Starting from the basis that at first I was a bit lost regarding the web programming, I have learned to use new technologies to explore a part of the computer science that I have always been interested in. I am also learning different software architectural patterns that I did not know which has helped me a lot to carry out a part of this project and that for sure will be useful at anytime during my future career. It should be said that at the beginning I thought that creating a web system was much more easier than it really is, I have seen that it has lot of work behind in order to create a robust solution and that lots of things need to be checked.

Thanks to the knowledge acquired during these years of degree and the help of the internet, it has been possible to develop a final solution that is functional and fulfills the objective of helping this tattoo studio with the basic management of a business and improving the quality of it (an example would be that before the creation of this web application they did not have any history about the appointments they made and now they can easily check it). Throughout the development of the project I have learned many new concepts related to web development and the need to have a good knowledge of database management, as it is vital in most projects, as well as into the management of a small business.

10.2 Future Work

As future work for this project, it must be said that some sort of small things need to be applied. Firstly, some of the pictures that this website has will be changed (once the Stakeholders provide them to me). Secondly, I will add a new layer for the project, the *Business Layer*, in order to encapsulate all the methods that are applied to provide some business logic. Once this is done, some methods that are now in the Controllers will be simply moved to this new layer as the Controller would not have to care about how the business works. Finally, the Admin Panel has a "Products" section that was intended to hold products to be sold in an online store that the application would have. As per request of the Stakeholder, because of the way how they work, they retracted about this online store. For that reason, it has been left empty but I will make it to be used for them to, internally, know about their stock (the Ink, ointments, tattooing machines and more).

Also, I will advise the Stakeholder to find for a good hosting service and introduce them to all the integration part until the web application is fully working for everyone.

ACKNOWLEDGEMENTS

First of all I want to thank my tutor Yolanda Benítez for the time dedicated to guide me through this project and for the support and motivation offered.

I would also like to thank my colleagues for the help given at times when I was stuck as well as for the advice they gave me when I started this project.

Finally, I want to thank my whole family for the support they have given me during the years that this degree has lasted and, especially, to my couple Laura for all her help and support during tough times.

REFERENCES

- [1] title = "Fresh Books"
available = https://www.freshbooks.com/financial-reporting/?ref=12779&utm_medium=marketplaces&utm_campaign=business-management&utm_source=capterra
- [2] title = "Entity Framework-Overview"
available = https://www.tutorialspoint.com/entity_framework/entity_framework_overview.html
- [3] author = "Rachel Appel"
title = "Use ViewModels to manage data & organize code in ASP.NET MVC applications"
available = <http://rachelappel.com/use-viewmodels-to-manage-data-amp-organize-code-in-asp-net-mvc-applications/>
- [4] author = "Richard"
title = "Que es Data Access Object"
available = https://www.javamexico.org/blogs/richardmx/que_es_data_access_object

