University of Thessaly
Faculty of Engineering
Department of Electrical & Computer Engineering

An electronic health record management system

Diploma Thesis

STELLA D. TOMPAZI

Supervisor
Michael Vassilakopoulos
Associate Professor

Volos, June 2019

University of Thessaly
Faculty of Engineering
Department of Electrical & Computer Engineering

An electronic health record management system

Diploma Thesis

STELLA D. TOMPAZI

Supervising committee

Supervisor	Co-supervisor	Co-supervisor
Michael Vassilakopoulos	Eleni Tousidou	Aspassia Daskalopulu
Associate Professor	Laboratory Teaching Staff	Assistant Professor

Volos, June 2019

University of Thessaly
Faculty of Engineering
Department of Electrical & Computer Engineering

Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Σύστημα διαχείρισης ηλεκτρονικού φακέλου υγείας

Διπλωματική Εργασία

ΣΤΥΛΙΑΝΗΣ Δ. ΤΟΜΠΑΖΗ

Επιβλέπων
Μιχαήλ Βασιλακόπουλος
Αναπληρωτής Καθηγητής

Βόλος, Ιούνιος 2019

Abstract

Traditional, handwritten health records were effective before we had the means to manage them easier. They surely were a significant factor in the progress of healthcare systems, as they provided a way to share health information, which led to improved health care. However, they were associated with multiple drawbacks. Costs, storage, accessibility, and some common errors regarding sorting and transfering the data, are just a few of them. As a result, the provided health care was quite mediocre.

The aim of this thesis is to overcome as many of the above problems as possible, with the use of a web application. The implementation was made with a Python-based framework, called Django. The app is useful for both doctors and patients. Specialists, along with some crucial information about their services, can access and update any information about the health record of the patient they are interested in. At the same time, they are able to organize their appointments, using a calendar, provided by the app. On the other side, patients have also access to their own health records, and can search for a doctor anytime. Moreover, there is a function which allows patients and doctors to communicate, in order to schedule an appointment.

The app is available through any device that has a compatible web browser.

Keywords

Electronic Health Record, Django, Python, Web application

Περίληψη

Η χρήση παραδοσιακών, γραπτών ιατρικών φακέλων ήταν αποτελεσματική όσο δεν είχαμε κάποιο πιο εύκολο τρόπο διαχείρισής τους. Ενώ σίγουρα πρόσφεραν σημαντικά στην ανάπτυξη του τομέα της υγείας, επιτρέποντας σχετικά εύκολα τη μετάδοση της πληροφορίας και τη βελτίωση της φροντίδας που προσφέρεται, ωστόσο δεν είναι λίγα τα προβλήματα που έπρεπε να αντιμετωπιστούν. Το κόστος, ο χώρος αποθήκευσης, η προσβασιμότητα, αλλά και τα λάθη που συνέβαιναν στη μεταφορά των στοιχείων και της αρχειοθέτησης, είναι μόνο μερικά από τα μειονεκτήματα που έχει η χρήση των κλασικών φακέλων. Αποτέλεσμα όλων των παραπάνω είναι μια μέτρια φροντίδα και εξυπηρέτηση των ασθενών που εγκυμονεί κινδύνους για την υγεία των ασθενών.

Σκοπός της διπλωματικής είναι η δημιουργία μιας εφαρμογής ιστού, η οποία λύνει όσα από τα παραπάνω προβλήματα είναι δυνατόν. Η υλοποίηση έγινε με τη χρήση ενός εργαλείου για τη δημιουργία εφαρμογών ιστού, που έχει σαν βάση την Python, το Django. Πρόκειται για μια εφαρμογή χρήσιμη τόσο σε γιατρούς, όσο και σε ασθενείς. Οι ειδικοί παραθέτουν πληροφορίες για τις υπηρεσίες που προσφέρουν μέσω της ειδικότητάς τους και ταυτόχρονα έχουν πρόσβαση σε οποιαδήποτε καταχωρημένη πληροφορία υπάρχει για τον ασθενή που τους ενδιαφέρει, και προφανώς έχουν τη δυνατότητα να προσθέσουν οποιαδήποτε πληροφορία επιθυμούν. Επίσης, μπορούν να διαχειριστούν εύκολα το πρόγραμμά τους μέσω της ατζέντας που τους παρέχεται. Από την άλλη πλευρά, οι ασθενείς έχουν και αυτοί πρόσβαση στο ιστορικό τους, ενώ μπορούν εύκολα να αναζητήσουν κάποιο γιατρό. Επιπλέον, υπάρχει υπηρεσία ανταλλαγής μηνυμάτων, ώστε να κανονίζονται επισκέψεις μέσω της εφαρμογής.

Η εφαρμογή είναι διαθέσιμη μέσω οποιασδήποτε συσκευής διαθέτει συμβατό φυλλομετρητή.

Λέξεις Κλειδιά

Εφαρμογή Ιστού, Ηλεκτρονικός Φάκελος Υγείας

Acknowledgements

I would like to thank professor Michael Vassilakopoulos for supervising this diploma thesis and giving me the chance to collaborate with him. Furthermore, I want to thank my parents and family for the guidance they offered me during all these years.

Table of contents

Abstract i

Περίληψη iii

Acknowledgements v

Table of contents vii

List of figures xi

List of tables xiii

1 Introduction 1

1.1 Thesis Objective . . . 1

1.1.1 Contribution . . . 2

1.2 Organization . . . 2

2 Related Projects 3

2.1 75Health . . . 3

2.2 InSync . . . 4

2.3 iPatientCare . . . 4

2.4 EHR Software Comparison . . . 5

3 Development and Design Tools 7

3.1 What is a Web application? . . . 7

3.1.1 How a Web App works . . . 7

3.1.2 Benefits . . . 8

3.2 Django . . . 8

3.2.1 Top Web applications that use Django . . . 9

3.2.2 The model layer . . . 10

3.2.3 The view layer . . . 11

3.2.4 URL dispatcher . . . 12

3.2.4.1 How Django processes a request . . . 12

3.2.5 The template layer 13
3.2.5.1 Configuration 13
3.2.5.2 The Django Template Language 14
3.2.6 Forms 15
3.2.6.1 GET and POST 16
3.2.6.2 Django's role in forms 16
3.2.6.3 Forms in Django 17
3.2.7 Authentication 20
3.2.7.1 The admin 20
3.2.7.2 User objects 21
3.2.8 Security 21
3.2.8.1 Cross site scripting (XSS) protection 21
3.2.8.2 Cross site request forgery (CSRF) protection 21
3.2.8.3 SQL injection protection 21
3.2.8.4 Clickjacking protection 22
3.2.8.5 SSL/HTTPS 22
3.3 Heroku 22
3.3.1 Heroku and data 22
3.3.1.1 Postgres 22
3.3.2 Ecosystem of services 24
3.3.3 Scale and enterprise 24

4 Drawbacks of paper medical records 25
4.1 Cost and storage 25
4.2 Accessibility 25
4.3 Lost productivity 26
4.4 Quality 26
4.5 Security 26
4.6 Diagnosis misidentification 26

5 Advantages of Electronic Health Records 27
5.1 Real-time records 27
5.2 Accessibility 27
5.3 Safety 27
5.4 Security 28
5.5 EHR pros in a nutshell 28

6 Electronic Health Record web application 29
6.1 ICD-10 29
6.2 EHR Homepage 29
6.2.1 Users 30
6.3 User: Doctor 32

6.3.1 Registration for professionals . . . 32
6.3.2 Main page . . . 33
6.3.3 Schedule . . . 33
6.3.3.1 Create/View Event . . . 34
6.3.4 Opening Hours . . . 36
6.3.5 Price List . . . 37
6.3.6 Messages . . . 37
6.3.7 My Profile . . . 38
6.3.8 Search . . . 40
6.3.8.1 Add New Record . . . 40
6.3.8.2 Find Record . . . 44
6.4 User: Patient . . . 45
6.4.1 Registration for patient . . . 45
6.4.2 Homepage . . . 46
6.4.3 My Profile . . . 47
6.4.4 My Records . . . 48
6.4.5 Search Doctor . . . 49
6.4.6 Favourites . . . 52
6.4.7 Messages . . . 52

7 Implementation **53**
7.1 Models . . . 54
7.2 Views . . . 54
7.2.1 ListView . . . 54
7.2.2 DetailVew . . . 55
7.2.3 DeleteView . . . 56
7.2.4 UpdateView . . . 56
7.2.5 function view . . . 56
7.3 Forms . . . 58
7.4 Templates . . . 58

8 Conclusion **61**
8.1 Summary and conclusion . . . 61
8.2 Future Extensions . . . 61

I **63**
I.1 How to install Django on Windows . . . 63
I.1.1 Install Python . . . 63
I.1.2 pip . . . 63
I.1.3 virtualenv and virualenvwrapper . . . 63
I.1.4 Install Django . . . 64
I.2 Creating a project in Django . . . 64

I.2.1 Database setup 65

I.2.1.1 Creating Models 65

I.2.2 Database API 65

I.2.3 Admin User 65

I.3 Getting Started on Heroku with Python 67

I.3.1 Prepare the app 68

I.3.2 Deploy the app 68

I.3.3 Procfile 68

I.3.4 App dependencies 69

Bibliography **71**

Abbreviations **73**

List of figures

3.1 How a Web Application works (source: [24]) 8
3.2 Django Architecture Flowchart (source: [5]) 9

6.1 EHR - Homepage 30
6.2 EHR - Patient - Log In 30
6.3 EHR - Doctor - Registration 33
6.4 EHR - Doctor - Main page 33
6.5 EHR - Doctor - Schedule 34
6.6 EHR - Doctor - New Event 34
6.7 EHR - Doctor - New Event 35
6.8 EHR - Doctor - Event 35
6.9 EHR - Doctor - Opening Hours 36
6.10 EHR - Doctor - Opening Hours Form 36
6.11 EHR - Doctor - Price List 37
6.12 EHR - Doctor - Price List Form 37
6.13 EHR - Doctor - Inbox 38
6.14 EHR - Doctor - Reply 38
6.15 EHR - Doctor - Sent 38
6.16 EHR - Doctor - My Profile 39
6.17 EHR - Doctor - User Profile Update 39
6.18 EHR - Doctor - Profile Info Update 39
6.19 EHR - Doctor - Search 40
6.20 EHR - Doctor - Patient's Profile 41
6.21 EHR - Doctor - New Diagnosis Record (part 1) 41
6.22 EHR - Doctor - New Diagnosis Record ICD10 code 42
6.23 EHR - Doctor - New Diagnosis Record (part 2) 42
6.24 EHR - Doctor - New Diagnosis Record (part 3) 43
6.25 EHR - Doctor - New Vaccination Record 43
6.26 EHR - Doctor - New Surgery Record 44
6.27 EHR - Doctor - Diagnosis Records List 44
6.28 EHR - Doctor - Diagnosis Record 45
6.29 EHR - Patient - Request Appointment 46

6.30 EHR - Patient - Homepage 47
6.31 EHR - Patient - Scheduled Appointment 47
6.32 EHR - Patient - My Profile 48
6.33 EHR - Patient - Diagnosis History List 49
6.34 EHR - Patient - Diagnosis Record 49
6.35 EHR - Patient - Specialties List 50
6.36 EHR - Patient - List of Doctors 50
6.37 EHR - Patient - List of Doctors 51
6.38 EHR - Patient - Request Appointment 51
6.39 EHR - Patient - Favourites 52
6.40 EHR - Patient - Inbox 52

7.1 Django MVT (source: [4]) 53

I.1 Django - Admin - Log In 66
I.2 Django - Admin - Index 67

List of tables

2.1 EHR software comparison table 5

Chapter 1

Introduction

The history of medical record management begins with the simplest form of the ability to record a patient's symptoms, complaints and treatment for the use of one provider, to a comprehensive aggregation of data to improve care and outcomes. While enhancements in technology increased responsibilities from managing paper records to managing the full scope of the process assembling and sharing electronically-captured information, the name of the function eventually changed to health information management. The original forms of medical records were narratives, written in ancient Greece, to archive successful treatments and share observations. In the 1920s, physicians realized that documenting thoroughly their observations and process was the best way to improve diagnosing and treating. For a very long time, the only way to do that was by having handwritten archives [19, 22, 25].

This method though has quite a few disadvantages [20], including:

- **Costs** - associated with supplies and storing
- **Lost productivity** - mistakes like misplacing or sorting manually are time consuming
- **Accessibility** - not accessible by more than one physician at a time, unless there is a duplicate
- **Quality** - paper is considered fairly fragile, illegible handwriting by professionals is very common
- **Fragmentation** - records at various facilities may be incomplete
- **Diagnosis misidentification** - the lack of a universal medical language can lead to lost information through translation

1.1 Thesis Objective

This dissertation aims to eliminate as much of the above problems as possible, by developing a web app that manages health records electronically. The web application requires minimum costs comparing to using handwritten records. There is no lost productivity, as misplacing and sorting the files isn't a concern anymore. Meanwhile, every record will be accessible by those interested and no one needs to worry about the quality of the file, because it will be digitally written. Finally, with the integrated ICD-10 codes, everyone who reads the file will have exactly the information they need about their patient.

1.1.1 Contribution

The contribution of the thesis can be summarized as follows:

1. Real-time records that can be available any time.
2. Accessibility at any moment by multiple specialists.
3. Security and legitimate storing.
4. Patient's safety.
5. Scheduling.
6. Universal medical language.

1.2 Organization

Similar projects are presented in Chapter 2. In Chapter 3 there are some technical information about the tools that were used during the development of the web application, that are essential to understand how it was built. Chapter 4 shows why such an app is significant by discussing the drawbacks of paper medical records, while Chapter 5 εmphasized on the advantages of using Electronic Health Records. There follows a detailed presentation of how the web app works in Chapter 6, including many photos. Chapter 7 explains some of the code used to develop the app. Finally, Chapter 8 consists of a conclusion and future extensions to the project.

Chapter 2

Related Projects

As computers became smaller and easily accessible, hospitals and clinics started creating their own archive. However, these files could not be shared between different departments. As health systems began to grow faster, the increased need for interoperability that supported data-sharing grew. The importance of integrated electronic health records (EHRs) to enable specialists to make better decisions expanded, thus many physicians used them in order to minimize the incidence of error.

2.1 75Health

75Health is the Electronic Health Record software which is not just about being paperless. It is about inculcating a tradition of simulated and simplified working with intelligent systems that help doctors work efficiently and enhance patient care. Working on a cloud-based technology, it is cost-efficient, fast and secure. 75Health is an attempt for improving the doctor-patient relationship. Successfully track, store and pass on patient information from one end to the other [1, 7].

Web based electronic health records software are patient-centric and real-time digital version of the paper charts describing in detail the patient data. This way of furnishing data enables information exchange in a quick, reliable, and accurate manner to the destination in a secure way. It becomes necessary for patients at times, to approach multiple medical treatment at different medical facilities. In certain cases, a number of physicians located at various places need to be contacted simultaneously as well. The problems such as understanding and furnishing details—conveying what each doctor requires, and where the data has to reach, etc, have to be tackled in a quick and intelligent manner. This multiplicity is met effectively using EHRs. EHR software unfolds case-specific solutions.

Digitizing patient details helps meet these challenges in an efficient way as in this format, information can be circulated with ease irrespective of time and location. EHR Software, created keeping in mind these features, helps individuals as well as professionals in the medical service field.

2.2 InSync

InSync is a fully integrated, easy-to-use, cloud-based healthcare IT solutions. It provides practice management, revenue cycle management and medical transcription solutions and services that can better serve you and your patients, grow your practice, help maintain compliance and increase productivity and profitability [7, 13]. InSync's electronic medical record and practice management software solutions are configured to work with your practice, not against it while operating seamlessly across all workflows. InSync is committed to delivering the market's most complete healthcare solution - one that will unify the patient experience across a growing number of patient healthcare touch points, for both providers and patients alike.

InSync provides the following software solutions, depending on a professionals practice:

- Behavioral Health Software
- Substance Abuse Software
- Ob/Gyn Software
- Pediatric Therapy Software
- Primary Care Software
- Physical Therapy Software
- Software for other specialties

2.3 iPatientCare

iPatientCare a pioneer in mHealth and Cloud-based EHR and integrated PMS, Patient Portal, HIE, and mobile solutions, serves the ambulatory, acute/sub-acute is a Preferred MU Partners for more than 77,000 physicians/users nationally. Certified for Meaningful Use stage 2 Ambulatory and Inpatient, selected by NASA Space Medicine, US Army, numerous Regional Extension Centers (REC), hospitals and has been designated as a Test EHR by the CMS [7, 15].

Their simple and intuitive web-based electronic medical records software makes patient charting, e-prescribing, scheduling, and other tasks easy, so that a specialist can devote more time to their patients. Its user-centric design and high performance architecture allows to work efficiently, again leaving a lot of time for the patients. The patient records are securely held in a HIPAA compliant storage.

iPatientCare also provides different software for different specialties, such as:

- Allergy
- Cardiology
- Chiropractic
- Dermatology
- ENT
- Gastroenterology
- Nephrology
- OB/Gyn
- Opthalmology

- Orthopedics
- Pain Management
- Pediatrics
- Physical Therapy
- Podiatry
- Psychiatry
- Pulmonary
- Rheumatology
- Urology
- Vascular

2.4 EHR Software Comparison

The EHR developed for this thesis might not include hand/voice recognition, but what makes it stand out, is that EHR has integrated the addition of ICD-10 codes in diagnosis records. This function is quite important, as it allows professionals to communicate through a global medical language, without worrying about confusing others who gain access to the records.

Below there is a comparison table between EHR and other electronic medical records software [7].

	EHR	75Health	inSync	iPatientCare
Medical History	✓	✓	✓	✓
Appointment Scheduling	✓	✓	✓	✓
E-Prescribing	✓	✓	✓	✓
Hand/Voice Recognition	-	✓	✓	✓
Billing	-	✓	✓	-
ICD-10	✓	-	-	✓

Table 2.1: EHR software comparison table

Chapter 3

Development and Design Tools

3.1 What is a Web application?

Any program that performs functions by using a web browser as its client, is actually, called a Web Application. Countless businesses use the internet in order to commute information with their target market, as it is a profitable communication channel. Additionally, it gives them a way to make quick and secure transactions. Nevertheless, the business needs to be able to store significant data, as well as to process them, to provide sufficient services.

Web applications use client-side scripts (JavaScript, HTML) to display information to users and server-side scripts (PHP, ASP) to deal with the storage and data. In this way, users are able to interact with a company using online forms, making transactions, etc [24].

3.1.1 How a Web App works

Normally, Web applications are written in browser-supported languages, like JavaScript and HTML. Some of them are totally static, which means that they do not require any processing at the server at all, while others are dynamic and require server-side processing.

A common web application flow looks like this:

1. A user makes a request to the **web server**, either through the applications interface or through a browser.

2. The **web server** forwards the request to the correct **web application server**

3. The **Web application server** performs the requested task (processing the data or querying the database for instance) and generates the results, which then are forwarded to the **web server**, including the requested information.

4. The **Web server** responds the requested information and eventually it appears on the user's display.

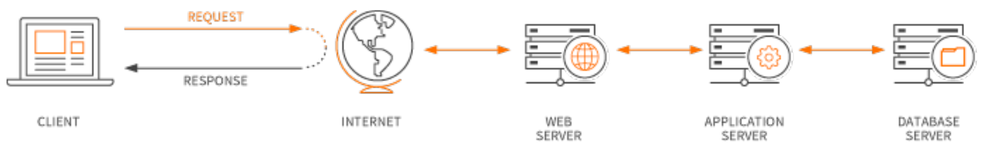


Figure 3.1: How a Web Application works (source: [24])

3.1.2 Benefits

- They can run on various platforms regardless of the device or OS. Their only demand is a compatible browser.
- There are no space limitations, as a result of not being installed on the hard drive.
- Compatibility issues are eradicated, as all users have access to the same version.
- Decreased cost for businesses and users as well.
- Reduced software piracy in web applications that require subscription (i.e. SaaS).

3.2 Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design, and most importantly it is free and open source. A framework is actually a collection of modules that make development simple. Basically, instead of starting from zero, they allow you to build applications and websites based on existing sources. This is how modern websites and apps include advanced functionality like management and admin panels, authentication support, etc. If someone tried to create a website from scratch, they'd need to develop all these components by themselves. By using a framework, a developer just has to configure these components in order to match their project [2, 6, 11, 21, 3].

Django provides:

- **Speed**. It was designed to help developers build their applications as fast as possible.
- **Modules**. It combines many extras that can be used to handle common Web development tasks, such as content administration, user authentication, site maps, etc.
- **Security**. Security means a great deal, thus Django helps developers avoid typical security mistakes, like SQL injection, cross-site request forgery, cross-site scripting and clickjacking. A protected way to handle user accounts and passwords is provides through the authentication system.
- **Scalability**. It has the ability to scale to meet the haviest traffic needs.

- **Versatility**. Companies, organizations and governments have used Django to build such different kind of things, from content management systems to social networks, or even scientific computing platforms.

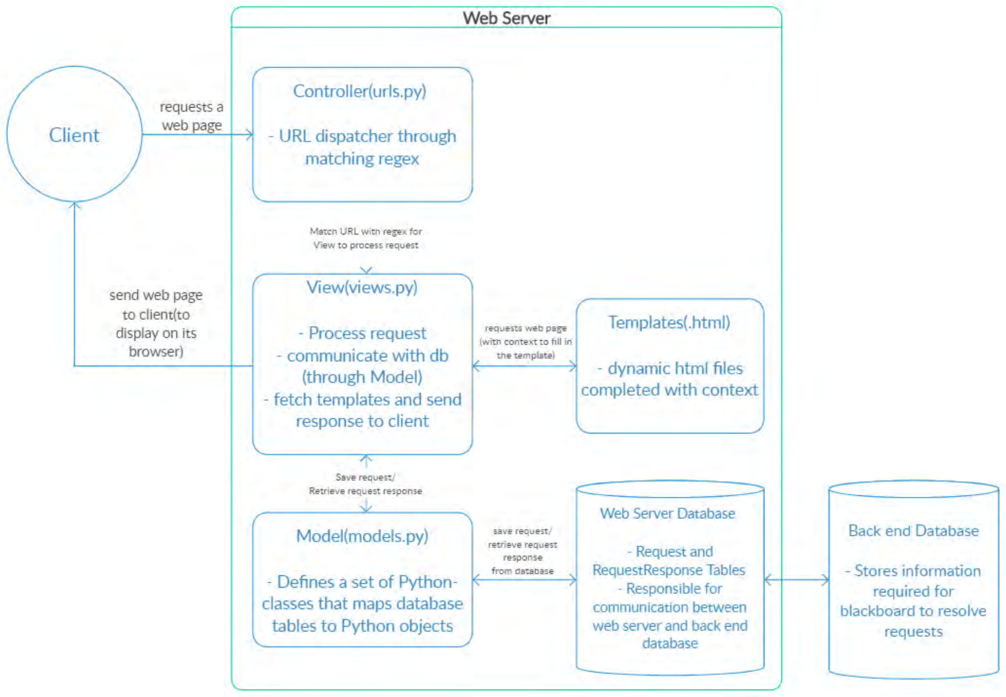

Web Server
Client
requests a web page
Controller(urls.py)
- URL dispatcher through matching regex
Match URL with regex for View to process request
send web page to client(to display on its browser)
View(views.py)
- Process request
- communicate with db (through Model)
- fetch templates and send response to client
requests web page (with context to fill in the template)
Templates(.html)
- dynamic html files completed with context
Save request/ Retrieve request response
Model(models.py)
- Defines a set of Python-classes that maps database tables to Python objects
save request/ retrieve request response from database
Web Server Database
- Request and RequestResponse Tables
- Responsible for communication between web server and back end database
Back end Database
- Stores information required for blackboard to resolve requests

Figure 3.2: Django Architecture Flowchart (source: [5])

3.2.1 Top Web applications that use Django

Many known web applications have turned to Django [23].
Some of them are:

- Instagram
- Spotify
- Youtube
- The Washington Post
- Dropbox
- Mozilla
- Prezi

3.2.2 The model layer

A model contains all the important information about the data, like fields and behaviour. Generally, each model represents a database table.