11 APPENDIX

1.1 Application Pictures

LOGIN & REGISTER

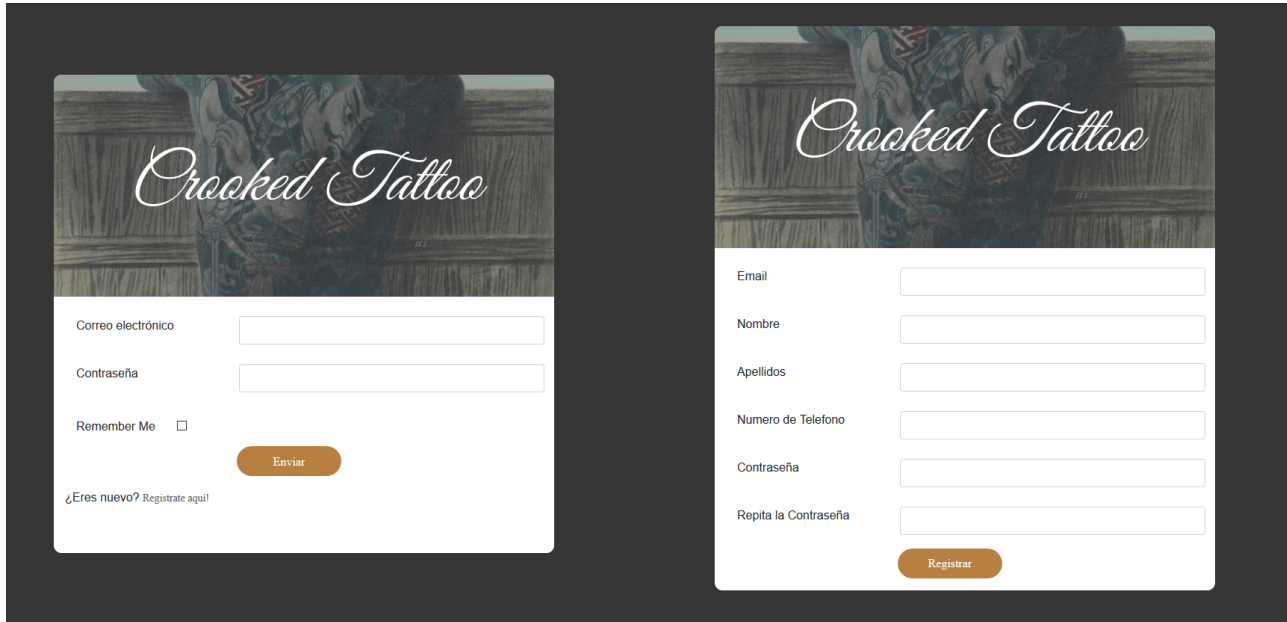


Fig. 25: Login and Register Views

HOMEPAGE

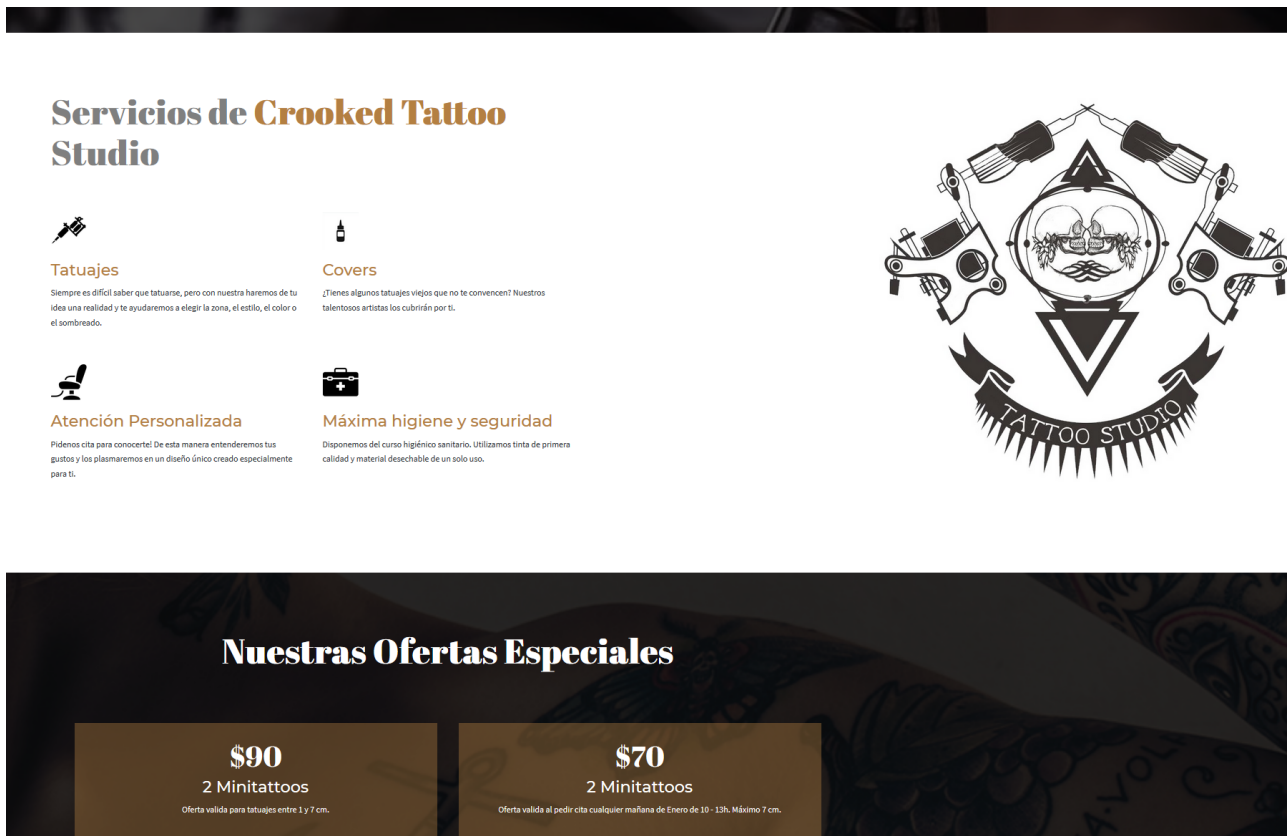


Fig. 26: Home View



Fig. 27: Home View

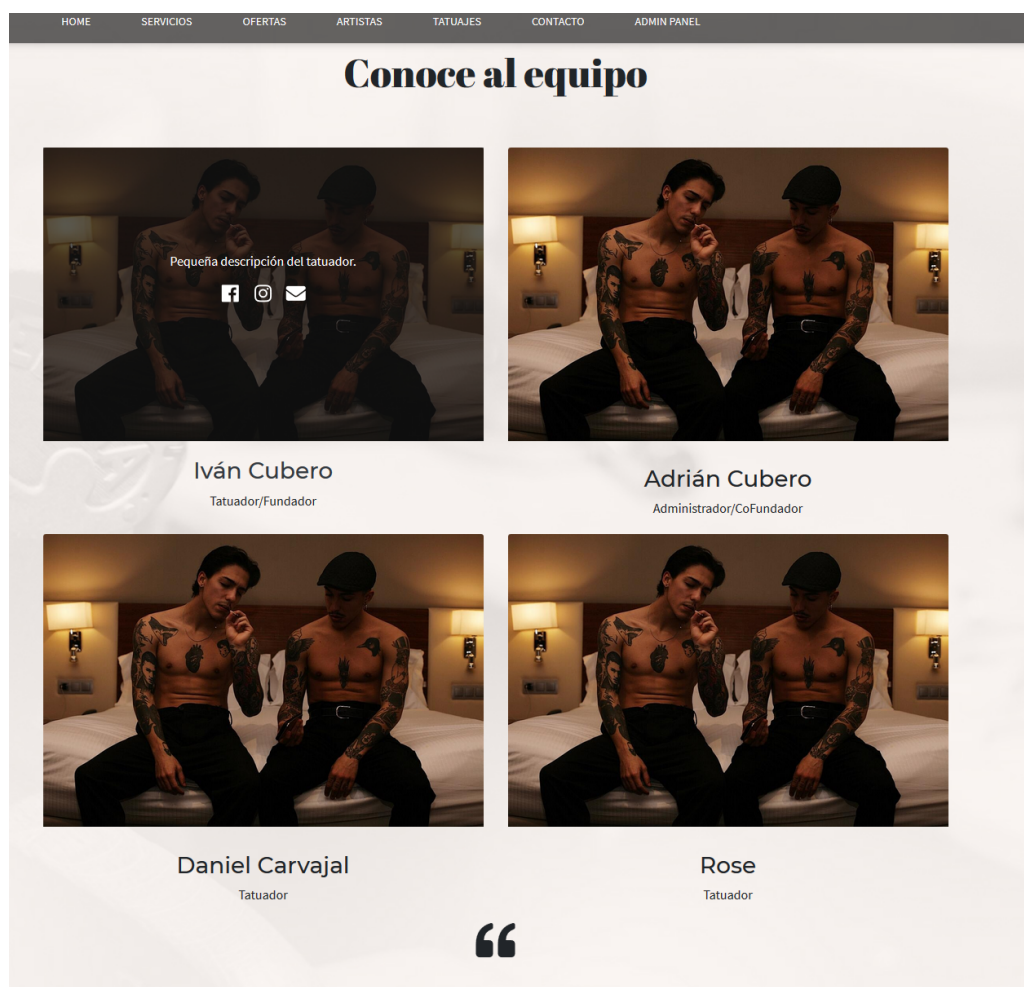


Fig. 28: Home View

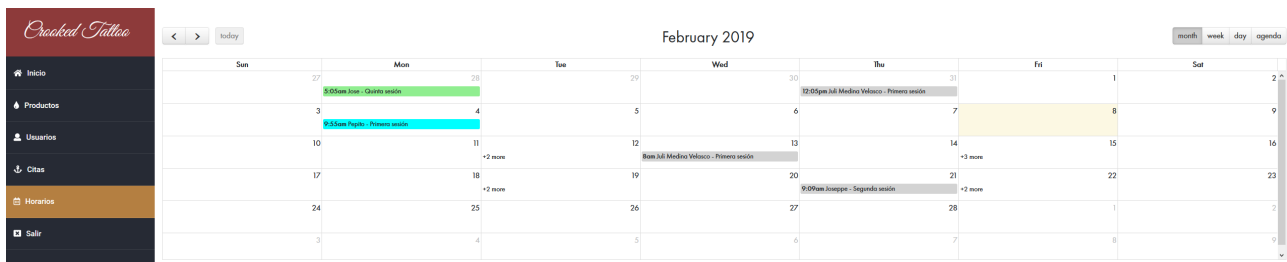