- Each model is a Python class with a **django.db.models.Model** subclass.
- Every database field is represented by a model attribute.
- With the above, Django gives an automatically-generated database-access API

The following model defines a **PatientProfile**:

```
from django.db import models

class PatientProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)

    birthday = models.DateField(null=True)
    social_security_number = models.CharField(max_length=11, unique=True)
    gender = (
        ('Male', 'Male'),
        ('Female', 'Female')
    )
    sex = models.CharField(max_length=6, choices=gender, default='Male')
    insurance = models.CharField(max_length=100, blank=True)

    blood_type = (
        ('O-', 'O-'),
        ('O+', 'O+'),
        ('A-', 'A-'),
        ('A+', 'A+'),
        ('B-', 'B-'),
        ('B+', 'B+'),
        ('AB-', 'AB-'),
        ('AB+', 'AB+'),
    )
    blood = models.CharField(max_length=3, choices=blood_type, default='')
    city = models.CharField(max_length=50, blank=True)
    address = models.CharField(max_length=50, blank=True)
    phone = models.CharField(max_length=12, blank=True)
    important_notes = models.TextField(default='')
```

The above model would create a database table like this:

```
CREATE TABLE mr_patientprofile (
    "id" serial NOT NULL PRIMARY KEY AUTOINCREMENT,
    "birthday" date NULL,
    "social_security_number" varchar(11) NOT NULL UNIQUE,
    "sex" varchar(6) NOT NULL,
    "insurance" varchar(100) NOT NULL,
    "blood" varchar(3) NOT NULL,
    "city" varchar(50) NOT NULL,
    "address" varchar(50) NOT NULL,
    "phone" varchar(12) NOT NULL,
    "important_notes" text NOT NULL,
    "user_id" integer NOT NULL UNIQUE REFERENCES "auth_user" ("id")
            DEFERRABLE INITIALLY DEFERRED
);
```

3.2.3 The view layer

A view function, is actually a Python function that takes a Web request and returns a Web response. The response can be literally anything, HTML content of a web page, 404 error, redirect, an image, an XML document, etc. The view is used to generate this response, by executing any code that is necessary. For convenience, the views exist in a file called **views.py**, in the project's or app's directory.

A simple view that gives us the index when a doctor is logged in.

```
from django.shortcuts import render
from django.utils.timezone import datetime

def index(request):
    today = datetime.today()
    todays_events = Event.objects.filter(
        doctor=request.user.userprofile,
        start_time__year=today.year,
        start_time__month=today.month,
        start_time__day=today.day)
    received_messages = Message.objects.filter(
        receiver=request.user,
        was_read=False)

    return render(request, 'mr/index.html', {
        "home": "active",
        "todays_events": todays_events,
        "received_messages": received_messages
        })
```

Let's step through this code one line at a time:

- First, we import **render** from the **django.shortcuts** module, along with **datetime** from the **django.utils.timezone** library.
- **Request** it the object used to generate this response.
- Next, we use **datetime.today** which returns the current local datetime.
- We define **todays_events** and **received_messages** which both are querysets and consist of events that are scheduled for today and unread messages associated with the current user, respectively.

- Finally, we use **render** to return the desired content, in this case the content includes **todays_events** and **received_messages** querysets and a **home** variable with **value**=”active” and helps declare which tab is currently active in out homepage. This content is being given to **mr/index.html** template.

To display this view in a particular URL, a *URLconf* should be created.

3.2.4 URL dispatcher

To design URLs for an app, you create a Python module informally called a URLconf (URL configuration). This module is pure Python code, used to map URL path expressions to Python functions (views). This mapping can have any length. short or long, and even reference to other mappings. The fact that it’s pure Python, makes it possible to have a dynamic construction.

3.2.4.1 How Django processes a request

Algorithm 1 Algorithm to decide which Python code to execute when a request is made

```
1: Django determines the root URLconf module to use. Ordinarily, this is the value of the ROOT_URLCONF
   setting, but if the incoming HttpRequest object has a urlconf attribute, its value will be used in place of the
   ROOT_URLCONF setting.
2: Django loads that Python module and looks for the variable urlpatterns.
   This should be a sequence of django.urls.path() and/or django.urls.re_path() instances.
3: Django runs through each URL pattern, in order, and stops at the first one that matches the requested URL.
4: Once one of the URL patterns matches, Django imports and calls the given view, which is a simple Python function
   (or a class-based view). The view gets passed the following arguments:

     • An instance of HttpRequest

     • If the matched URL pattern returned no named groups, then the matches from the regular expression are
       provided as positional arguments.

     • The keyword arguments are made up of any named parts matched by the path expression, overridden by
       any arguments specified in the optional kwargs argument to django.urls.path() or django.urls.re_path().

5: If no URL pattern matches, or if an exception is raised during any point in this process, Django invokes an appro-
   priate error-handling view.
```

A sample URLconf:

```python
from django.urls import path
from . import views

app_name = 'mr'
urlpatterns = [
    path('', views.ehr_index, name='ehr_index'),
    path('homepage/', views.index, name='index'),
    path('register/', views.register, name='register'),
    path('login/', views.user_login, name='login'),
    path('logout/', views.user_logout, name='logout'),
    path('calendar/', views.CalendarView.as_view(), name='calendar'),
    path('calendar/new_event/', views.new_event, name='new_event'),
    path('calendar/event/edit/<int:pk>/', views.EventUpdate.as_view(),
        name='edit_event'),
    path('calendar/event/delete/<int:pk>', views.EventDelete.as_view(),
        name='delete_event')
]
```

Notes:

- Angle brackets are used to capture a value from the URL.
- Captured values can include a converter type. For example, **<int:name>** can be used to capture an integer parameter. In case there isn't any converter, any string, excluding a / character, is matched.

Example requests:

- A request to **/calendar/event/edit/5/** would match the 8th pattern of the list.
- **/calendar/event/delete/** would not match any of the patterns.

3.2.5 The template layer

Every web framework needs a handy way to generate dynamically HTML, and so does Django. The most typical way is by using templates. In a template, there are the static parts of the HTML, along with some special syntax, used to illustrate how the dynamic content will be inserted.

None, one, or even multiple template engines can be used to configure a Django project. Django has its own template-system, called Django Template Language or DTL. Jinja2 is a common alternative.

A standard API is used for loading and rendering templates in Django. Loading includes finding the template for a specific identifier and preprocessing it. Rendering means interpolating the template with context data and returning the resulting string.

The DTL is also used by Django's contrib apps that include templates.

3.2.5.1 Configuration

To configure template engines, the **TEMPLATES** setting is used. It includes a list of configurations, one for each engine. The default value is empty. The **startproject** command generates a **settings.py** file with a more useful value.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            ... some options here ...
        },
    },
]
```

BACKEND is a dotted Python path to a template engine class implementing Django's template backend API. The top-level configuration for each engine has the following common settings, as most templates get loaded from files.

- **DIRS** is a list of directories where the engine should look for template source files, in search order.
- **APP_DIRS** tells whether the engine should look for templates inside installed applications. Each backend defines a conventional name for the subdirectory inside applications where its templates should be stored.

3.2.5.2 The Django Template Language

Django's template language is designed in such a way, that those working with HTML feel very comfortable.

A template is basically a text file, and is able to generate any text-base format (HTML, XML, CSV, etc.). It includes **tags**, that are used to control the logic of the template, and **variables**, which during the evaluation of the template, get replaced with values.

The function of **tags** is similar to some programming constructs, such as boolean tests using the **if** tag, looping using the **for** tag, etc. However, these are not executed as common Python code, and the system will not execute arbitrary Python expressions.

The Django Template Language syntax consists of the following four constructs.

1. **Variables** look like this: **{{ variable }}**

   When a template gets evaluated, the variable gets replaced by a corresponding value. Variable names can not start with underscore. They consist of combinations of alphanumeric characters and the underscore.

   A dot **(.)** is used to access the attributes of a variable.

2. **Tags** look like this: **{% tag %}**

   Tags can be more complicated than variables. They can be used to create text in the output, control the flow with logic or loops, load some external information, etc.

Some tags require beginning and ending tags:

(i.e. **{% tag %} ... tag contents ... {% endtag %}**), such as:

- {% for %} ... {% endfor %}
- {% if %} ... {% elif %} ... {% else %} ... {% endif %}

3. **Filters** are like this: **{{ name|lower }}**

   This would display the value of the name variable, after filtering through the lower filter, which actually converts the text to lowercase. More than one filters can be applied with the chain method.

4. **Comments**.

   They are used to comment-out part of a line in a template. Their syntax is: **{# #}**

Example:

```
{% extends 'mr/base.html' %}

{% block body_block %}
    <h3 class="text-center">Welcome {{ user.first_name }}!</h3>
    <table class="table">
        {% if todays_events %}
            <tr>
                <th>Today's Appointments</th>
            </tr>
        {% endif %}
        {% for event in todays_events %}
            <tr>
                <td><a style="cursor: pointer"
                    href="{% url 'mr:edit_event' event.id %}">
                    {{ event.start_time.time}} -
                    {{ event.patient.user.get_full_name }}
                    </a>
                </td>
            </tr>
        {% empty %}
            <p>You have no appointments scheduled for today.</p>
        {% endfor %}
    </table>
{% endblock %}
```

3.2.6 Forms

An extended list of tools and libraries that Django provides, help the developers build forms depending on their needs, either accept input from site visitors, or process the input to respond.

In HTML, a form is built inside **<form>...</form>**, which includes a collection of elements. These elements allow the visitor to do different actions, such as enter a text, select between options, manipulate objects and send these information to the server.

Some of them are so simple that are built into HTML itself, while others that can be quite complex, as for example an interface that pops a date picker, will probably use CSS and JavaScript as well.

Regarding an **<input>** element, a form must specify a couple of things:

- *where*: the URL to which the data corresponding to the user's input should be returned
- *how*: the HTTP method the data should be returned by

The Django admin form, for example, is consisting of several **<input>** type elements.

- **type="text"** for the username
- **type="password"** for the password
- **type="submit"** for the log in button

It also contains some hidden text fields that are not visible to the user, that help Django decide what should happen next.

The **<form>**'s **action** attribute specifies the URL - **/admin/** - that the form data should be sent, while the **method** attribute - **post** - declares the HTTP mechanism that should be used.

When the **<input type="submit" value="Log in">** element is triggered, the data is returned to **/admin/**.

3.2.6.1 GET and POST

When dealing with forms, there are only two available methods, **GET** and **POST**. Django's login form uses the **POST** method to send the data.

GET bundles the submitted data into a string, which later to compose a URL, which contains the address where the data must be sent, as well as the data keys and values.

GET and **POST** are generally used for different purposes. Requests that could change the state of a system, update the database for example, should use the **POST** method, while requests that do not affect the system at all, should use the **GET** method.

GET would also be inappropriate for a password form, as the password would appear in the URL, in the browser history and server logs, all in plain text. If a web application user **GET** requests for admin forms is a security risk. This way, an attacker can mimic a form's request and consequently gain access in sensitive information. **GET** would be suitable for different actions, like a search form.

POST offers better control over access, especially when used with other Django's protection, suck as *CSRF protection*.

3.2.6.2 Django's role in forms

Handling forms can be complex. Numerous items of different types of data need to be prepared for display in a form, rendered as HTML, edited using a user-friendly interface, and then returned back to the server where they first get cleaned and validated, and then saved or even passed on for processing.

This process can be simplified thanks to Django's form functionality.

Django handles three distinct parts of the work involved in forms:

When handling forms, Django deals with three parts:

- get data ready for rendering by preparing and restructuring them
- creating HTML forms for the data
- receiving and processing submitted forms and data from the client

By choice, someone can do all of this manually.

3.2.6.3 Forms in Django

Django Form Class

A form class describes a form and determines the way it works and appears, just like a Django model describes the structure of an object. A form class's fields map to HTML form **<input>** elements. Form's fields are also classes that manage data when a form is submitted.

Every field type handles different kinds of data, and have to do different things with it.

A form field is displayed to a user in the browser as an HTML "widget" - a piece of user interface machinery. There is a default Widget class for each field type, which can be overridden if required.

Instantiating, processing, and rendering forms

Rendering an object:

1. gets hold of it in the view (fetch it from the database, for example)
2. passes it to the template context
3. expands it to HTML markup using template variables

Rendering a form in a template is quite similar to rendering other objects. A form gets instantiated in the view, and can be left empty or pre-populated, for example, with:

- data from a saved model instance (in case of update)
- data collated from other sources
- data received from a another HTML form submission

Building a form

- **The Form class**

  forms.py

```
from django import forms
from mr.models import *
from django.contrib.auth.models import User

class UserForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput())

    class Meta:
        model = User
        fields = ('username', 'email', 'password',
                  'first_name', 'last_name')


class UserProfileForm(forms.ModelForm):
    class Meta:
        model = UserProfile
        fields = ('specialty', 'city', 'address', 'phone')
```

- **The view**

  views.py

```
from django.shortcuts import render_to_response
from .models import *
from .forms import *


def register_patient(request):
    context = RequestContext(request)

    registered = False
    if (request.method == 'POST'):
        patient_form = UserForm(data=request.POST)
        patient_profile_form = PatientProfileForm(data=request.POST)

        if (patient_form.is_valid() and patient_profile_form.is_valid()):
            user = patient_form.save()
            user.set_password(user.password)
            user.save()

            profile = patient_profile_form.save(commit=False)
            profile.user = user
            profile.save()
            registered = True

            user = authenticate(username=
                            patient_form.cleaned_data['username'],
                            password=
                            patient_form.cleaned_data['password']
                            )
            login(request, user)
        else:
            print (patient_form.errors, patient_profile_form.errors)
    else:
        patient_form = UserForm()
        patient_profile_form = PatientProfileForm()

    return render_to_response('mr/register_patient.html', {
        'patient_form': patient_form,
        'patient_profile_form': patient_profile_form,
        'registered': registered},
        context)
```

- **The Template**

  mr/register_patient.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <link rel="stylesheet" type="text/css"
         href="{% static 'mr/css/style.css' %}">
   <title>EHR register</title>
</head>
<body>
    <h1>Register in Electronic Health Record</h1>
    {% if registered %}
        <strong>Thank you for registering!</strong><br>
        <a href="{% url 'my_mr:my_index' %}">Go to Homepage</a><br>
    {% else %}
        <strong>Register here!</strong><br>
        <form id="user_form" method="post"
              action="{% url 'mr:register_patient' %}"
              enctype="multipart/form-data">

            {% csrf_token %}
            {{ patient_form.as_p }}
            {{ patient_profile_form.as_p }}

            <input type="submit" name="submit" value="Register">
        </form>
    {% endif %}
```

```
</body>
</html>
```

- **The urlconfig**

  urls.py

```
from django.urls import path
from . import views

app_name = 'mr'
urlpatterns = [
    ...
    path('register_patient/', views.register_patient,
         name='register_patient'),
    ...
]
```

3.2.7 Authentication

Django provides a full-featured and secure authentication system. It is designed to handle user accounts, permissions, groups, even cookie-based user sessions, and let users create accounts and log in/out with safety.

3.2.7.1 The admin

The automatic admin interface of Django is a powerful asset. It provides a simple, model-centric interface, by reading metadata from models, where authorized users can manage the data.

Both authentication and authorization are handled by Django;s authentication system. Authentication verifies that a user is who they claim to be, while authorization determines what an authenticated user is allowed to do.

The auth system consists of:

- Users
- Permissions: Binary (yes/no) flags designating whether a user may perform a certain task.
- Groups: A generic way of applying labels and permissions to more than one user.
- A configurable password hashing system
- Forms and view tools for logging in users, or restricting content
- A pluggable backend system

The Django authentication system is basic, so it doesn't include some features commonly used in authentication systems, like:

- Password strength checking

- Throttling of login attempts
- Authentication against third-parties (OAuth, for example)
- Object-level permissions

However, they can be implemented with extra packages.

3.2.7.2 User objects

An authentication system's foundations are the user objects. Generally, they represent the people interacting with the site. In Django's authentication framework there is only one class of users, **'superusers'** or admin **'staff'** users.

A default user has the following primary attributes:

- username
- password
- email
- first_name
- last_name

3.2.8 Security

3.2.8.1 Cross site scripting (XSS) protection

XSS attacks allow a user to inject client side scripts into the browsers of other users. For this to happen, the malicious scripts need to be stored in the database, and then retrieved and displayed to other users, or by getting them to click a link which allows the attacker's script to be executed by the user's browser.

By using Django Templates, a user is protected against the majority of such attacks.

3.2.8.2 Cross site request forgery (CSRF) protection

CSRF attacks allow a malicious user to execute actions using the credentials of another user without that user's knowledge or consent. Including a built-in protection, Django provides security against most CSRF attacks.

3.2.8.3 SQL injection protection

SQL injection is when a malicious user is able to execute arbitrary SQL code on a database, which can lead to deleted or leaked data.

Django's querysets are safe from this kind of attacks, since their queries are constructed using query parameterization.

3.2.8.4 Clickjacking protection

Clickjacking is an attack where a malicious site wraps another site in a frame. This way, an unsuspecting user can be tricked into performing unintended actions on the target site.

Django contains clickjacking protection in the form of the X-Frame-Options middleware which in a supporting browser can prevent a site from being rendered inside a frame.

3.2.8.5 SSL/HTTPS

It is always safer to deploy a site behind HTTPS, as it is less possible for malicious users to sniff credentials or any kind of information.

3.3 Heroku

Heroku is a platform as a service based (PaaS) on a managed container system, with integrated data services and a powerful ecosystem, for deploying and running modern apps. The Heroku developer experience is an app-centric approach for software delivery, integrated with today's most popular developer tools and workflows. Briefly, it enables companies to build, deliver, monitor and scale applications without trouble.

Heroku is one of the first cloud platforms, as it has been in development since June 2007, when it only supported one programming language, Ruby. Currently, it also supports Java, Node.js, Python, PHP, Scala, Clojure, Go and therefore, is described as a polyglot platform, as it allows a developer to build, run and scale applications in a similar way across different programming languages.

The procedure of deploying, tuning, configuring, scaling, and managing an application can be quite irritating. Heroku makes it as straightforward as it can be, and thus it allows developers to focus on what is more vital, building better apps that satisfy the costumers [10].

3.3.1 Heroku and data

At the center of any app lie data. Whether it is about the service or costumer data, an app and its data go conjointly. Heroku's services provide Postgres, a built in database service which comes with operational expertise. In this way, developers do not need to be concerned about provision a database.

3.3.1.1 Postgres

Heroku Postgres is a managed SQL database service provided directly by Heroku. You can access a Heroku Postgres database from any language with a PostgreSQL driver, including all languages officially supported by Heroku. In addition to a variety of management commands available via the Heroku CLI, Heroku Postgres provides a web dashboard, the ability to share queries with dataclips, and several other helpful features [17, 8, 16].

PostgreSQL is one of the world's most popular relational database management systems. Millions of developers and companies rely on PostgreSQL as their transactional data store of choice to drive application health and decision-making. And developers with knowledge of Oracle or MySQL databases can use their SQL querying experience to quickly leverage PostgreSQL's capabilities as a fast, functional, and powerful data resource.

PostgreSQL earned a strong reputation for its proven reliability, architecture, extensibility, data integrity, robust feature set, and the dedication of the open source community. PostgreSQL runs on all major operating systems, and has powerful add-ons such as PostGIS, a geospatial database extender.

PostgreSQL features intent to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help with data management no matter how big or small the dataset.

PostgreSQL tries to conform with the **SQL standard** and supports many of its features, though sometimes with slightly differing syntax or function [18].

Below there is a list of features found in PostgreSQL:

- **Data Types**
  - Primitives: Integer, Numeric, String, Boolean
  - Structured: Date/Time, Array, Range, UUID
  - Document: JSON/JSONB, XML, Key-value (Hstore)
  - Geometry: Point, Line, Circle, Polygon
  - Customizations: Composite, Custom Types
- **Data Integrity**
  - UNIQUE, NOT NULL
  - Primary Keys
  - Foreign Keys
  - Exclusion Constraints
  - Explicit Locks, Advisory Locks
- **Concurrency, Performance**
  - Indexing: B-tree, Multicolumn, Expressions, Partial
  - Advanced Indexing: GiST, SP-Gist, KNN Gist, GIN, BRIN, Covering indexes, Bloom filters
  - Sophisticated query planner / optimizer, index-only scans, multicolumn statistics
  - Transactions, Nested Transactions (via savepoints)
  - Multi-Version concurrency Control (MVCC)
  - Parallelization of read queries and building B-tree indexes

  - Table partitioning
  - All transaction isolation levels defined in the SQL standard, including Serializable
  - Just-in-time (JIT) compilation of expressions

- **Reliability, Disaster Recovery**
  - Write-ahead Logging (WAL)
  - Replication: Asynchronous, Synchronous, Logical
  - Point-in-time-recovery (PITR), active standbys
  - Tablespaces

- **Security**
  - Authentication: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate, and more
  - Robust access-control system
  - Column and row-level security

- **Extensibility**
  - Stored functions and procedures
  - Procedural Languages: PL/PGSQL, Perl, Python (and many more)
  - Foreign data wrappers: connect to other databases or streams with a standard SQL interface
  - Many extensions that provide additional functionality, including PostGIS

- **Internationalisation, Text Search**
  - Support for international character sets, e.g. through ICU collations
  - Full-text search

3.3.2 Ecosystem of services

Heroku includes add-ons that are fully-managed services, integrated for use with Heroku. Developers are able to extend the efficiency of an application by provisioning and scaling these add-ons in one command.

3.3.3 Scale and enterprise

Heroku makes it effortless for apps to scale, even if it is a 2-person startup or a 10,000-person enterprise, by insuring that the app stays app, as well as in terms of how the app is managed.

Chapter 4

Drawbacks of paper medical records

There are several reasons to have medical records for each patient. They can be used as memory support for the professionals, or give information to others who get involved in the healthcare process of a patient. It is also possible that documenting the process is required by law in several countries.

A paper based medical record includes particular distinct parts. The identity of the patient, the background and the history , the reason of visit, symptoms, the results of examinations, assessment of the situation and treatments, as well as description of the process and who the record was written by, are among them.

Throughout the visit, the physician takes notes on the description of symptoms by the patient and checks them, in an attempt to exclude some and finally reach to a diagnosis. Eventually, the more visits a patient makes, the thicker the record file gets. Obviously, analyzing paper based patient records can be challenging when using computational linguistic methods. To do that, the records need to be scanned and processed with optical character recognition techniques (OCR).

However, storing the records in digital form, can be quite convenient when it comes to accessing and analyzing data [20].

4.1 Cost and storage

Paper records, aside from taking up considerably more space, are not eco-friendly. They require a huge amount of paper and ink supplies and sooner or later they naturally deteriorate when stored for long, even in well preserved environment.

4.2 Accessibility

One of the main disadvantages of paper-based records is the fact they are not accessible simultaneously by other interested parties. In this case, they need to either be mailed or converted to digital format before they can be transferred via email for example. This process requires time and possibly money.

4.3 Lost productivity

By storing medical records in the traditional way, the possibility of errors is quite large. Because people tend to make mistakes, like misplacing files. Moreover, sorting handwritten data is almost impossible when it comes to such big archives. Even if the files are sorted as correctly as possible, retrieving a specific file can be time-consuming, especially in emergency cases, when time is significant. A combination of the above can have serious impact on a patient's care, because errors can become too easy when a professional does not have all the information they need, exactly when they need it.

4.4 Quality

Paper archives tend to be quite unreliable, as they are easily damaged. They can be shred effortlessly or ruined by any liquid. Even with common use they become deteriorated over time. Though the material is not the only flaw when it comes to paper-based records. The quality of the person writting them must also be taken into account. Deciphering handwriting that includes terminology can be extremely challenging, especially when someone is not familiar with that particular vocabulary.

4.5 Security

Paper records storage systems suffer from some vulnerabilities. They may be generally safe from unauthorized individuals, unless someone gets access to them physically, however, they are susceptible when it comes to natural disasters and accidents like floods and fire. Additionally, because of the resources they require in space and supplies, a facility will probably won't have more than one copy of each file, which means that in case of an unfortunate event, the files will be possibly completely lost.

4.6 Diagnosis misidentification

The foundation of effective healthcare is the correct identification of a patient. It is common for specialists to describe diagnosis slightly different relatively to each other, which will probably lead to misdiagnosing or mistreatment eventually. Such errors can be proven really dangerous, fatal even, for a patient's health.

Chapter 5

Advantages of Electronic Health Records

Today's demands on improved health care and outcomes have made the collection of data absolutely necessary, as it's the only way for specialists to make fast and better decisions, while reducing medical errors. In an attempt to alleviate the drawbacks of traditional medical records, specialists started to turn to Electronic Health Records (EHR). Electronic Health Records have literally revolutionized modern health care in many ways [20, 25].

5.1 Real-time records

One of the major advantages of Electronic Health Records, is that unlike regular records, they can be viewed in real-time. Hypothetically, every examination result can be available to entire medical teams just as soon as they are completed.

5.2 Accessibility

By using Electronic health records, professionals can access a patient's chart at any time, while being able to exchange and store information instantaneously. This possibility turns out extremely important during emergency situations or even when multiple professionals need to coordinate care for patients whose conditions are somehow complicated.

5.3 Safety

Compared to paper health records, EHRs provide an added level of safety, as they include all of patient's pertinent information. They make it easier to prevent possibly risky drug interactions or prohibit severe reactions regarding to allergies.

5.4 Security

Firstly, misfiling belongs to the past when using electronic health records. They are kept in a protected database, where is almost impossible to get lost or destroyed by a cup of coffee for example. At the same time, it is easy to have more than one copies of each file and keep them safe from unanticipated disasters, such as fire or floods.

5.5 EHR pros in a nutshell

- Decreased cost
- Patient safety and better health care quality
- Increased storage capabilities
- Accessibility to multiple interested parties at various locations at the same time
- Almost instantaneous information retrieval
- Real-time update
- Medical alerts
- Less charting time and fewer errors

Chapter 6

Electronic Health Record web application

For the purposes of this thesis, there was developed a web application that not only provides an easy way to store and handle electronic health records, but at the same time makes it possible to find a specialist and arrange an appointment. It is meant to be used by professionals and patients as well. Most importantly, EHR allows specialists to include one or multiple ICD10 codes in a record.

6.1 ICD-10

ICD-10 is the 10th revision of the International Statistical Classification of Diseases and Related Health Problems, which is actually a medical classification list, created by the World Health Organization (WHO). It consists of codes for diseases, symptoms, signs, complaints, abnormal findings, social circumstances, and external causes of injury or diseases [12].

It was designed to map health conditions to corresponding generic categories together with specific variations, by assigning a designated code up to six characters long. This way, it allows the world to share health information using a common language, by including more than 70.000 codes [14]. This organized list, allows:

- easy storage, retrieval and analysis of health information, helping a specialist make evidence-based decisions.
- comparing and sharing information between professionals.
- data comparison in locations across different periods.

6.2 EHR Homepage

In the EHR homepage users choose whether they want to register or log in to the app. The log in page requires a username and a password, in order to identify the user.

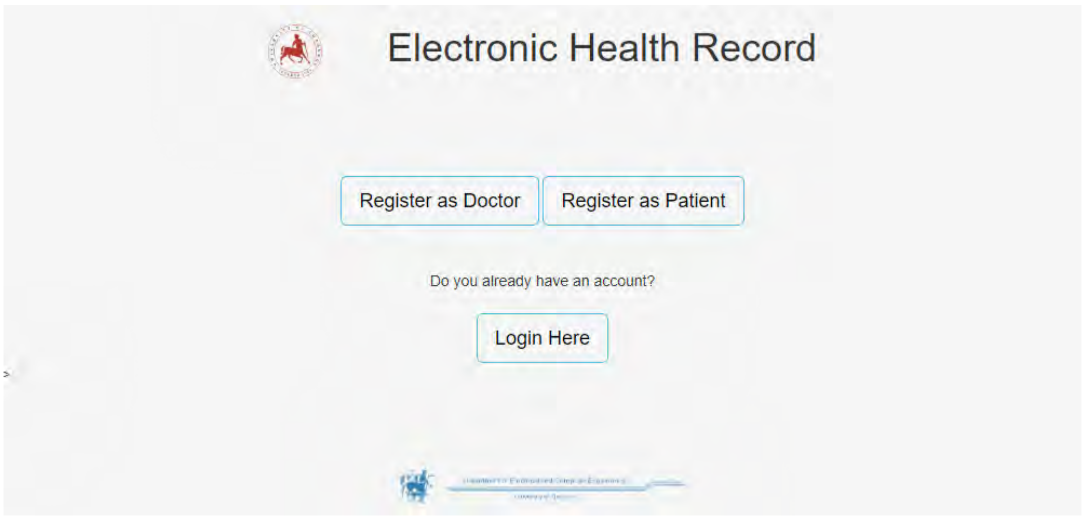


Figure 6.1: EHR - Homepage

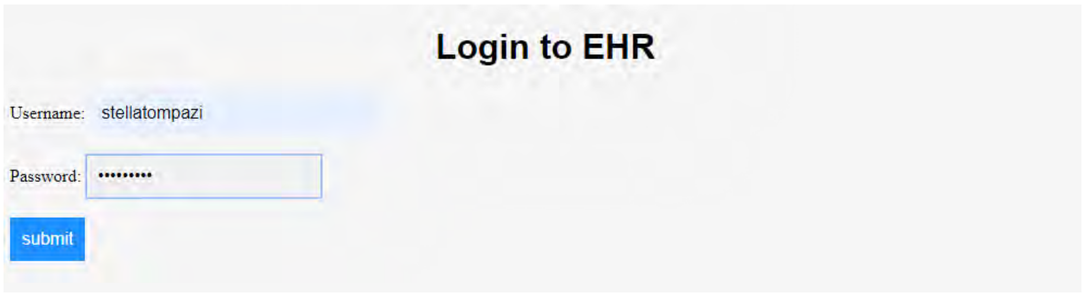


Figure 6.2: EHR - Patient - Log In

6.2.1 Users

The users of the application can be identified either as doctors or patients. The main difference between them, is the fact that patients can not interact with the data of medical records. In order to have two different types of users, there were built to models, one for each type. Regardless the type, each object is related to a built-in User using one-to-one connection, which is similar to a **ForeignKey** with **unique=True**, although the ”reverse” side of this relation returns a single object.

So, besides the User attributes (username, password, email, first_name, last_name), each user type has the following attributes:

- **Doctor**
  - specialty
  - city
  - address
  - phone

```
class UserProfile(models.Model):
user = models.OneToOneField(User, on_delete=models.CASCADE)

specialties_list = (
    ('Accident_and_Emergency_Medicine',
        'Accident and Emergency Medicine'),
    ('Allergiology', 'Allergiology'),
    ('Anesthetics', 'Anesthetics'),
    ('Cardiology', 'Cardiology'),
    ('Child_Psychiatry', 'Child Psychiatry'),
    ('Clinical_Biology', 'Clinical Biology'),
    ('Clinical_Chemistry', 'Clinical Chemistry'),
    ('Clinical_Neurophysiology', 'Clinical Neurophysiology'),
    ('Dental_Oral_Maxillo-facial_surgery',
        'Dental, Oral and Maxillo-facial Surgery'),
    ('Dermato-Venereology', 'Dermato-Venereology'),
    ('Dermatology', 'Dermatology'),
    ('Endocrinology', 'Endocrinology'),
    ('Family_and_general_Medicine', 'Family and General Medicine'),
    ('Fictional_Medical_Specialist', 'Fictional Medical Specialist'),
    ('Gastro-enterologic_Surgery', 'Gastro-enterologic Surgery'),
    ('Gastroenterology', 'Gastroenterology'),
    ('General_Hematology', 'General Hematology'),
    ('General_Practice', 'General Practice'),
    ('General_Surgery', 'General Surgery'),
    ('Geriatrics', 'Geriatrics'),
    ('Immunology', 'Immunology'),
    ('Infectious_Diseases', 'Infectious Diseases'),
    ('Internal_Medicine', 'Internal Medicine'),
    ('Laboratory_Medicine', 'Laboratory Medicine'),
    ('Maxillo-facial_Surgery', 'Maxillo-facial Surgery'),
    ('Microbiology', 'Microbiology'),
    ('Nephrology', 'Nephrology'),
    ('Neuro-psychiatry', 'Neuro-psyciatry'),
    ('Neurosurgery', 'Neurosurgery'),
    ('Nuclear_Medicine', 'Nuclear Medicine'),
    ('Obstetrics_and_Gynecology', 'Obstetrics and Gynecology'),
    ('Occupational_Medicine', 'Occupational Medicine'),
    ('Opthalmology', 'Opthalmology'),
    ('Orthopaedics', 'Orthopaedics'),
    ('Otorhinolaryngology', 'Otorhinolaryngology'),
    ('Paediatrics', 'Paediatrics'),
    ('Paediatrics_Surgery', 'Paediatrics Surgery'),
    ('Pathology', 'Pathology'),
    ('Physical_Medicine_and_Rehabilitation',
        'Physical Medicine and Rehabilitation'),
    ('Plastic_Surgery', 'Plastic Surgery'),
    ('Pneumology', 'Pneumology'),
    ('Podiatric_Surgery', 'Podiatric Surgery'),
    ('Psychiatry', 'Psychiatry'),
    ('Public_health_and_Preventive_Medicine',
        'Public Health and Preventive Medicine'),
    ('Radiation_Oncology', 'Radiation Oncology'),
    ('Respiratory_Medicine', 'Respiratory Medicine'),
    ('Rheumatogoly', 'Rheumatology'),
    ('Stomatology', 'Stomatology'),
    ('Thoragic_Surgery', 'Thoragic Surgery'),
    ('Tropical_Medicine', 'Tropical Medicine'),
    ('Urology', 'Urology'),
    ('Vascular_Surgery', 'Vascular Surgery'),
    ('Venereology', 'Venereology')
)

specialty = models.CharField(max_length=100, choices=specialties_list,
            default='General_Practice')
city = models.CharField(max_length=50, blank=True)
address = models.CharField(max_length=50, blank=True)
phone = models.CharField(max_length=12, blank=True)
```

- **Patient**

  The PatientProfile model is mentioned in Ch. 3.

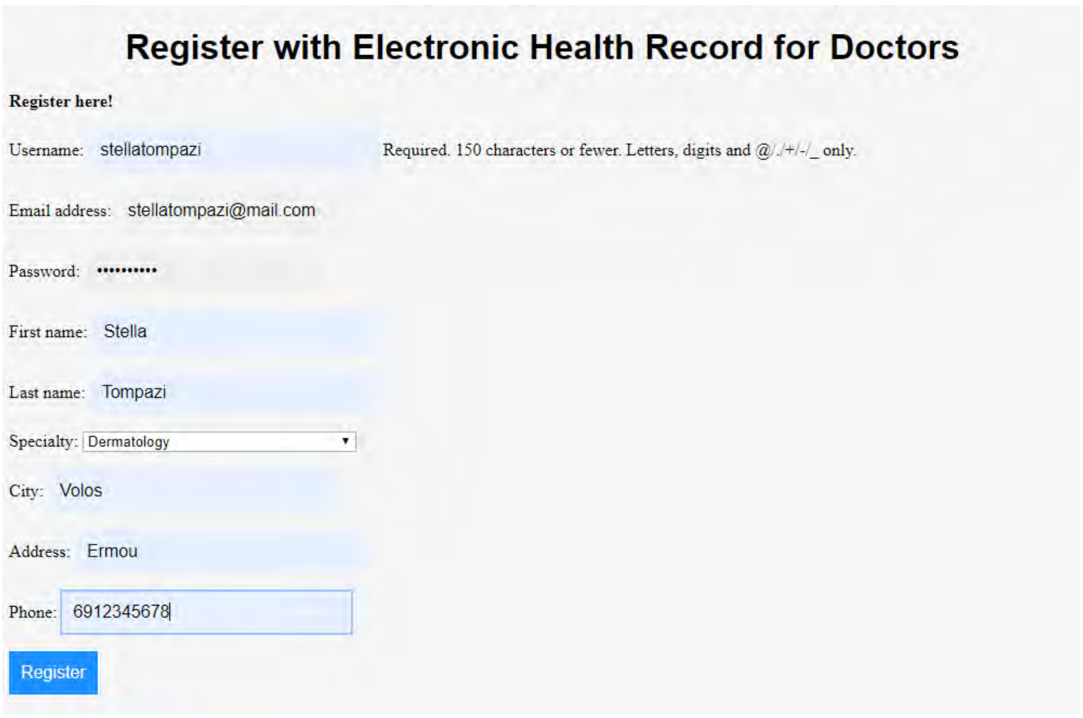
  – birthday
  – social security number
  – sex
  – insurance
  – city
  – address
  – phone

6.3 User: Doctor

Undoubtedly, the most important function for doctor users is the capability to retrieve medical information of patients, as well as save or update data regarding their visits by those patients. They have also been given the opportunity to organize their schedule, while accepting requests for appointments online, through a messaging system. Furthermore, they can include some additional information about the services they provide, such as opening hours and a price list for their services.

6.3.1 Registration for professionals

During registration, professional need to include some basic contact information along with their specialized field.



Register with Electronic Health Record for Doctors

Register here!

Username: stellatompazi Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address: stellatompazi@mail.com

Password: ••••••••••

First name: Stella

Last name: Tompazi

Specialty: Dermatology

City: Volos

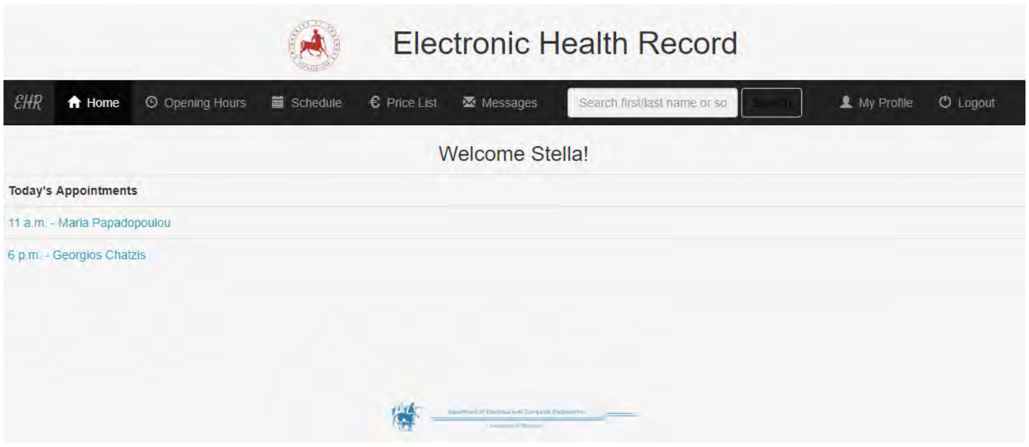
Address: Ermou

Phone: 6912345678

Figure 6.3: EHR - Doctor - Registration

6.3.2 Main page

When registering or logging in as a specialist, the user is redirected to a homepage, where they can view a list of appointments that they have in their schedule for this day. More information can be seen by clicking on an appointment.



Welcome Stella!

Today's Appointments

11 a.m. - Maria Papadopoulou

6 p.m. - Georgios Chatzis

Figure 6.4: EHR - Doctor - Main page

6.3.3 Schedule

The schedule tab includes a monthly calendar, where the doctor can see every appointment that has been scheduled, and also add new appointments.

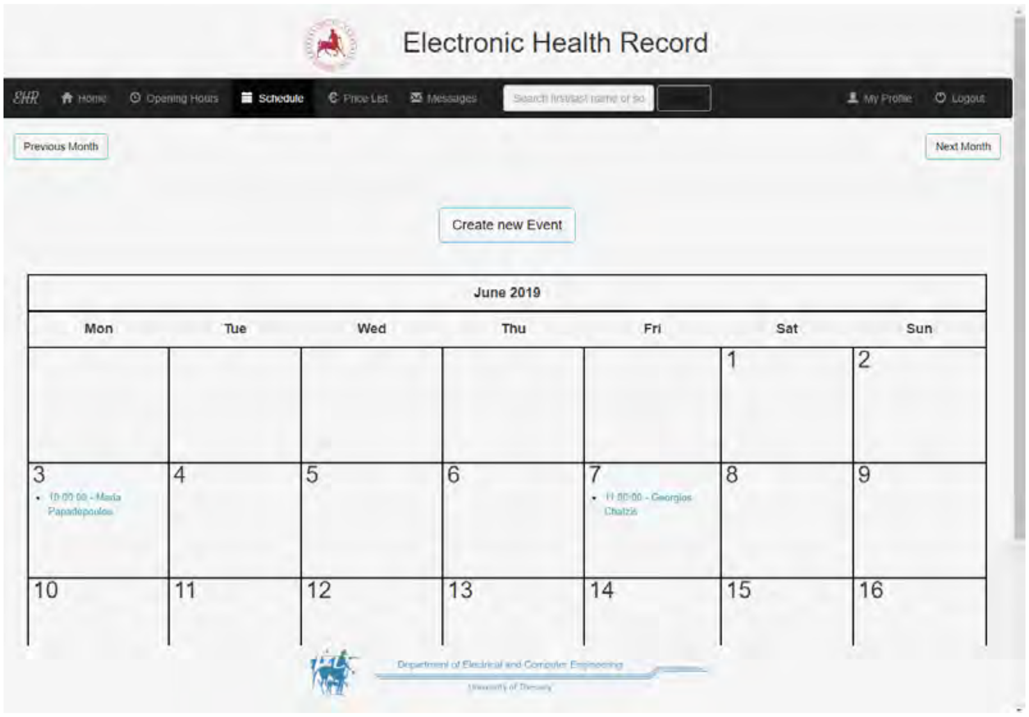

Electronic Health Record
EHR Home Opening Hours Schedule Price List Messages My Profile Logout
Previous Month
Next Month
Create new Event
June 2019
Mon Tue Wed Thu Fri Sat Sun
10:00:00 - Maria Papadopoulou
11:00:00 - Georgios Chatzis
Department of Electrical and Computer Engineering
University of Thessaly

Figure 6.5: EHR - Doctor - Schedule

6.3.3.1 Create/View Event

Doctors can easily add new events in their schedule. Firstly, a search term regarding the patient needs to be given, as for example a first or last name, or a social security number. Right after, only a simple form with information has to be filled.