Fig. 29: Tattooers Calendar

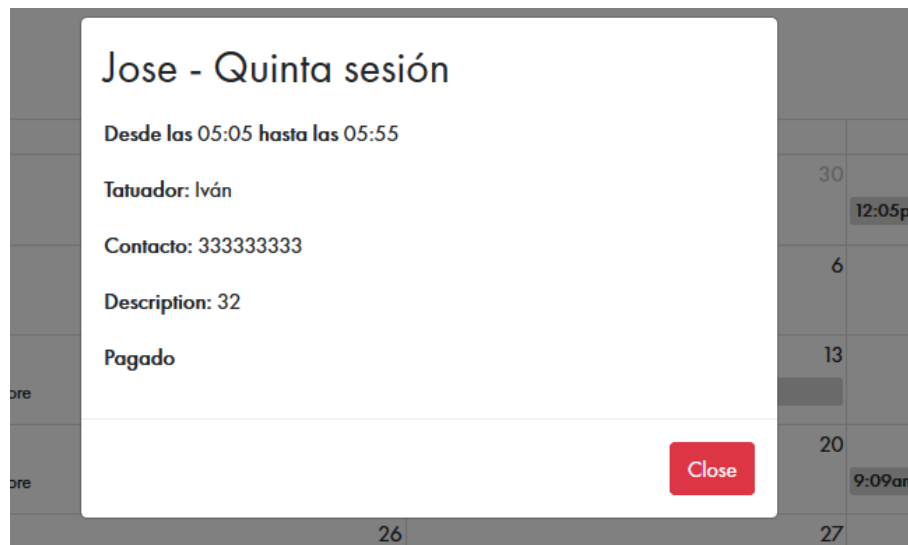


Fig. 30: Sessions Information

USER PROFILE

The screenshot shows the 'User Profile' page for a user named Julia. The page has a header with the user's name 'Julia' and the 'Cracked Tattoo' logo. Below the header, there are two tabs: 'Perfil' (selected) and 'Mis Citas'. The main content area is titled 'Mi Perfil' and contains a form with the following fields: 'Nombre' (Julia), 'Apellidos' (empty), 'Email' (Julia@Julia.es), 'Numero de Telefono' (222222222), 'Contraseña' (masked with dots), and 'Repita la Contraseña' (masked with dots). A 'Guardar' button is located at the bottom of the form.