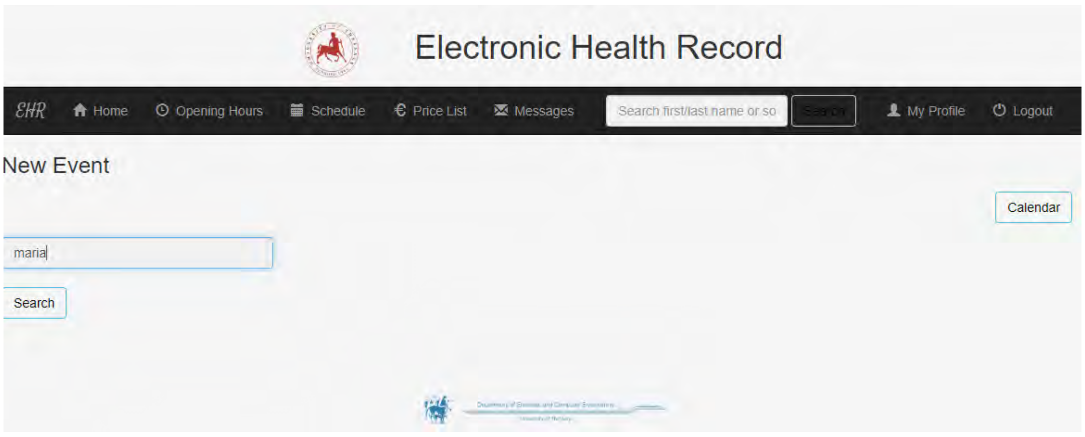
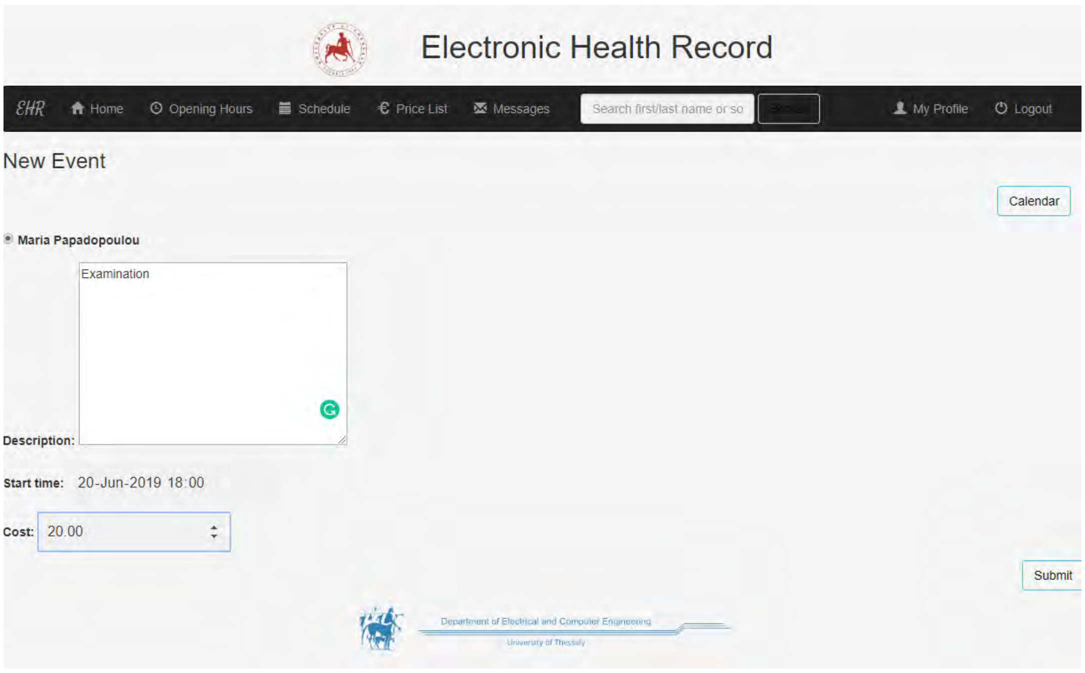

Electronic Health Record
EHR Home Opening Hours Schedule Price List Messages Search first/last name or so Search My Profile Logout
New Event
Calendar
maria
Search

Figure 6.6: EHR - Doctor - New Event



New Event

Maria Papadopoulou

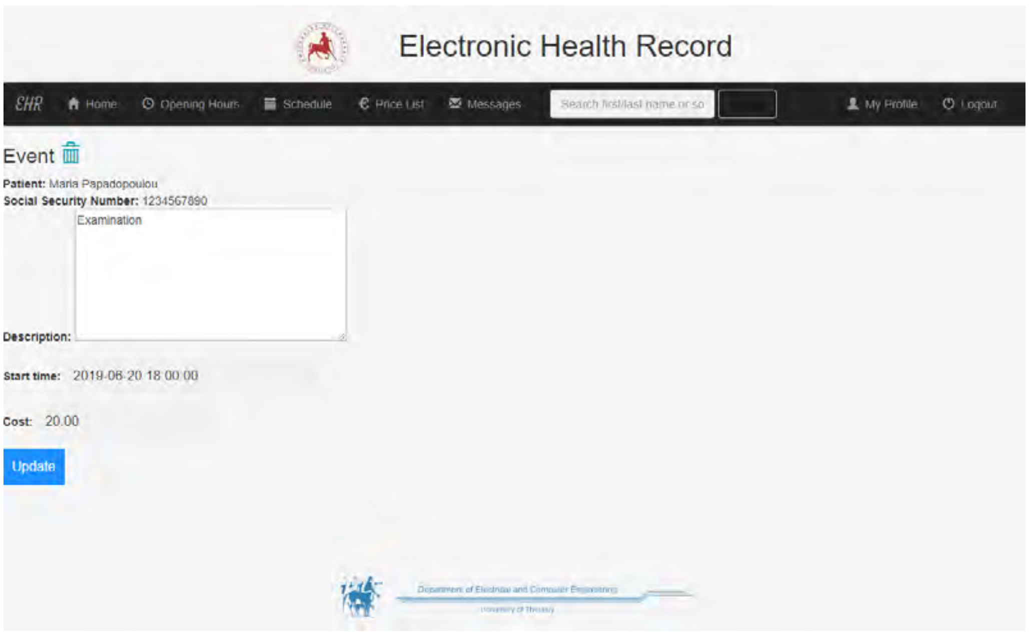
Description: Examination

Start time: 20-Jun-2019 18:00

Cost: 20.00

Figure 6.7: EHR - Doctor - New Event

When clicking on an already scheduled event, through the homepage or the calendar, the information about it will appear, where the event can be completely deleted or updated.



Event

Patient: Maria Papadopoulou

Social Security Number: 1234567890

Description: Examination

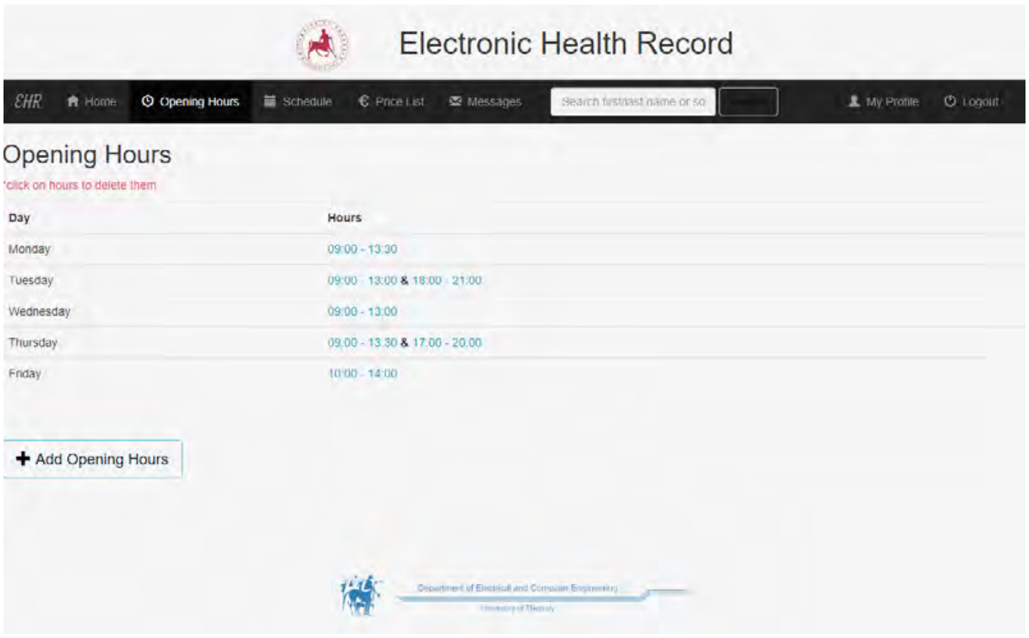
Start time: 2019-06-20 18:00:00

Cost: 20.00

Figure 6.8: EHR - Doctor - Event

6.3.4 Opening Hours

In this tab, professionals can include the hours they are available for visitation. By clicking on **Add Opening Hours** appears a form where proper details are needed to create an object.

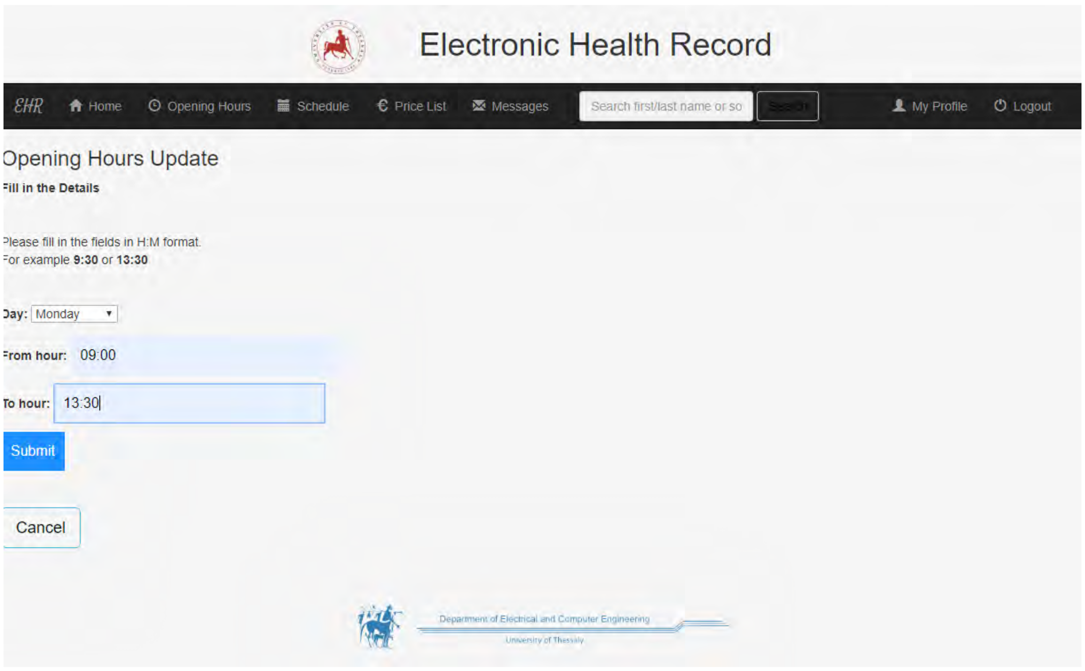


Opening Hours

*click on hours to delete them

Day	Hours
Monday	09:00 - 13:30
Tuesday	09:00 - 13:00 & 18:00 - 21:00
Wednesday	09:00 - 13:00
Thursday	09:00 - 13:30 & 17:00 - 20:00
Friday	10:00 - 14:00

Figure 6.9: EHR - Doctor - Opening Hours



Opening Hours Update

Fill in the Details

Please fill in the fields in H:M format.
For example **9:30** or **13:30**

Day: Monday

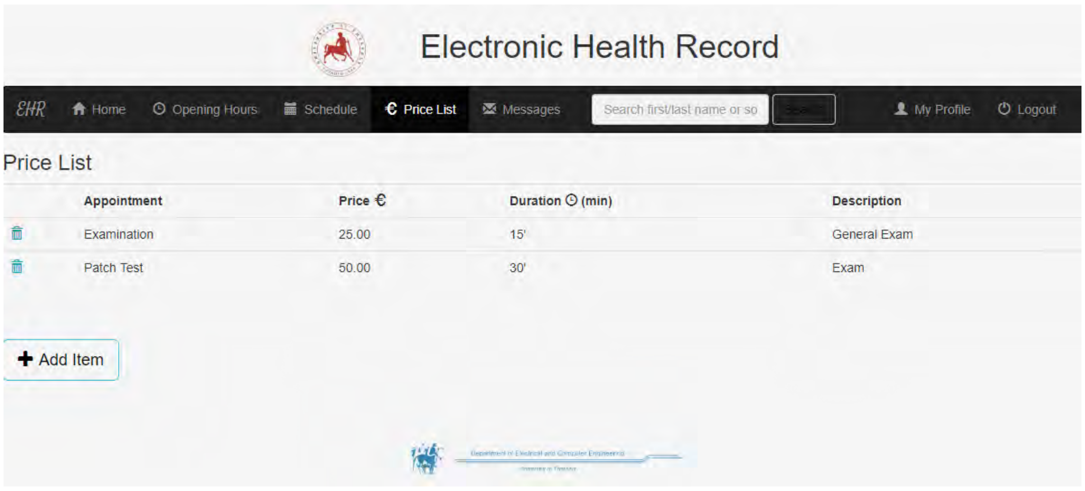
From hour: 09:00

To hour: 13:30

Figure 6.10: EHR - Doctor - Opening Hours Form

6.3.5 Price List

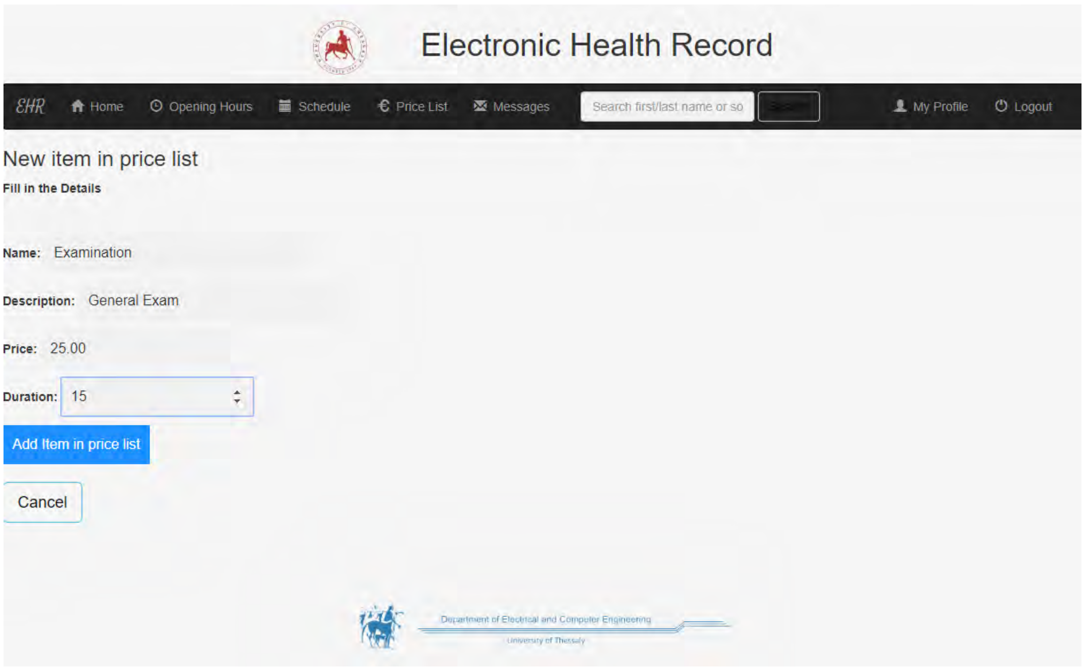
Specialists can add information about the services they provide in this sector, quite similarly like the opening hours tab. Among the details they can include a short name of the service, as well as an extended description, and also, the cost and the approximate duration of such visit.



Price List

	Appointment	Price €	Duration (min)	Description
	Examination	25.00	15'	General Exam
	Patch Test	50.00	30'	Exam

Figure 6.11: EHR - Doctor - Price List



New item in price list

Fill in the Details

Name: Examination

Description: General Exam

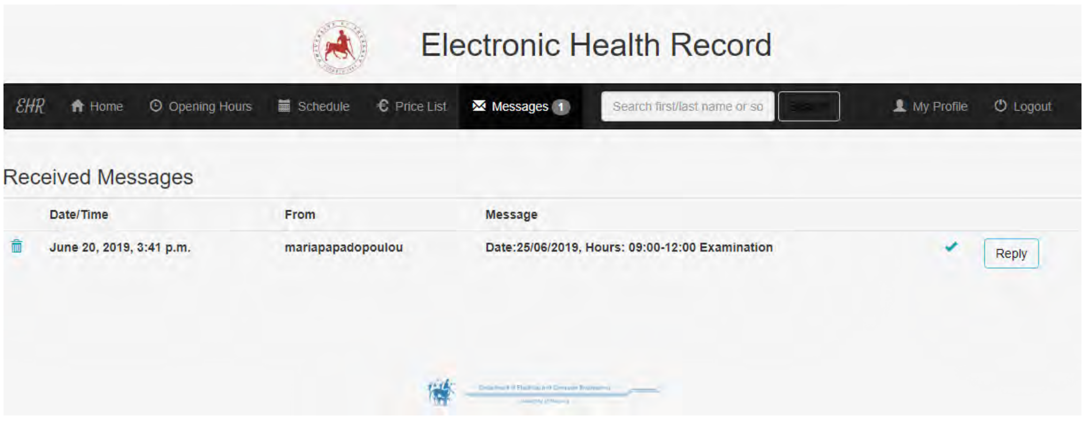
Price: 25.00

Duration: 15

Figure 6.12: EHR - Doctor - Price List Form

6.3.6 Messages

The messages includes an indicator that mentions the number of unread incoming messages, if they exist. When clicking on **Messages** on the navigation bar, two choices appear: inbox and sent. A message that hasn't be marked as read, appears in bold. The reply button pops up a window where the doctor state whether the suggested appointment has been confirmed or declined.



Received Messages

Date/Time	From	Message
June 20, 2019, 3:41 p.m.	mariapapadopoulou	Date:25/06/2019, Hours: 09:00-12:00 Examination

Figure 6.13: EHR - Doctor - Inbox

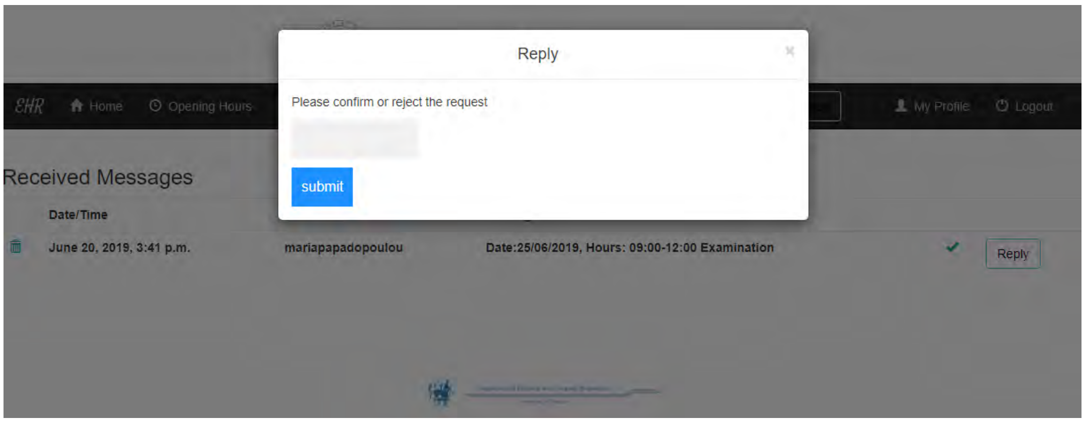
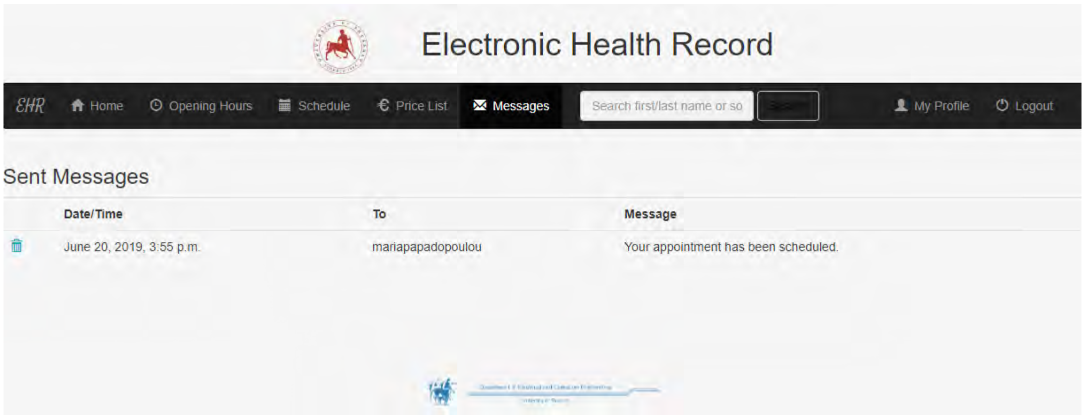


Figure 6.14: EHR - Doctor - Reply



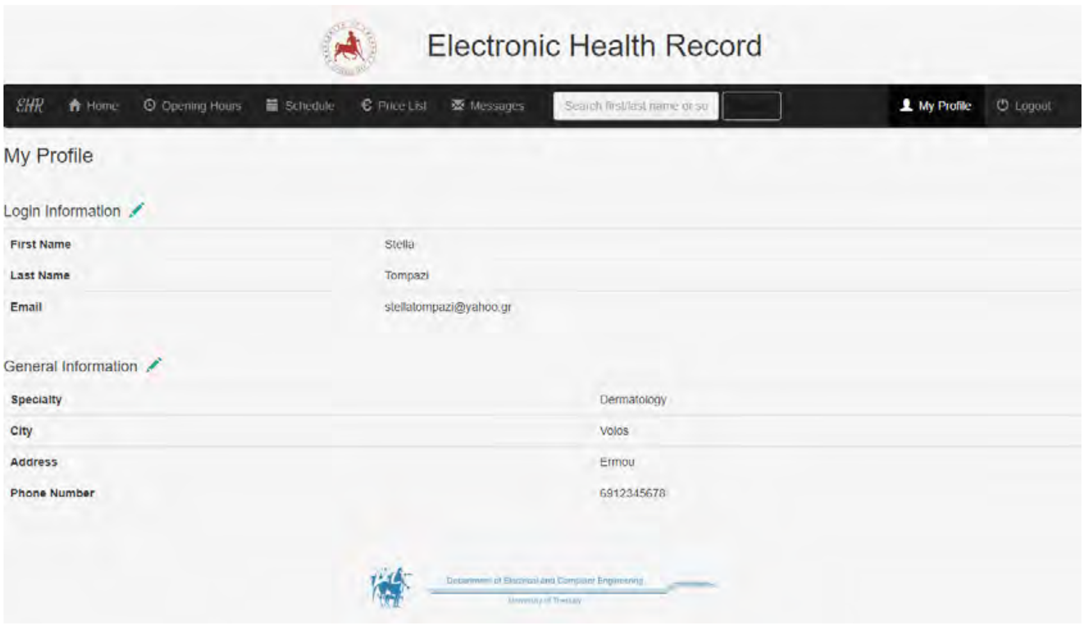
Sent Messages

Date/Time	To	Message
June 20, 2019, 3:55 p.m.	mariapapadopoulou	Your appointment has been scheduled.

Figure 6.15: EHR - Doctor - Sent

6.3.7 My Profile

In **My profile** there are all the information about the user that is currently using the application. These information can be easily updated by clicking on the ”update” pencil buttons.



My Profile

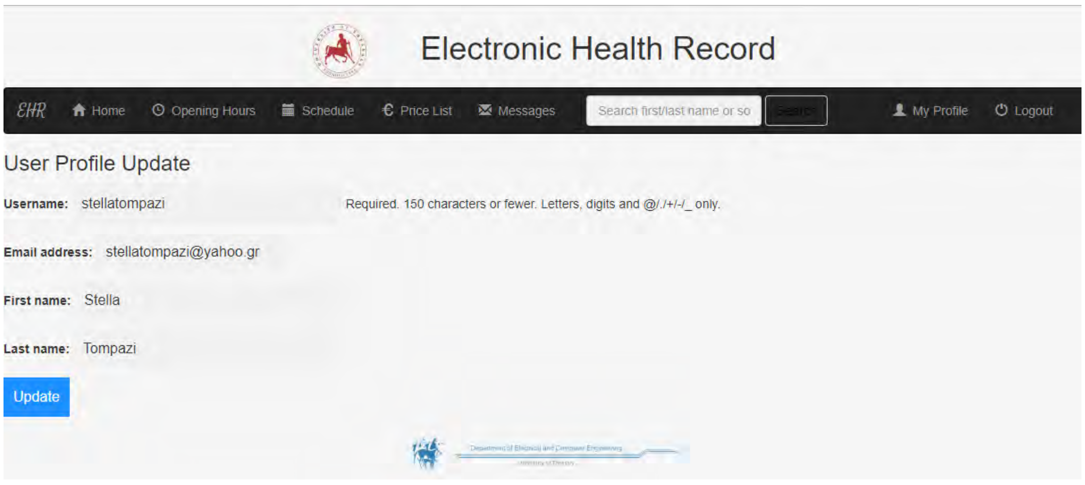
Login Information

First Name	Stella
Last Name	Tompazi
Email	stellatompazi@yahoo.gr

General Information

Specialty	Dermatology
City	Volos
Address	Ermou
Phone Number	6912345678

Figure 6.16: EHR - Doctor - My Profile



User Profile Update

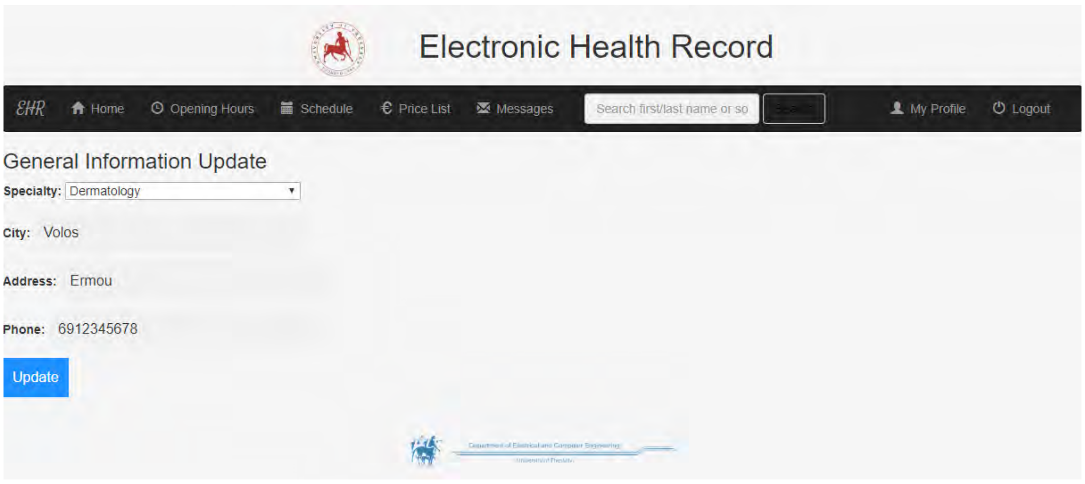
Username: stellatompazi Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email address: stellatompazi@yahoo.gr

First name: Stella

Last name: Tompazi

Figure 6.17: EHR - Doctor - User Profile Update



General Information Update

Specialty: Dermatology

City: Volos

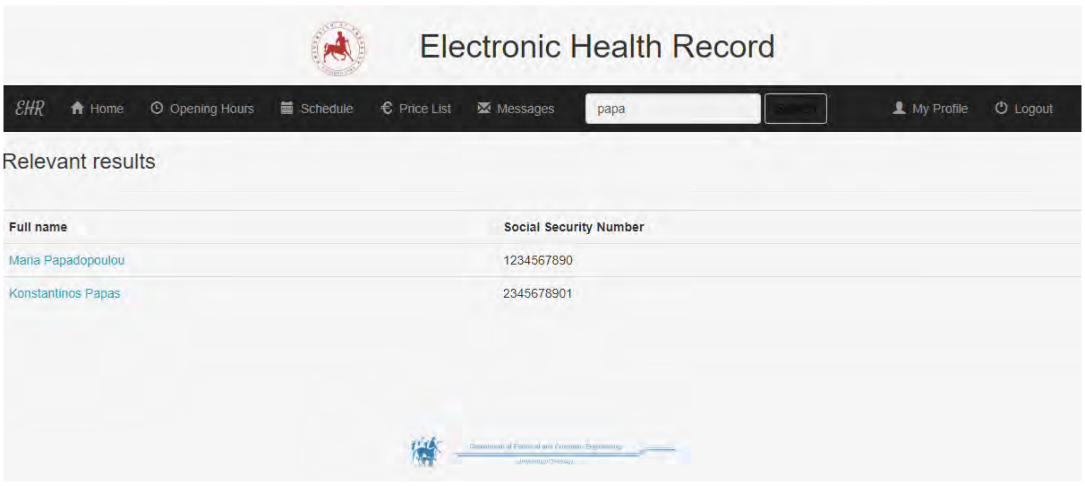
Address: Ermou

Phone: 6912345678

Figure 6.18: EHR - Doctor - Profile Info Update

6.3.8 Search

On the navigation bar there is a search form which is used to find a patient. To do so, a search term has to include either a first/last name or a social security number, full or partial. After that, there is available a list of possible results that can be chosen by clicking.



Relevant results

Full name	Social Security Number
Maria Papadopoulou	1234567890
Konstantinos Papas	2345678901

Figure 6.19: EHR - Doctor - Search

Patient's Profile

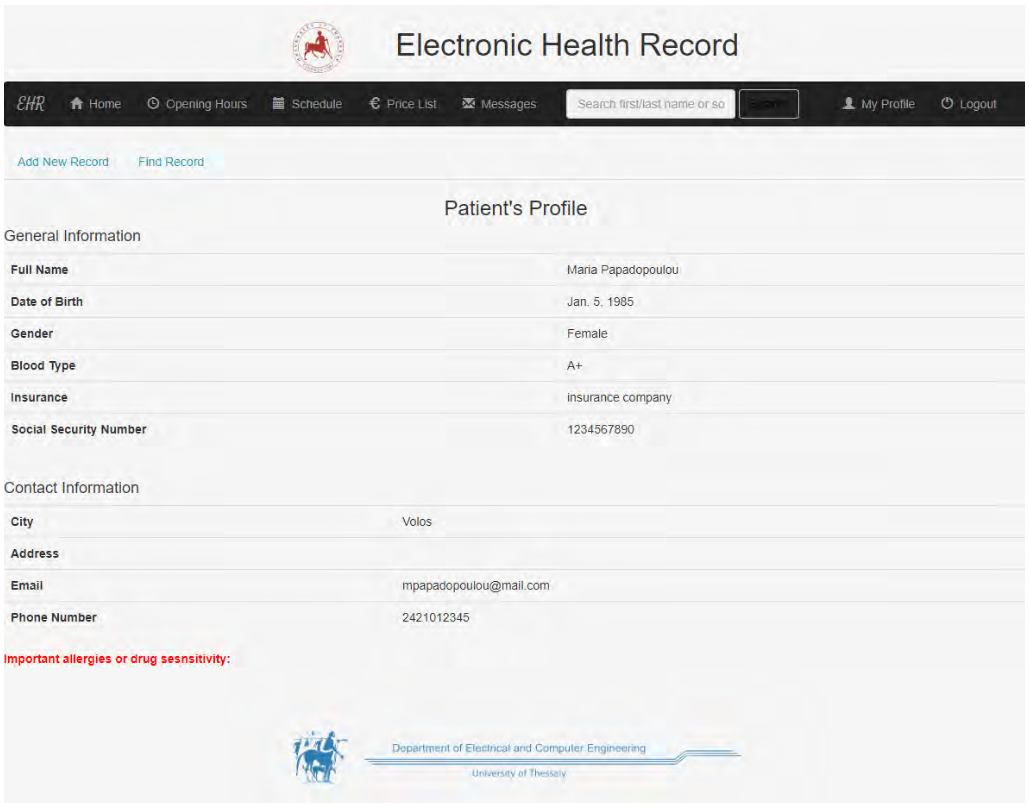
A patient's profile consists of all the information about them. On top, there are two buttons to navigate through the records or create new ones. Each button has three options in order to find/create a diagnosis record, a vaccination record or a surgery record.

6.3.8.1 Add New Record

Each of these options display different forms, whose their steps vary according to their content and purpose.

- **Add Diagnosis Record**

  When creating a new record that sums up a visit, a 3 step form need to be completed. The first step includes information about when the visit was made, what symptoms the patient had, the examination, the prescribed medication and side effects, and even a field where a doctor can add a note which will be visible only to them.



Patient's Profile

General Information

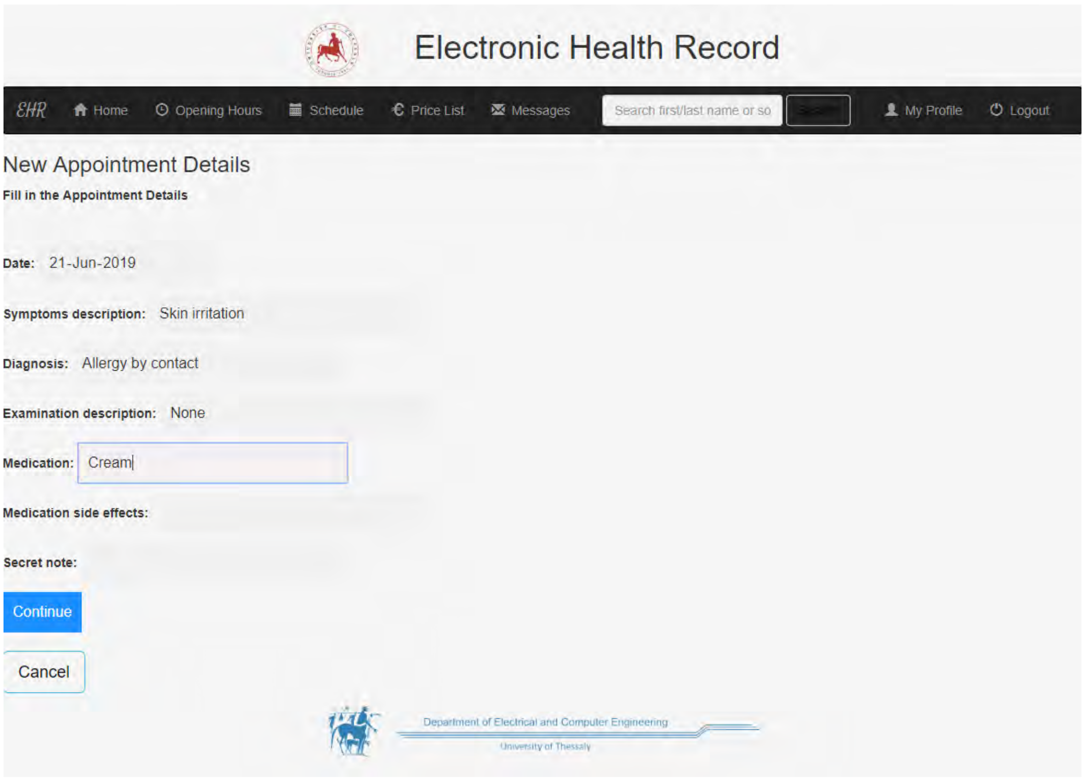
Full Name	Maria Papadopoulou
Date of Birth	Jan. 5, 1985
Gender	Female
Blood Type	A+
Insurance	insurance company
Social Security Number	1234567890

Contact Information

City	Volos
Address	
Email	mpapadopoulou@mail.com
Phone Number	2421012345

Important allergies or drug sesnsitivity:

Figure 6.20: EHR - Doctor - Patient's Profile



New Appointment Details

Fill in the Appointment Details

Date: 21-Jun-2019

Symptoms description: Skin irritation

Diagnosis: Allergy by contact

Examination description: None

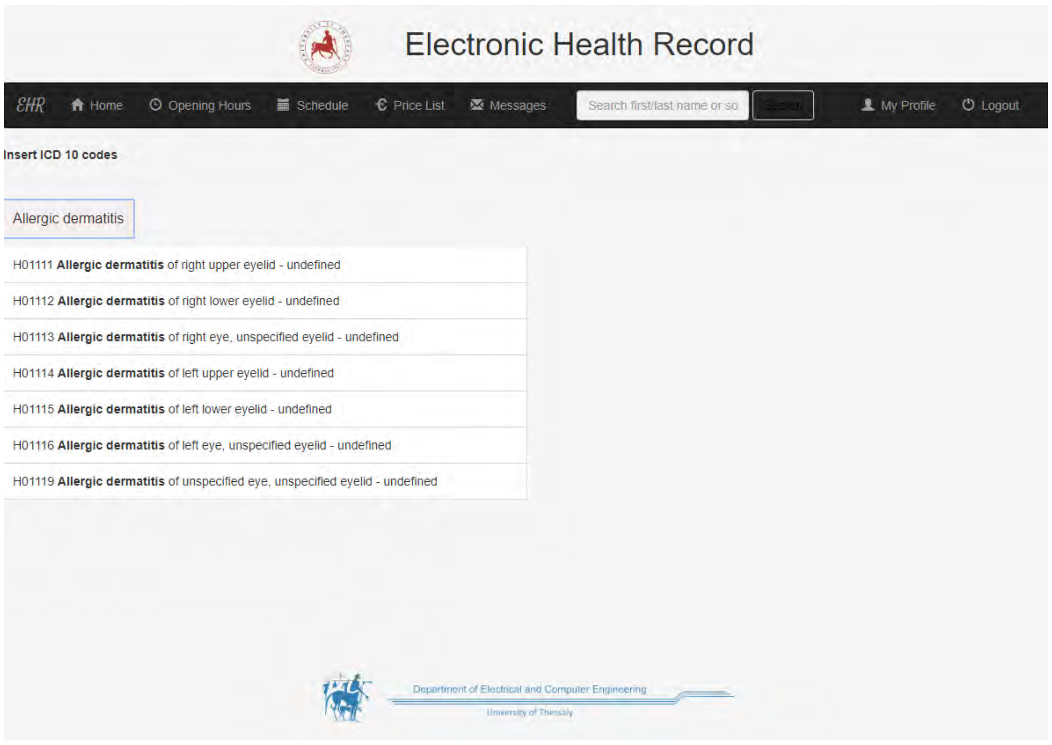
Medication: Cream

Medication side effects:

Secret note:

Figure 6.21: EHR - Doctor - New Diagnosis Record (part 1)

In the next step, one or multiple icd10 codes can be used, to describe the diagnosis according to World Health Organization. As the list of the icd10 codes is overlong, when typing a word of interest, an auto-complete list is created in order to choose.

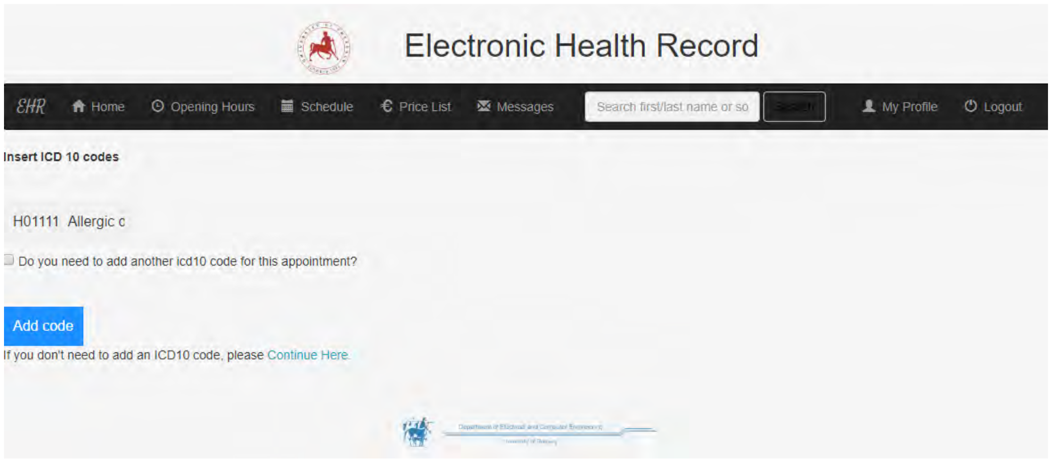


Insert ICD 10 codes

Allergic dermatitis

H01111 **Allergic dermatitis** of right upper eyelid - undefined
H01112 **Allergic dermatitis** of right lower eyelid - undefined
H01113 **Allergic dermatitis** of right eye, unspecified eyelid - undefined
H01114 **Allergic dermatitis** of left upper eyelid - undefined
H01115 **Allergic dermatitis** of left lower eyelid - undefined
H01116 **Allergic dermatitis** of left eye, unspecified eyelid - undefined
H01119 **Allergic dermatitis** of unspecified eye, unspecified eyelid - undefined

Figure 6.22: EHR - Doctor - New Diagnosis Record ICD10 code



Insert ICD 10 codes

H01111 Allergic c

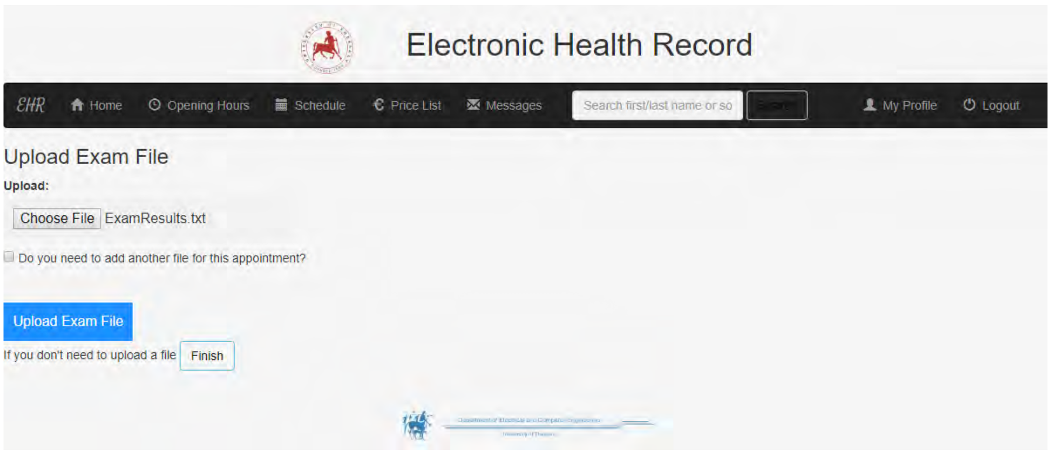
Do you need to add another icd10 code for this appointment?

Add code

If you don't need to add an ICD10 code, please Continue Here.

Figure 6.23: EHR - Doctor - New Diagnosis Record (part 2)

Finally, in the last step, one or more files regarding the visit can be uploaded.



Upload Exam File

Upload:

Choose File ExamResults.txt

Do you need to add another file for this appointment?

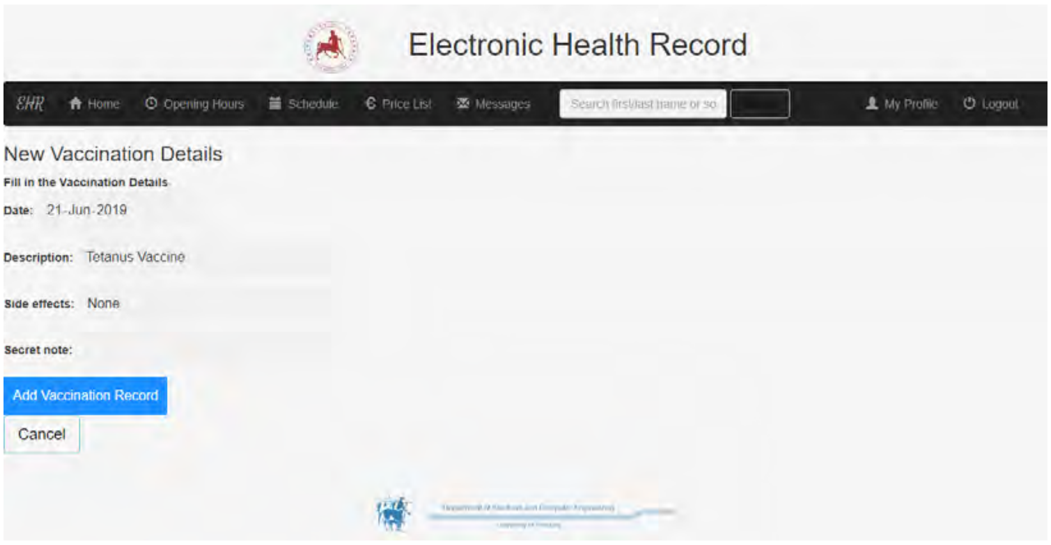
Upload Exam File

If you don't need to upload a file Finish

Figure 6.24: EHR - Doctor - New Diagnosis Record (part 3)

- **Add Vaccination Record**

  Creating a vaccination record only requires the completion of a simple form including the date, a description, side effects and a note.



New Vaccination Details

Fill in the Vaccination Details

Date: 21-Jun-2019

Description: Tetanus Vaccine

Side effects: None

Secret note:

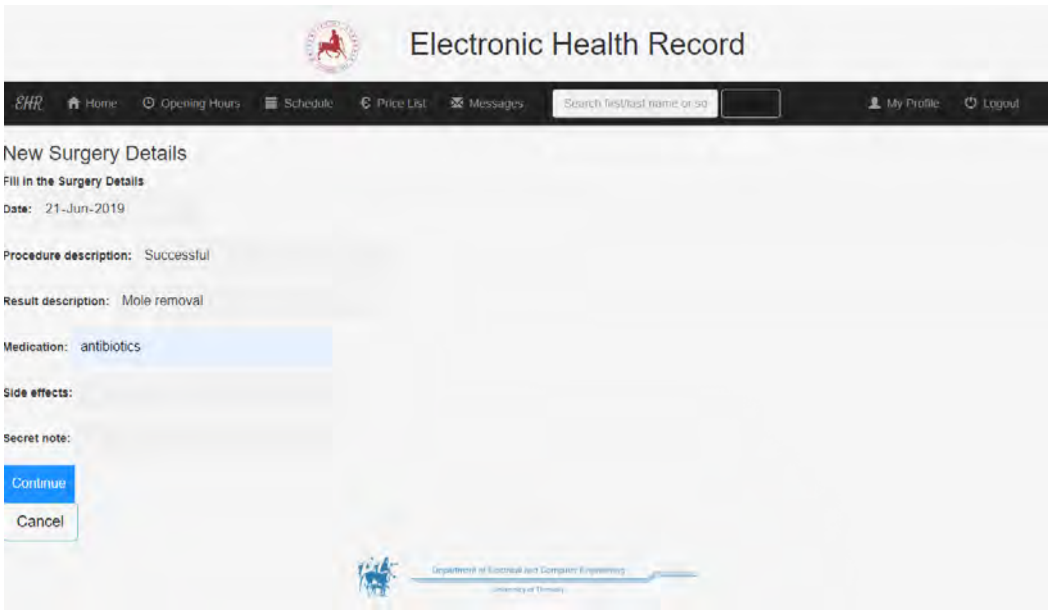
Add Vaccination Record

Cancel

Figure 6.25: EHR - Doctor - New Vaccination Record

- **Add Surgery Record**

  A surgery record form is also quite simple and consisting of two steps. The first one requires the details and the second is for uploading relative files, just like in a diagnosis record.