Fig. 31: User Profile

Julia

*Cracked
Tattoo*

Perfil

Mis Citas

Mis citas

Tatuador	¿Tiene sombra?	¿Es a color?	Fecha de solicitud	Descripción	Estado	Precio	Pagado
Iván	No	No	30/11/2018 21:10:52	Tatuaje de una moto	Solicitud enviada por un usuario.	20 €	0 €

Fig. 32: User Profile

GANTT CHART

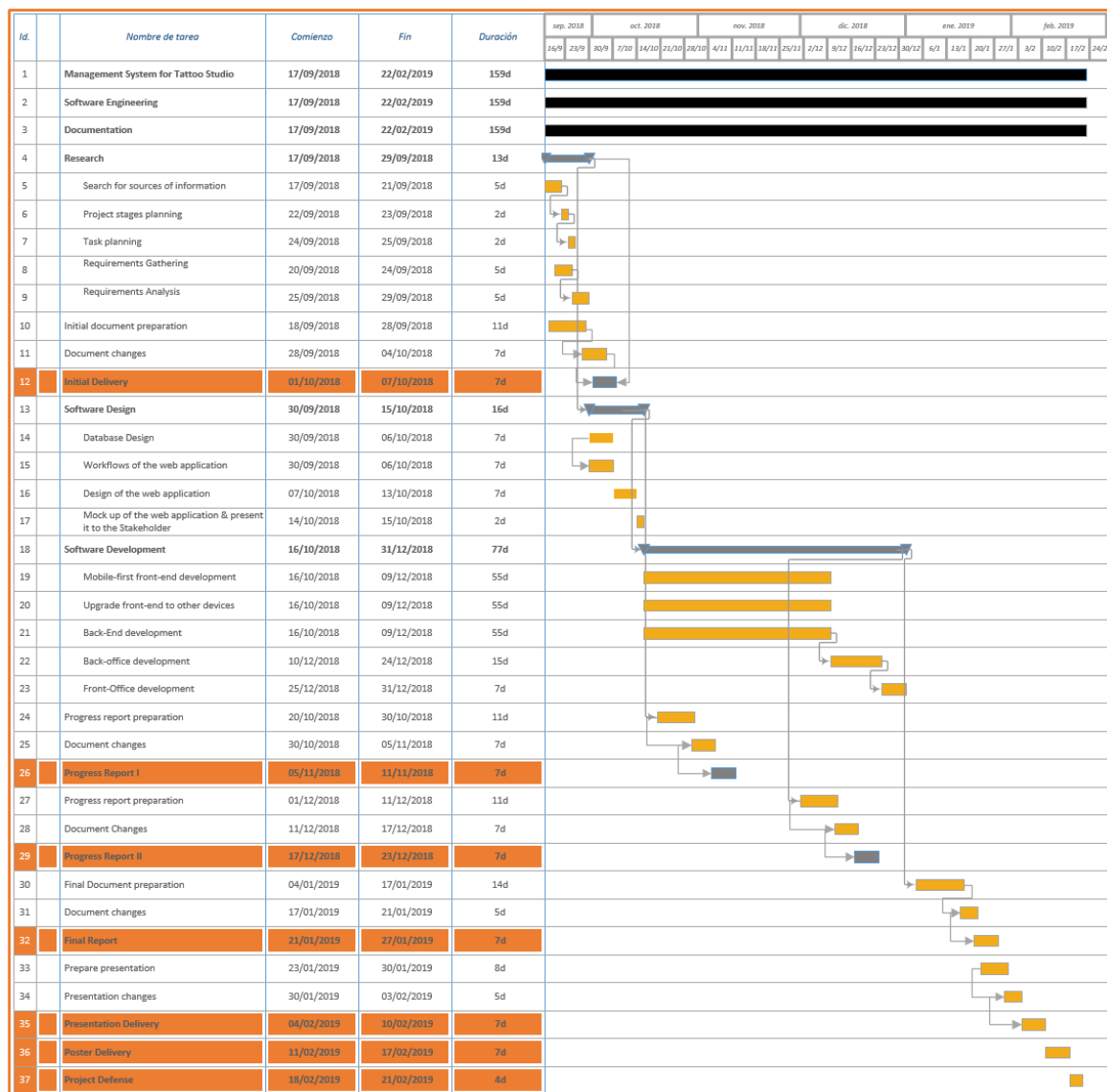


Fig. 33: User Profile