New Surgery Details

Fill in the Surgery Details

Date: 21-Jun-2019

Procedure description: Successful

Result description: Mole removal

Medication: antibiotics

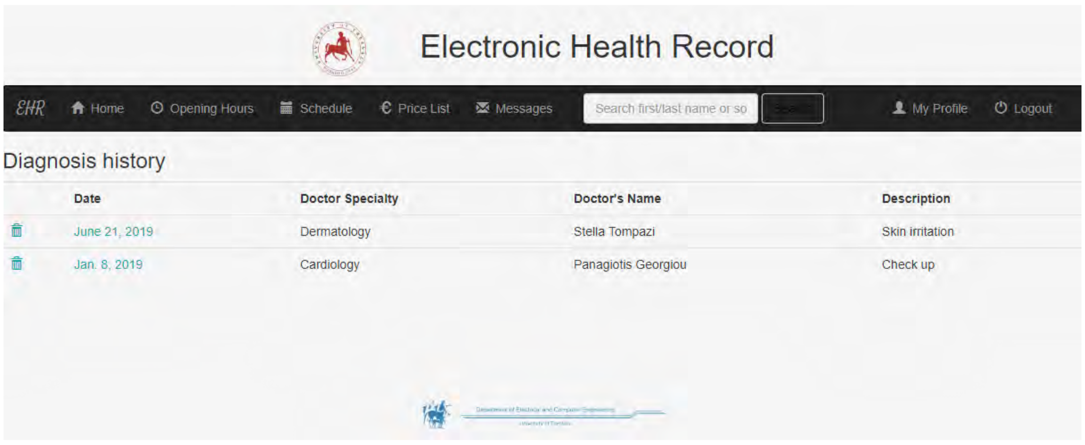
Side effects:

Secret note:

Figure 6.26: EHR - Doctor - New Surgery Record

6.3.8.2 Find Record

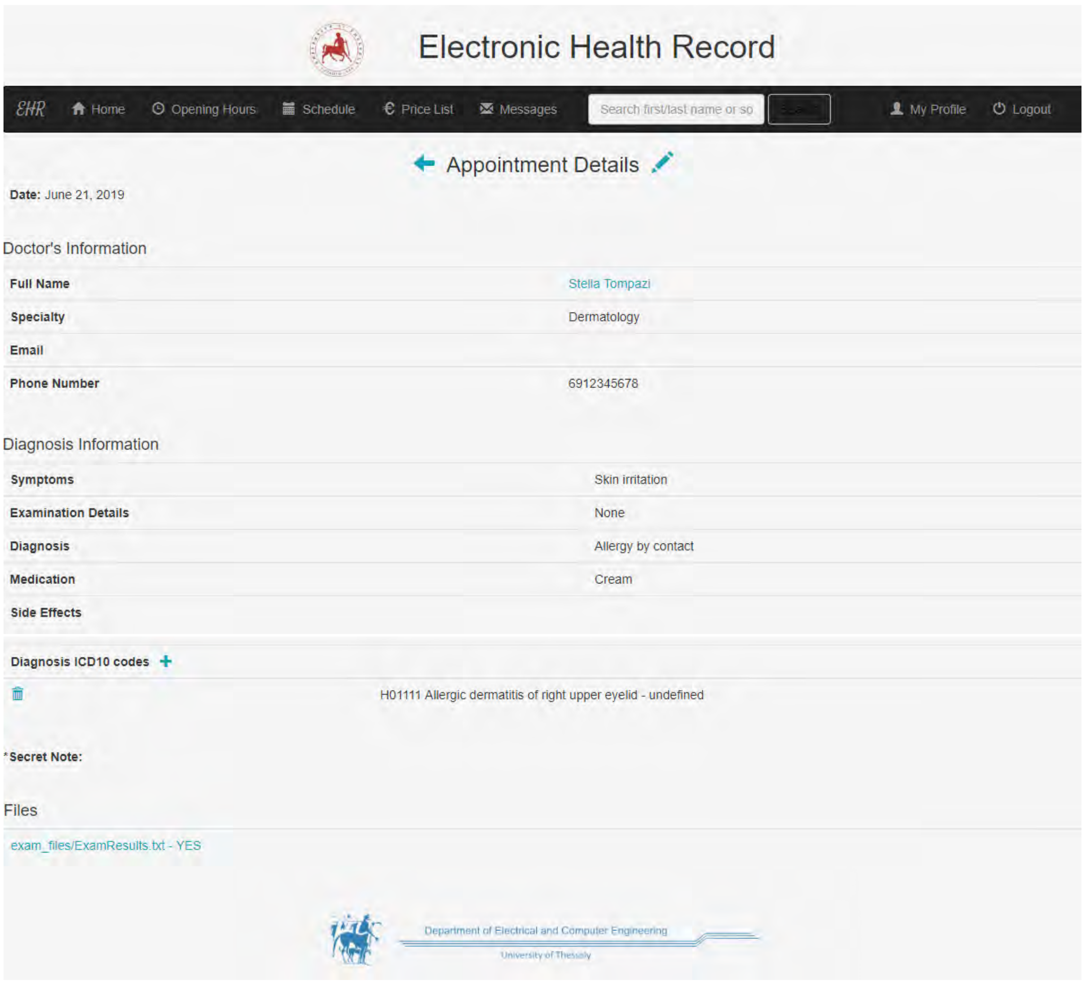
The existing records are also categorized as Diagnosis, Vaccination and Surgery history. All of them, when selected, will display a list of the related records and when they were inserted, as well as some basic information of the record. To view more details, a specialist can just click on the desired record. Every relevant file can be downloaded easily. Furthermore, it is possible for someone to take a look on the details of the doctor who created the record, in case of wishing to contact them. Every detail can be updated any time.



Diagnosis history

Date	Doctor Specialty	Doctor's Name	Description
June 21, 2019	Dermatology	Stella Tompazi	Skin irritation
Jan. 8, 2019	Cardiology	Panagiotis Georgiou	Check up

Figure 6.27: EHR - Doctor - Diagnosis Records List



Date: June 21, 2019

Doctor's Information

Full Name	Stella Tompazi
Specialty	Dermatology
Email	
Phone Number	6912345678

Diagnosis Information

Symptoms	Skin irritation
Examination Details	None
Diagnosis	Allergy by contact
Medication	Cream
Side Effects	

Diagnosis ICD10 codes

H01111 Allergic dermatitis of right upper eyelid - undefined

***Secret Note:**

Files

exam_files/ExamResults.txt - YES

Department of Electrical and Computer Engineering
University of Thessaly

Figure 6.28: EHR - Doctor - Diagnosis Record

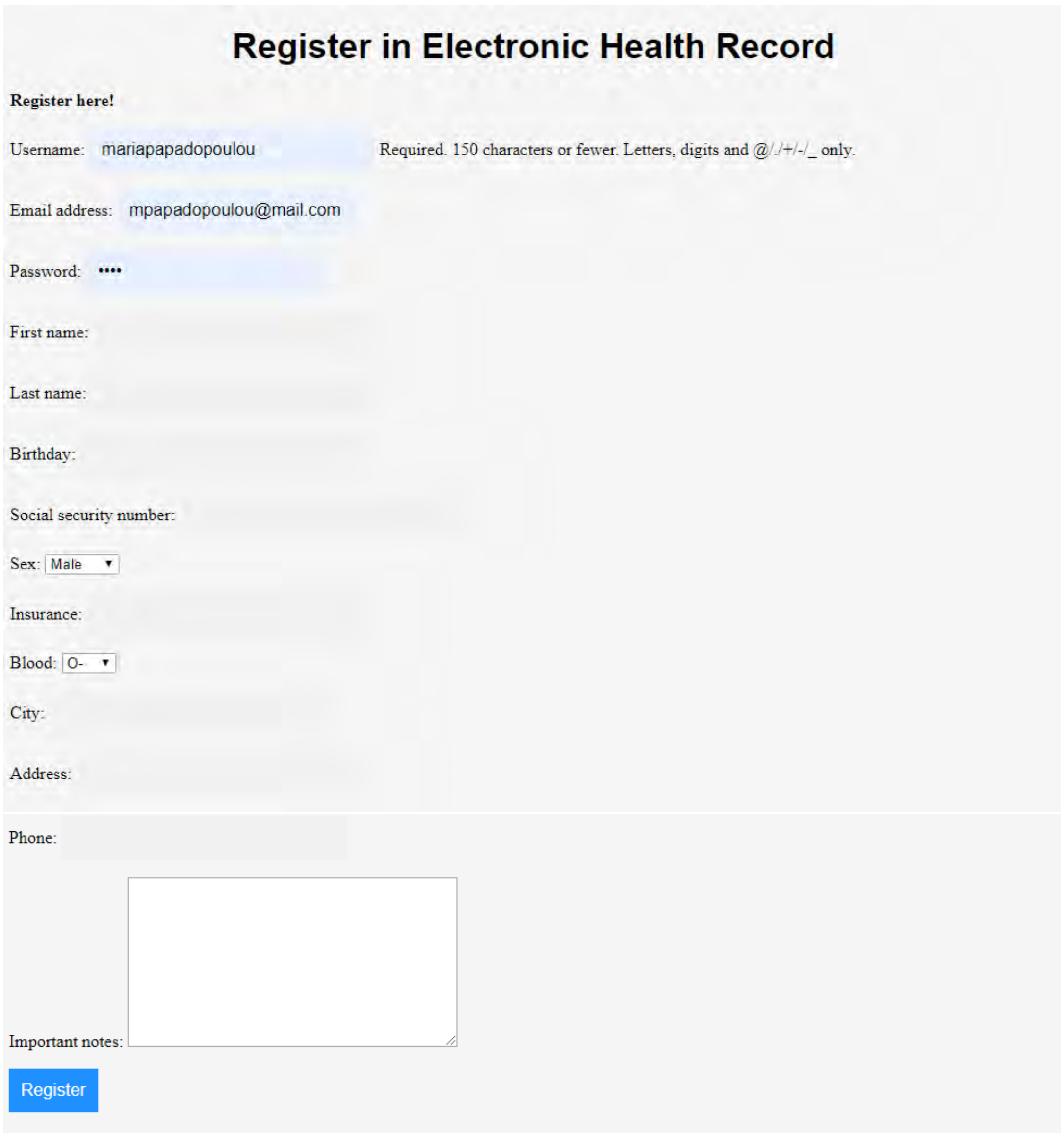
Vaccination and surgery records are being handled the exact same way.

6.4 User: Patient

EHR isn't just an application for doctors. It is useful for patients too. Except for having access to their own health records, although not being able to make any changes obviously, patients can search for doctors, request an appointment through the app, as well as mark doctors as ”favorites”. This bookmark system allows them to have an easy access to the doctors information, such as their opening hours and price list.

6.4.1 Registration for patient

Registration for patients require some more data that would be probably useful in any case. So, along with contact information, users need to include some information about themselves like their date of birth, type of blood, social security number, insurance, etc.


Register in Electronic Health Record

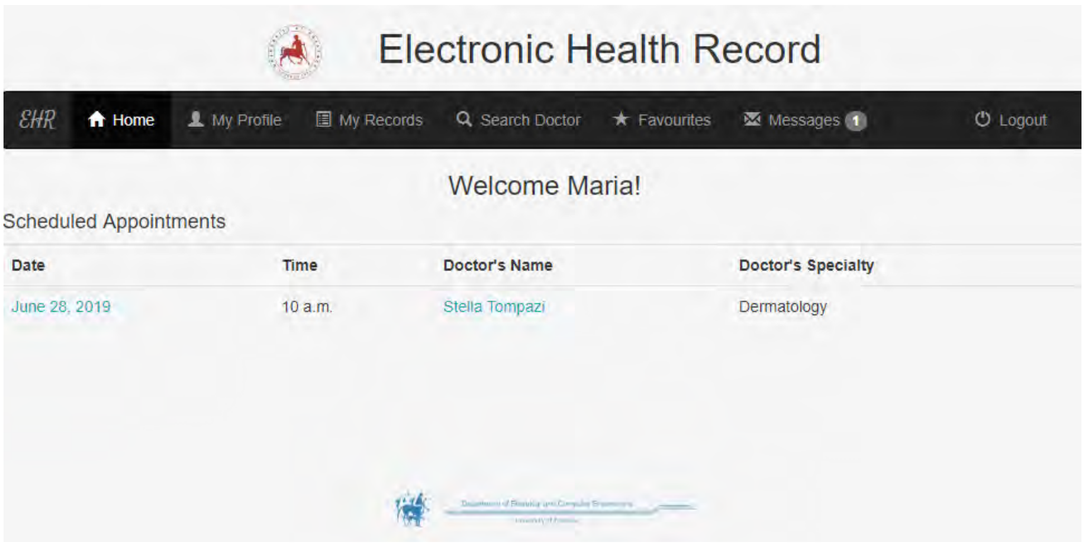
Register here!

Username:	mariapapadopoulou	Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
Email address:	mpapadopoulou@mail.com	
Password:	••••	
First name:		
Last name:		
Birthday:		
Social security number:		
Sex:	Male	
Insurance:		
Blood:	O-	
City:		
Address:		
Phone:		
Important notes:		

Figure 6.29: EHR - Patient - Request Appointment

6.4.2 Homepage

Registering or logging in as patient redirects the user to a homepage, which displays their future scheduled appointments. Through this list they are able to see the details of each appointment, regarding the doctor, the description and the cost.

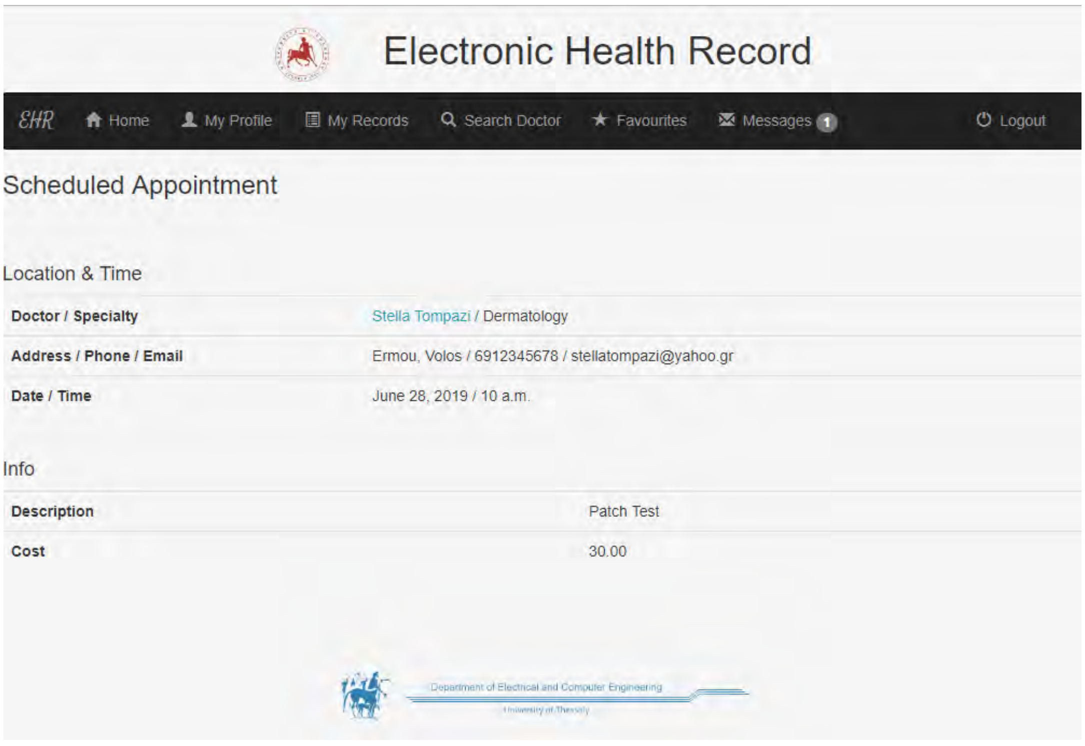


Welcome Maria!

Scheduled Appointments

Date	Time	Doctor's Name	Doctor's Specialty
June 28, 2019	10 a.m.	Stella Tompazi	Dermatology

Figure 6.30: EHR - Patient - Homepage



Scheduled Appointment

Location & Time

Doctor / Specialty	Stella Tompazi / Dermatology
Address / Phone / Email	Ermou, Volos / 6912345678 / stellatompazi@yahoo.gr
Date / Time	June 28, 2019 / 10 a.m.

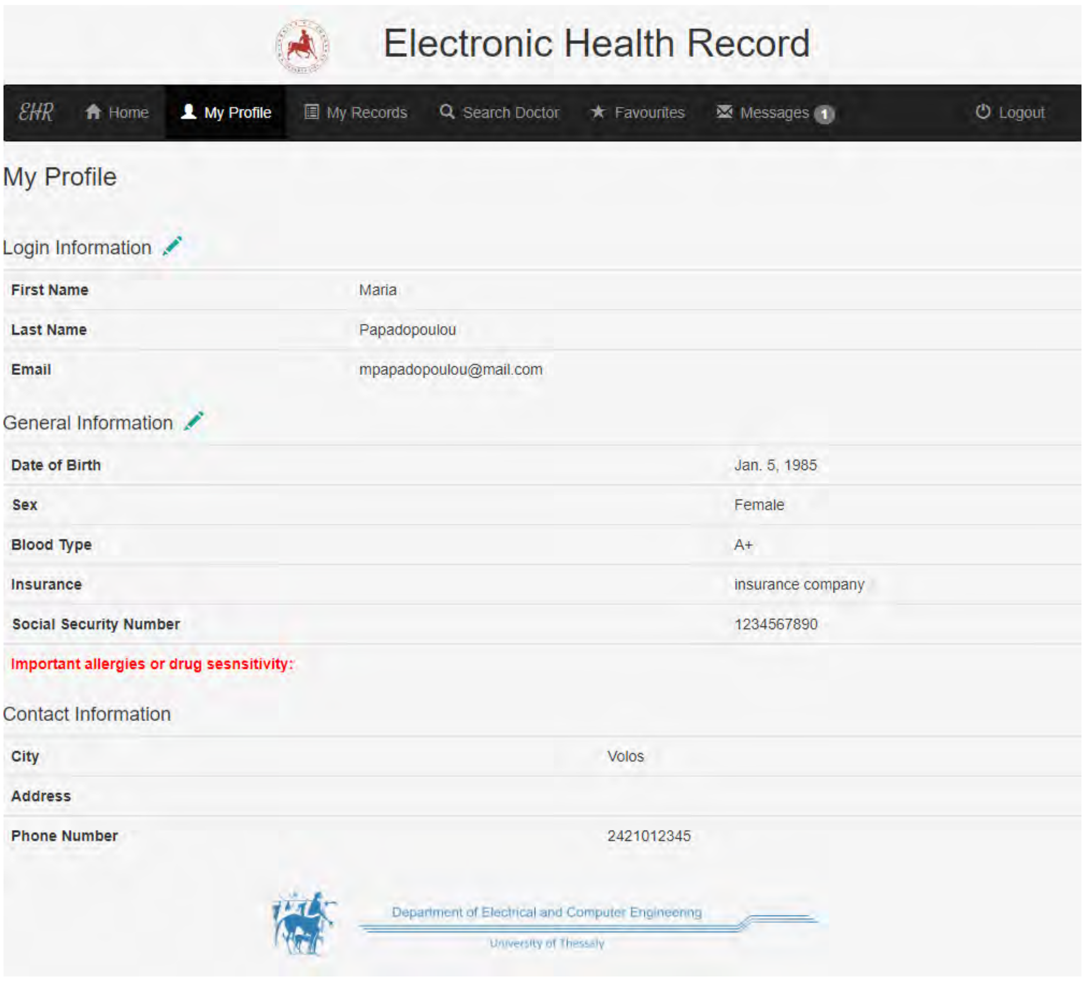
Info

Description	Patch Test
Cost	30.00

Figure 6.31: EHR - Patient - Scheduled Appointment

6.4.3 My Profile

This section includes all the user and personal information of the patient. Everything can be updated by clicking on the edit buttons, just like doctor users.



My Profile

Login Information

First Name	Maria
Last Name	Papadopoulou
Email	mpapadopoulou@mail.com

General Information

Date of Birth	Jan. 5, 1985
Sex	Female
Blood Type	A+
Insurance	insurance company
Social Security Number	1234567890

Important allergies or drug sesnsitivity:

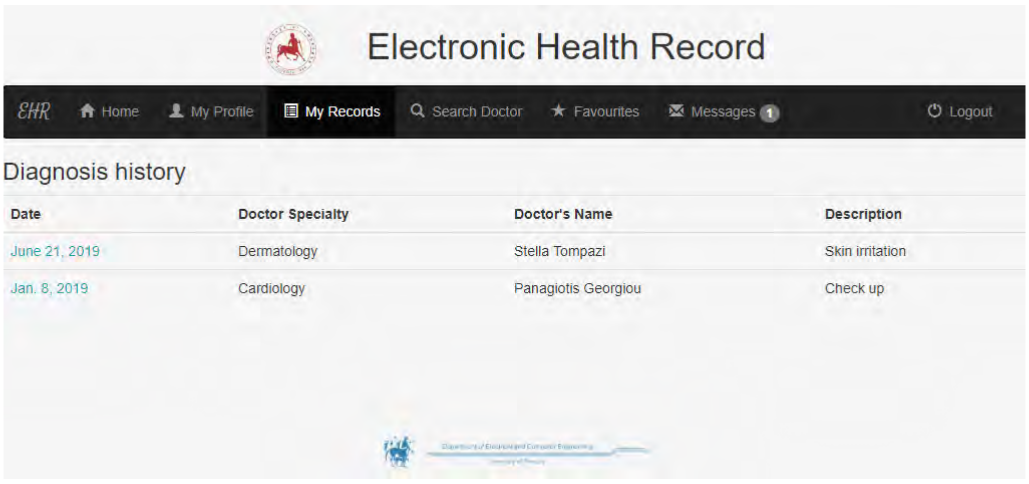
Contact Information

City	Volos
Address	
Phone Number	2421012345

Figure 6.32: EHR - Patient - My Profile

6.4.4 My Records

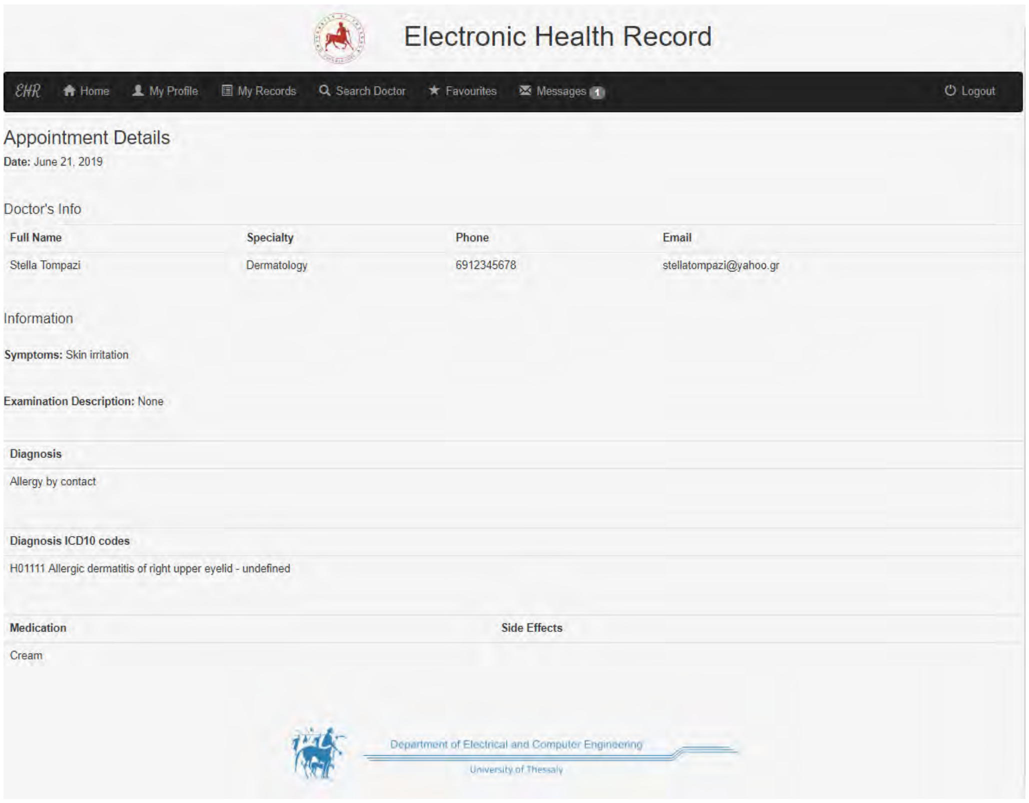
By clicking on **My records** on the navigation bar, a drop-down list shows, which consists of three choices: Diagnosis, Surgery and Vaccination History. Each option displays the a related list of records. This list is actually a table that includes some basic information, the name and specialty of the professional that added the record and a short description. Further information can also be accessed for each record.



Diagnosis history

Date	Doctor Specialty	Doctor's Name	Description
June 21, 2019	Dermatology	Stella Tompazi	Skin irritation
Jan. 8, 2019	Cardiology	Panagiotis Georgiou	Check up

Figure 6.33: EHR - Patient - Diagnosis History List



Appointment Details
Date: June 21, 2019

Doctor's Info

Full Name	Specialty	Phone	Email
Stella Tompazi	Dermatology	6912345678	stellatompazi@yahoo.gr

Information

Symptoms: Skin irritation

Examination Description: None

Diagnosis
Allergy by contact

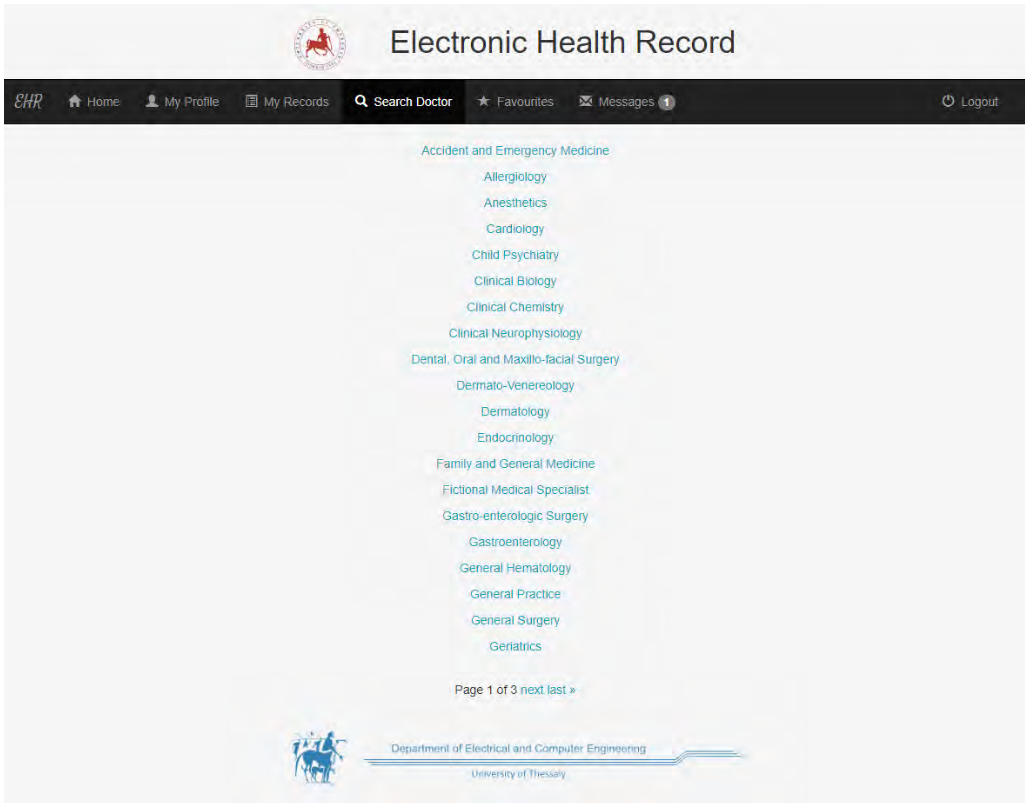
Diagnosis ICD10 codes
H01111 Allergic dermatitis of right upper eyelid - undefined

Medication	Side Effects
Cream	

Figure 6.34: EHR - Patient - Diagnosis Record

6.4.5 Search Doctor

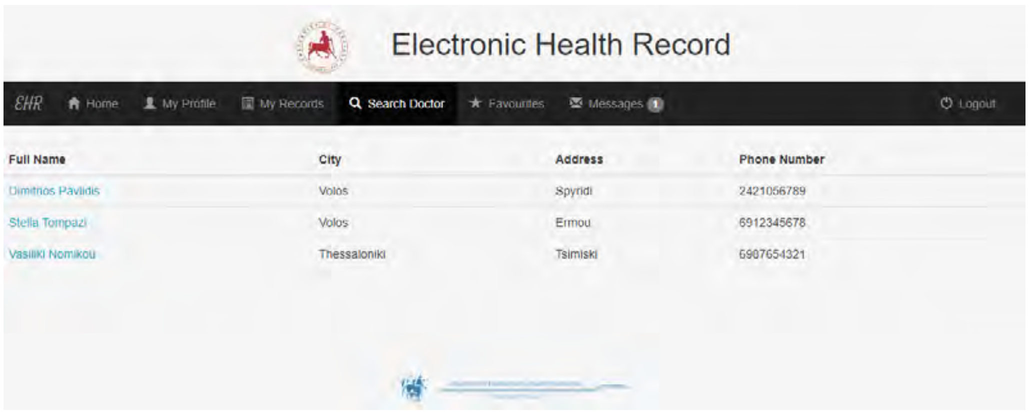
This tab displays a list of professional specialties in descending order. When selecting a specialty, a corresponding list of available doctors will appear. From this list, there can be found further information for each specialist, regarding their contact information, the opening hours and the price list, if they have been provided. Furthermore, through the page with the doctor's information, a user can bookmark that specialist as their favourite by clicking the star on the top, and even request an appointment. To request an appointment, the user has to submit a simple pop up form where they suggest a convenient date and time period, as well as a preferred type of appointment.



Accident and Emergency Medicine
Allergiology
Anesthetics
Cardiology
Child Psychiatry
Clinical Biology
Clinical Chemistry
Clinical Neurophysiology
Dental, Oral and Maxillo-facial Surgery
Dermato-Venereology
Dermatology
Endocrinology
Family and General Medicine
Fictional Medical Specialist
Gastro-enterologic Surgery
Gastroenterology
General Hematology
General Practice
General Surgery
Geriatrics

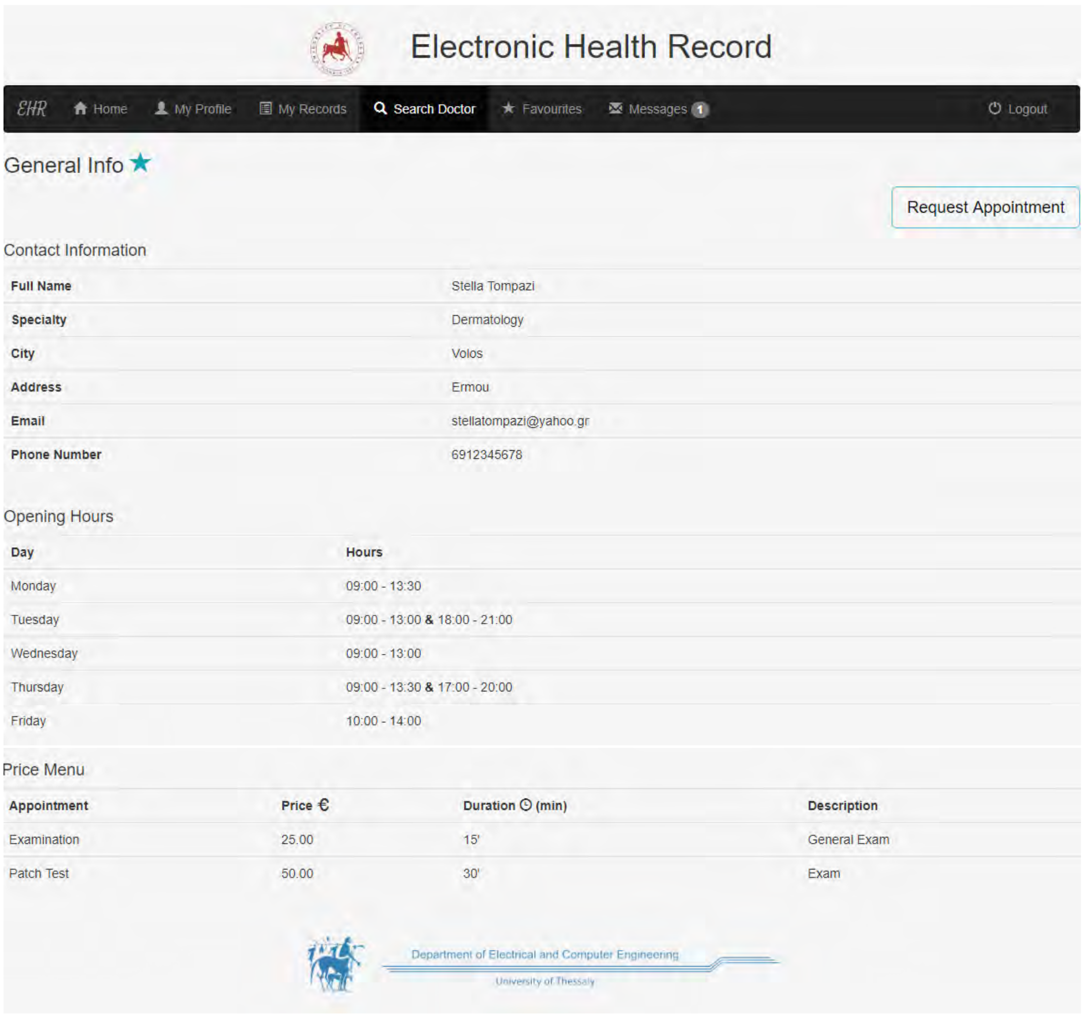
Page 1 of 3 next last »

Figure 6.35: EHR - Patient - Specialties List



Full Name	City	Address	Phone Number
Dimitrios Pavlidis	Volos	Spyridi	2421056789
Stella Tompazi	Volos	Ermou	6912345678
Vasiliki Nomikou	Thessaloniki	Tsimiski	6987654321

Figure 6.36: EHR - Patient - List of Doctors



General Info ★

Contact Information

Full Name	Stella Tompazi
Specialty	Dermatology
City	Volos
Address	Ermou
Email	stellatompazi@yahoo.gr
Phone Number	6912345678

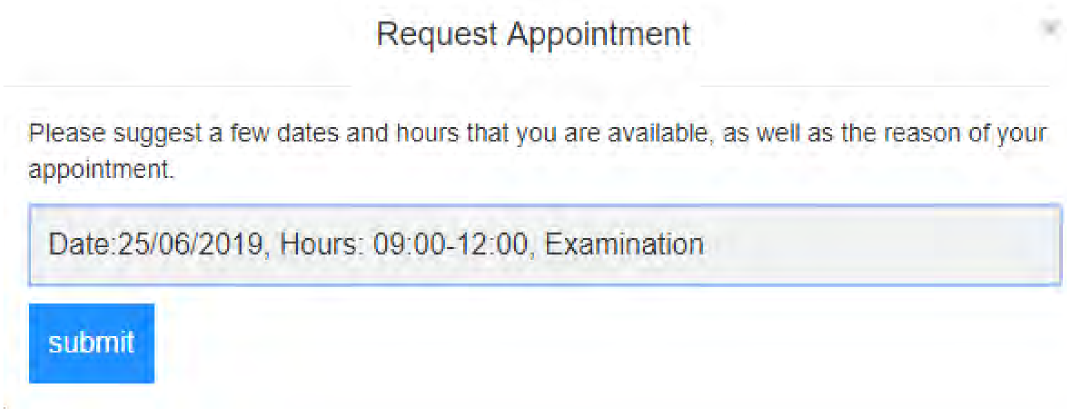
Opening Hours

Day	Hours
Monday	09:00 - 13:30
Tuesday	09:00 - 13:00 & 18:00 - 21:00
Wednesday	09:00 - 13:00
Thursday	09:00 - 13:30 & 17:00 - 20:00
Friday	10:00 - 14:00

Price Menu

Appointment	Price €	Duration (min)	Description
Examination	25.00	15'	General Exam
Patch Test	50.00	30'	Exam

Figure 6.37: EHR - Patient - List of Doctors



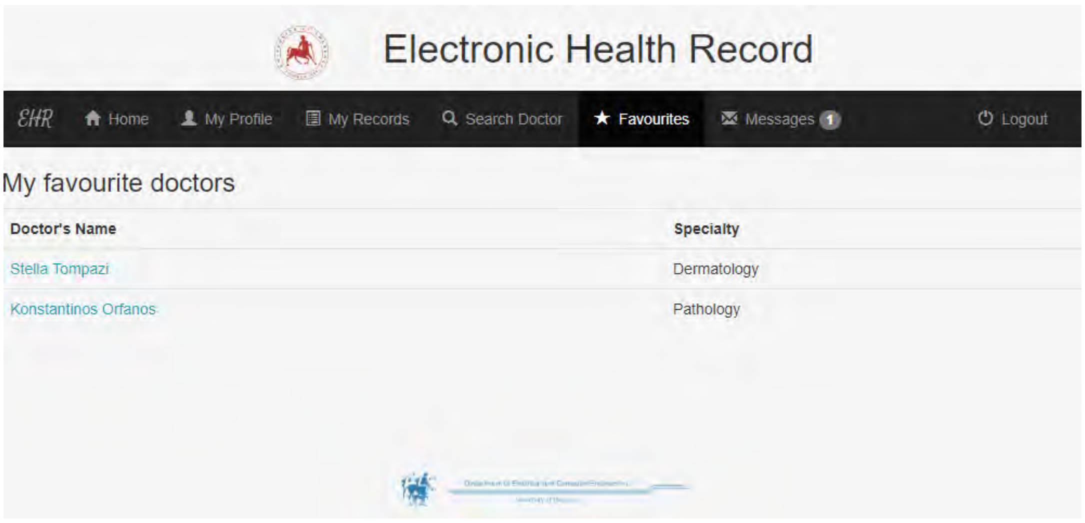
Please suggest a few dates and hours that you are available, as well as the reason of your appointment.

Date:25/06/2019, Hours: 09:00-12:00, Examination

Figure 6.38: EHR - Patient - Request Appointment

6.4.6 Favourites

All specialists marked as ”favourite” is presented in this page, so the user can have immediate access to their information and be able to request an appointment very easily.



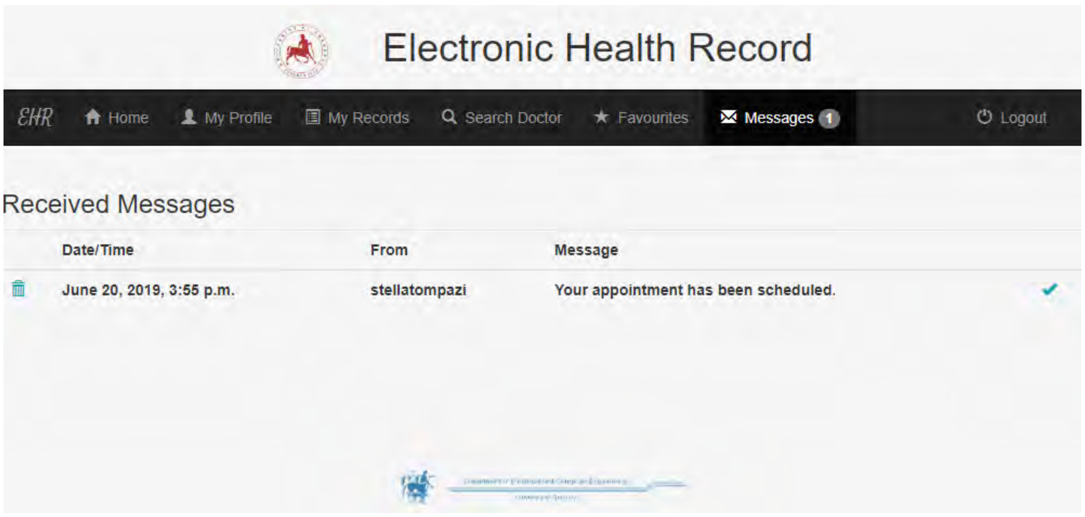
My favourite doctors

Doctor's Name	Specialty
Stella Tompazi	Dermatology
Konstantinos Orfanos	Pathology

Figure 6.39: EHR - Patient - Favourites

6.4.7 Messages

Clicking on Messages drops a list of two choices, inbox and sent, where a user can see their messages, mark them as read or delete them. Unread messages appear in bold.



Received Messages

	Date/Time	From	Message	
🗑	**June 20, 2019, 3:55 p.m.**	**stellatompazi**	**Your appointment has been scheduled.**	✓

Figure 6.40: EHR - Patient - Inbox

Chapter 7

Implementation

Any Django project consists of one or more apps, as explained in Chapter 3. And any Django app includes some standard **.py** files. The most important ones, that include most of the code that is written by the developer, are:

- forms.py
- models.py
- views.py
- templates (folder with HTML files)

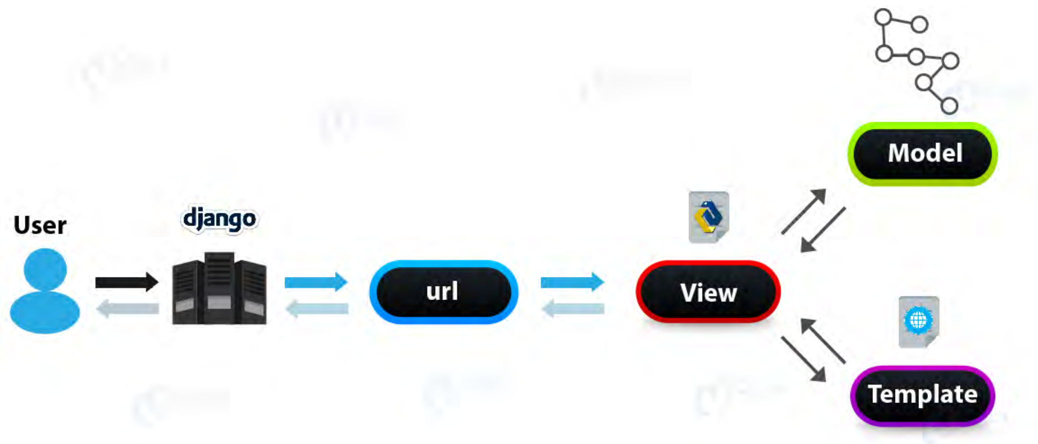


Figure 7.1: Django MVT (source: [4])

So basically, by filling a form for example, or pressing a button, the user makes a request, which is processed in a view. This request may result in alternation of the database. Finally, the view returns the results to a template where they are visible to the user.

7.1 Models

A Django model represents a table in the database. They are used to store data through views, and display them in templates.

The following model was used to save information about surgical history of a patient.

```
from django.db import models
from datetime import date
import os

class Surgery(models.Model):
    patient = models.ForeignKey(PatientProfile , on_delete=models.CASCADE)
    doctor = models.ForeignKey(UserProfile , on_delete=models.CASCADE)

    date = models.DateField(null=False , default=date.today)
    procedure_description = models.CharField(max_length=5000, blank=True)
    result_description = models.CharField(max_length=5000, blank=True)
    medication = models.CharField(max_length=500, blank=True)
    side_effects = models.CharField(max_length=1000, blank=True)
    secret_note = models.CharField(max_length=1000, blank=True)

    def __str__(self):
        return self.patient.user.username + ' – ' + self.doctor.specialty
```

- The **patient** and **doctor** variables are defined as Foreign Keys which allows to create a many-to-one relationship. Shortly, it means that the patient and the doctor are actually objects of PatientProfile (3.2.2) and UserProfile (6.2.1) respectively.
- **Date** is a DateField, used to store the date that the surgery took place, it can not be null and has a default value, which is the day that the record is inserted. Default does not mean it can not change.
- The rest are all defined as CharField. These fields include strings with a predefined max length. The blank option allows the variable to be empty.
- The function returns the username of the patient and the doctor's specialty.

7.2 Views

Every time a user makes a request, a view is triggered. The views are python functions, but they can also be classes. Django provides some generic class-based views that are very useful, because they offer an easy way to display lists of a model, details of a specific object of a model, and even update and delete objects.

7.2.1 ListView

In order to show the list of surgeries that are related to a patient we can use the following class-based View:

```
from .models import *
from django.views import generic

class surgeries_list(generic.ListView):
    context_object_name = 'surgeries_list'
    template_name = 'mr/surgeries_list.html'
    paginate_by = 15

    def get_queryset(self):
        p_id = self.request.session.get('p_id')
        return Surgery.objects.filter(patient__id=p_id).order_by('-date')

    def get_context_data(self, **kwargs):
        context = super(surgeries_list, self).get_context_data(**kwargs)
        context['received_messages'] = Message.objects.filter(
                receiver=self.request.user, was_read=False)
        return context
```

- First of all, the template that displays whatever this view returns, is ”surgeries_list.html”
- The paginate_by variable defines the maximum records shown in a page. If there are more, more pages are created.
- What the view returns:
  1. ’surgeries_list’ which is the queryset generated by **get_queryset** So, p_id refers to the patient that the doctor has chosen.

     The query **Surgery.objects.filter(patient__id=p_id).order_by(’-date’)** actually returns objects of the Surgery model that have as patient the one the doctor asked for, and are order by newest to oldest.
  2. some extra content which is actually used in the parent template.

7.2.2 DetailVew

Another type of generic view is the following:

```
from .models import *
from django.views import generic

class surgery_detail(generic.DetailView):
    model = Surgery
    template = 'mr/surgery_detail.html'

    def get_context_data(self, **kwargs):
        context = super(surgery_detail, self).get_context_data(**kwargs)
        context['received_messages'] = Message.objects.filter(
                receiver=self.request.user, was_read=False)
        return context
```

This view will be triggered when a doctor tries to see the details for a specific surgery record. In this case, a primary key will be sent from the template to the view, through the URL. This is how the view knows which record the user asked for.

The **get_context_data** function wouldn't be necessary, but here it is useful for the parent template.

7.2.3 DeleteView

```
from .models import *
from django.views import generic
from django.urls import reverse_lazy

class SurgeryDelete(generic.DeleteView):
     model = Surgery
     success_url = reverse_lazy('mr:surgeries_list')
```

This is how simple it is to delete objects in Django. The primary key of the objects is passed through the URL, so Django knows which object to remove.

7.2.4 UpdateView

```
from .models import *
from django.views import generic
from django.urls import reverse

class SurgeryUpdate(generic.UpdateView):
    model = Surgery
    fields = ['date', 'procedure_description', 'result_description',
            'medication', 'side_effects', 'secret_note']
    template_name = "mr/update_surgery.html"

    def get_success_url(self):
        return reverse('mr:surgery_detail', kwargs={'pk': self.object.id})

    def get_context_data(self, **kwargs):
        context = super(SurgeryUpdate, self).get_context_data(**kwargs)
        context['received_messages'] = Message.objects.filter(
                receiver=self.request.use, was_read=Falser)
        return context
```

This view is used to update the data of an object. Again, Django knows which entry by the primary key. The fields variable determines which fields the user will be able to update.

7.2.5 function view

This is the most common way to handle forms that store data. The following view is used to display the proper form to insert a surgery object in the data.

```
from .forms import *
from .models import *
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from django.template.context_processors import csrf
from django.template import RequestContext

@login_required
def add_surgery(request):
    template_name = 'mr/add_surgery.html'
    p_id = request.session.get('p_id')
    registered = False
    received_messages = Message.objects.filter(receiver=request.user,
            was_read=False)

    if (request.method == 'POST'):
        surgery_form = SurgeryForm(data=request.POST)

        if (surgery_form.is_valid()):
            surgery = surgery_form.save()
            surgery.save()

            request.session['current_surgery_id'] = surgery.id
            registered = True
        else:
            print (surgery_form.errors)
    else:
        default_patient = PatientProfile.objects.get(id=p_id)
        default_doctor = UserProfile.objects.get(id=request.user.userprofile.id)

        surgery_form = SurgeryForm(initial={
            "patient": default_patient,
            "doctor": default_doctor
        })

    return render(request, template_name, {
        'surgery_form': surgery_form,
        'registered': registered,
        'p_id': p_id,
        'received_messages': received_messages
        })
```

- **@login_required** decorator: if the current user is logged in the view will execute normally, otherwise the user will be asked to log in.
- If the user clicks on the submit button and the first if is true, and also the form is valid, the data are saved with **surgery.save()** command. If the form is not valid the user will see the proper errors.
- Before the form is submitted (while request.method == 'POST' is false), the doctor is set to be the current user, and patient the one the doctor has chosen. This way the object that will save later will be associated with the correct users.

7.3 Forms

Django provides a helper class which allows users to create form class from Django models. The generated Form class will have a form field for every model field specified, in the order specified in the fields attribute. For example:

```
from django import forms
from mr.models import *

class SurgeryForm(forms.ModelForm):
    class Meta:
        model = Surgery
        fields = ('patient', 'doctor', 'date', 'procedure_description',
                'result_description', 'medication', 'side_effects',
                'secret_note')
        widgets = {
            'patient': forms.HiddenInput(),
            'doctor': forms.HiddenInput(),
            'date': forms.DateInput(attrs={'type': 'date'}, format='%d-%b-%Y')
            }
```

- The model variable defines the model that this form is based on.
- Then, with the fields variable, the developer chooses which fields of this model will be included in the form.
- A widget is Django's representation of an HTML input element. Django allows to specify a widget, if not, a default widget will be used. In this case, the HiddenInput widget means that this field will not be visible to the user. These fields are properly handled in the corresponding view. The DateInput widget is used to define the desired format.

The model variable defines the model that this form is based on.

7.4 Templates

Django displays data to the users through templates. Let's see the template that is used when a doctor tries to see the list of surgeries of a patient.

```
{% extends 'mr/base.html' %}

{% block body_block %}
{% if user.is_authenticated %}
    <h3>Surgery history</h3>
    <table class="table">
        <tr>
            <th></th>
            <th>Date</th>
            <th>Doctor Specialty</th>
            <th>Doctor's Name</th>
            <th>Description</th>
        </tr>
        {% for surgery in object_list %}
        <tr>
            <td><a data-toggle="modal"
                data-target="#modalConfirmDeleteSurgery{{ surgery.id }}">
                <span class="glyphicon glyphicon-trash"></span></a>
            </td>
            {% include 'mr/surgery_confirm_delete.html' with id=surgery.id %}
            <td><a href="{% url 'mr:surgery_detail' surgery.id %}">
                {{ surgery.date }}</a>
            </td>
            <td>{{ surgery.doctor.get_specialty_display }}</td>
            <td>{{ surgery.doctor.user.get_full_name }}</td>
            <td>{{ surgery.procedure_description }}</td>
        </tr>
        {% empty %}
            There are no surgery data.
        {% endfor %}
    </table>
    {% if is_paginated %}
        <ul class="pagination">
        {% if page_obj.has_previous %}
            <li><a href="?page={{ page_obj.previous_page_number }}">
                &laquo;</a>
            </li>
        {% else %}
            <li class="disabled"><span>&laquo;</span></li>
        {% endif %}
        {% for i in paginator.page_range %}
            {% if page_obj.number == i %}
                <li class="active"><span>{{ i }}
                    <span class="sr-only">(current)</span></span>
                </li>
            {% else %}
                <li><a href="?page={{ i }}">{{ i }}</a></li>
            {% endif %}
        {% endfor %}
        {% if page_obj.has_next %}
            <li><a href="?page={{ page_obj.next_page_number }}">
                &raquo;</a>
            </li>
        {% else %}
            <li class="disabled"><span>&raquo;</span></li>
        {% endif %}
        </ul>
    {% endif %}
{% endif %}
{% endblock %}
```

- The first thing we see is the **extends** command. This tag tells the template engine that this template ”extends” to another template. When the template system evaluates this template, first it locates the parent. In this case, ”base.html” in the mr folder.
- At that point, the template engine will locate the **block** tag that exists in ”base.html” and replace that block with the contents of the child.
- **user.is_authenticated** will be true if a user has a valid account. In this case only the following will be visible.
- There is a header, so the users know what they are looking at.
- Below there is a table which shows the data that were generated by the view.
- The **for** tag is used to display every object in the list, and to access to their attributes. So for every surgery in the objects_list is printed a row with the data that are essential for the doctor. The first row of the table is a trash symbol. When clicked, a modal form appears, in order to delete the particular record. The second row shows the date of the object, which is also clickable, and when triggered, shows the details of the record.
- If the object_list is empty, a message appears, that says ”There are no surgery data”
- The code included in **if is_paginated** creates multiple pages when there are more than 15 results returned by the view.

Chapter 8

Conclusion

8.1 Summary and conclusion

This dissertation reveals many of Django's capabilities. It literally is a multi-tool that can be used for so many purposes, from simple web applications, like a plain blog website, to much more complex projects with many functions. Though it my be a little challenging for beginners, the detailed documentation is designed to cover all needs, whatever the skills of the developer may be. Besides, it is not coincidental that more and more popular web apps turn to Django. Because it is widely known, there are countless tutorials and answers to every kind of questions. Django can offer everything a developer can possibly ever ask for, simple way to add new functionalities, effortless handling of innumerus users, unlimited scalability.

As mentioned, the only possible drawback is that beginners might need to struggle a little until they feel confident enough. They need nothing more than a little dedication, effort and time, so they can make their ideas come true.

This web application is able to deal with the significant complications that traditional health records may have, like patient's safety, accessibility, storing, security, etc. It might lack some functions that other similar applications provide, like hand-writting or voice recognition, but offers something extra too. The addition of ICD-10 in an easy integrated way, can probably reduce the occurrence of errors relatively to misdiagnosing and sharing health information of patients. Not to mention, that this project gives all necessary skills to someone who is interested in doing this professionally.

8.2 Future Extensions

There is still plenty of development to be made for this web application. One of the first things would be to advance the search of doctors, providing an expanded number of filters and showing the results on a map, so a patient can easily understand the position of the doctor's office. For this to happen, the use of geodjango would be crucial.

Another addition would be to create a bill every after visit, which would also allow an online payment through the app. In the meanwhile, there would be constructed a database regarding healthcare costs.

Finally, the most significant step, will be to take advantage of these data. Either by providing statistical analysis to the users, or by using the data to extract knowledge about trends regarding to health with machine learning algorithms. A few possible inclusions would be:

- Analysis of earnings (specialists)
- Report of bills (patients)
- Statistics about health care costs in different regions
- Statistics regarding diseases globally or in specific areas
- Information about epidemic
- Vaccination Data

Appendix I

This chapter will describe the installation of django and heroku, that were necessary for the creation of the web application.

All the files that were created for this project are on github [9].

- https://github.com/stellatompazi/EHR

I.1 How to install Django on Windows

Django is a Python Web framework, hence requires Python. So the first step is to make sure that a recent version of python is installed [6].

I.1.1 Install Python

To download the latest version of python visit: https://www.python.org/downloads/

Download the executable installer and run it. Be sure to check the box next to **Add Python 3.x to PATH** and click **Install Now**. After the completion of the installation, check that the Python Version is the same you installed, by opening the command prompt and executing:

```
python --version
```

I.1.2 pip

pip is a package manage for Python. With pip, installing or uninstalling Python packages (like Django) is very easy.

To install pip visit: https://pip.pypa.io/en/latest/installing/

Follow the instructions included in **Installing with get-pip.py**

I.1.3 virtualenv and virualenvwrapper

virtualenv and **virualenvwrapper** provide an environment for each Django project. It is not mandatory but will possibly save you time in the future, probably when deploying a project. On your **cmd** simply type:

```
pip install virtualenvwrapper-win
```

After that, an environment for the project can be created by typing:

```
mkvirtualenv myproject
```

The virtual environment is created automatically. To activate the environment in another command prompt window, use:

```
workon myproject
```

I.1.4 Install Django

Installing Django is easy using pip within the virtual environment (if you choose to use one). When using a virtual environment make sure you execute the following when the environment is active. Else you just execute the following:

```
pip install django
```

I.2 Creating a project in Django

In Django, a project consists of at least one app, which essentially is a Web application that actually does something. A project is a collection of apps and configuration for a website. Briefly, a project can contain multiple apps, but also, an app can be included in multiple projects.

To create a project, run the following command in the command line, in your preferred directory.

```
$ django-admin startproject ehr
```

This will create an ehr directory in the current directory.

The next step is to change into the outer **ehr** directory, the same directory as **manage.py**. There, to create an app, run the following:

```
$ python manage.py startapp mr
```

To include an app in a project, it needs to be referenced to its configuration class in the **INSTALLED_APPS** setting.

ehr/settings.py

```
INSTALLED_APPS = [
    'mr',
    'my_mr',
    'userMessages',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bootstrap_modal_forms',
    'formtools'
]
```

To start the development server simply run:

```
$ python manage.py runserver
```

If everything works, the development server should have started. Django has a lightweight Web server written in Python. Now, if you visit http://127.0.0.1:8000/ in your browser, you will see a ”Congratulations” page.

I.2.1 Database setup

Django provides a lightweight database, SQLite. For someone new, this is the easiest choice. In order to use other databases it would be wise to consult Django’s documentation.

I.2.1.1 Creating Models

A model includes all the essential information and behaviours of the data. After a model is created, it needs to get activated. This way Django can create database schema for this app (**CREATE TABLE** statements) and provide a Python database-access API for accessing the model objects. Let’s say that we created the **PatientProfile** model from Chapter 3, in our **mr** app.

To activate the model run:

```
 $ python manage.py makemigrations mr
```

This command tells to Django that there were made changes in the models of the **mr** app and we’d like to see these changes stored as a migration. Migrations are the way Django stores any change to the models. A simple command can run the migrations and manage the database schema automatically.

```
 $ python manage.py migrate
```

These 2 steps should be done every time there are changes in the models.

I.2.2 Database API

Django gives a free API to interact with the database. To invoke it, simply run:

```
 $ python manage.py shell
```

Through this API, a user can create, update, retrieve or delete objects stored in the database, by making queries.

I.2.3 Admin User

It is necessary to create a user who can login to the admin site., by running:

```
 $ python manage.py createsuperuser
```

Enter a username and press enter. For example

```
 Username: admin
```

Then you need to give an email address.

```
    Email address: admin@mail.com
```

Finally, you have to enter a password twice.

```
    Password: **********
    Password (again): *********
```

Let's explore the admin site by starting the development server.

```
    $ python manage.py runserver
```

Then, open a web browser and go to http://127.0.0.1:8000/admin/ The admin's login should look like this:

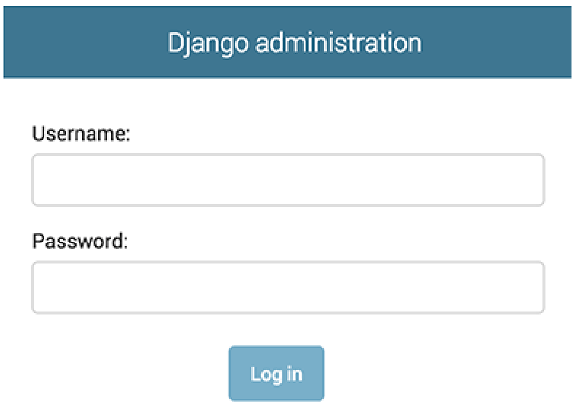


Figure I.1: Django - Admin - Log In

After logging with the superuser account, you should see the Django admin index page.

In order to make the models accessible through the admin's page you need to include them to the app's admin.py file.

mr/admin.py

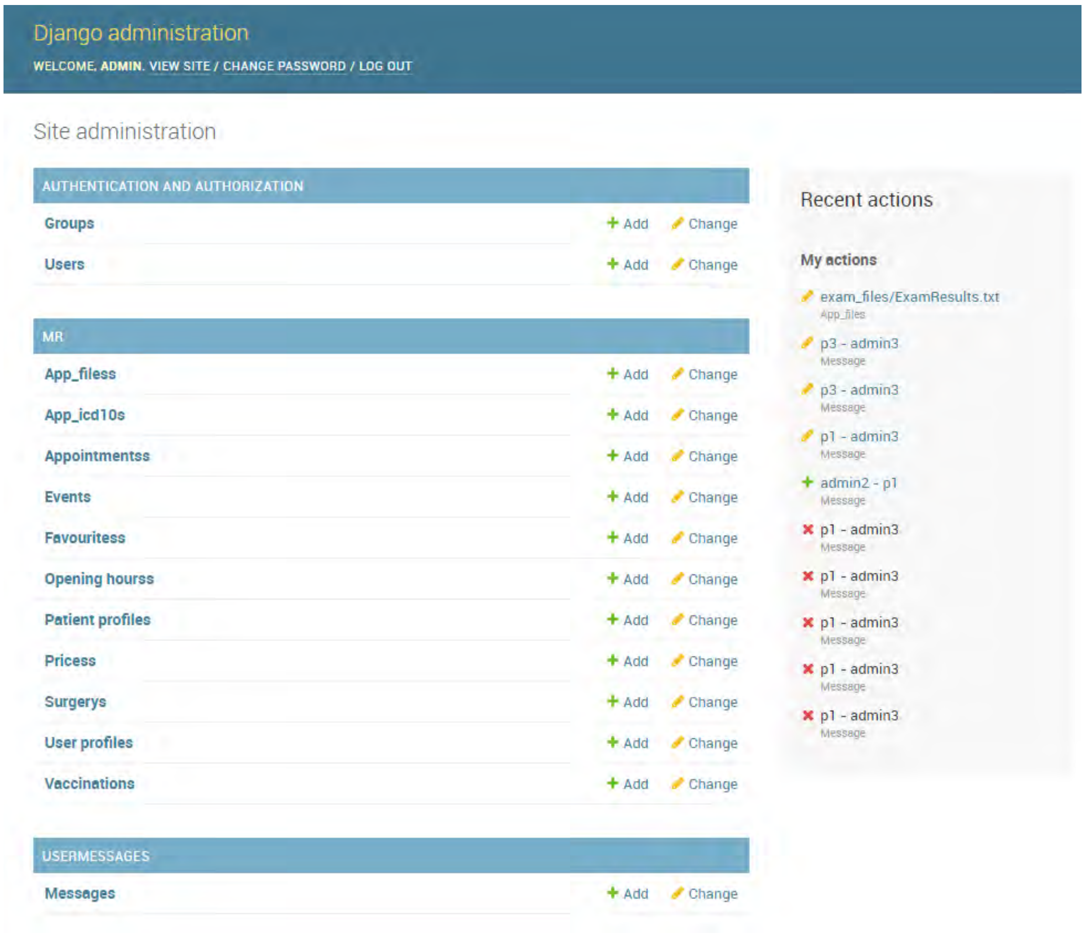
```
from django.contrib import admin
from mr.models import *

# Register your models here.
admin.site.register(UserProfile)
admin.site.register(PatientProfile)
admin.site.register(Appointments)
admin.site.register(Vaccination)
admin.site.register(Surgery)
admin.site.register(App\_files)
admin.site.register(Prices)
admin.site.register(App\_icd10)
admin.site.register(OpeningHours)
admin.site.register(Event)
admin.site.register(Favourites)
```

userMessages/admin.py

```
from django.contrib import admin
from userMessages.models import *

# Register your models here.
admin.site.register(Message)
```



Django administration

WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

MR		
App_filess	+ Add	Change
App_icd10s	+ Add	Change
Appointmentss	+ Add	Change
Events	+ Add	Change
Favouritess	+ Add	Change
Opening hourss	+ Add	Change
Patient profiles	+ Add	Change
Pricess	+ Add	Change
Surgerys	+ Add	Change
User profiles	+ Add	Change
Vaccinations	+ Add	Change

USERMESSAGES		
Messages	+ Add	Change

Recent actions

My actions

- exam_files/ExamResults.txt
  App_files
- p3 - admin3
  Message
- p3 - admin3
  Message
- p1 - admin3
  Message
- + admin2 - p1
  Message
- ✗ p1 - admin3
  Message
- ✗ p1 - admin3
  Message
- ✗ p1 - admin3
  Message
- ✗ p1 - admin3
  Message
- ✗ p1 - admin3
  Message

Figure I.2: Django - Admin - Index

I.3 Getting Started on Heroku with Python

To start deploying a project with Heroku [10], make sure to:

- Create a Heroku account for free
- Make sure Python version 3.7 is installed locally
- Install Postgres. Download Postgres here (Windows):

  https://www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows

  Do not forget to update your **PATH** environment variable to add the **bin** directory of your Postgres installation. The directory will be similar to this:

C:\Program Files\PostgreSQL\<VERSION>\bin.

In case you forget to update your **PATH**, commands like **heroku pg:psql** won't work.

- Install Git.

  Download Git here (Windows): https://gitforwindows.org/

- Install the Heroku Command Line Interface (CLI) Download here:

  https://devcenter.heroku.com/articles/getting-started-with-python#set-up

I.3.1 Prepare the app

It is a good idea to clone the application. This way you have a local version of the code to deploy to Heroku. To do so, execute the following to your **cmd**:

```
$ git clone https://github.com/heroku/electronichealthrecord.git
$ cd electronichealthrecord
```

I.3.2 Deploy the app

You need to create an app on Heroku. It prepares Heroku to receive the source code.

```
$ heroku create electronichealthrecord
```

Creating an app also creates a git remote, which is associated with your local git repository. To deploy your code, run:

```
$ git push heroku master
```

Open the app:

```
$ heroku open
```

Heroku offers information about the running app, through a logging command.

```
$ heroku logs --tail
```

I.3.3 Procfile

A procfile is a text file, existing in the root directory of the application, which declares what command should be executed to start the app. It should include something like this: Heroku offers information about the running app, through a logging command. It declares a web process type and the command needed to run it. The name **web** means that this process will be attached to the HTTP routing stack of Heroku and receive web traffic when deployed.

```
$ web: gunicorn ehr.wsgi --log-file -
```

I.3.4 App dependencies

One way Heroku recognizes a Python app, is via a **requirements.txt** included in the root directory. In the **requirements.txt** is included a list of the app's dependencies. To deploy the app, Heroku reads this file and installs the proper Python dependencies using the **pip install -r** command.

requirements.txt example:

```
django
gunicorn
django-heroku
```

To do this locally just run:

```
$ pip install -r requirements.txt
```

Bibliography

[1] 75health. https://www.75health.com/.

[2] Michel Anders. *Python 3 Web Development Beginner's Guide*. Packt Publishing, 2011.

[3] Leif Azzopardi and David Maxwell. *Tango with Django*. 2017.

[4] Django architecture. https://data-flair.training/blogs/django-architecture/.

[5] Django architecture flowchart. https://creately.com/diagram/iqjshero1/Django%20Architecture%20Flowchart.

[6] Django: the web framework for perfectionists with deadlines. https://www.djangoproject.com/.

[7] Electronic medical records (emr) software. https://www.capterra.com/electronic-medical-records-software/.

[8] Rames Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Pearson, 7 edition, 2015.

[9] Github. https://github.com/stellatompazi/EHR.

[10] Heroku. https://www.heroku.com/home.

[11] Steve Holden. *Python Web Programming*. Sams Publishing, 2002.

[12] Icd-10. https://en.wikipedia.org/wiki/ICD-10.

[13] Insync. https://www.insynchcs.com/.

[14] International statistical classification of diseases and related health problems. https://en.wikipedia.org/wiki/International_Statistical_Classification_of_Diseases_and_Related_Health_Problems.

[15] ipatientcare. https://ipatientcare.com/.

[16] Salahaldin Juba and Andrey Volkov. *Learning PostgreSQL 11: A beginner's guide to building high-performance PostgreSQL database solutions*. Packt Publishing, 3 edition, 2019.

[17] P. Luzanov, E. Rogov, and I. Levshin. *PostgreSQL for Beginners*.

[18] Postgresql. https://www.postgresql.org/.

[19] Tom Seymour, Dean Frantsvog, and Tod Graeber. Electronic health records (ehr). *American Journal of Health Sciences*, 3, October 2014.

[20] Surprising disadvantages of medical records you've never thought about. https://medrecordsinfo.com/disadvantages-of-medical-records/.

[21] Sheetal Taneja and Pratibha R. Gupta. Python as a tool for web server application development. *JIMS 8i-International Journal of Information, Communication and Computing Technology(IJICCT)*, 2, Jan. 2014.

[22] The history of health information management. https://liaison.opentext.com/blog/2017/05/02/history-heath-information-management-now/.

[23] Top 10 django apps. https://www.netguru.com/blog/top-10-django-apps-and-why-companies-are-betting-on-this-framework.

[24] What is a web application? https://blog.stackpath.com/web-application/.

[25] Swati Yanamadala, Doug Morrison, Catherine Curtin, Kathryn McDonald, and Tina Hernandez-Boussard. Electronic health records and quality of care. an observational study modeling impact on mortality, readmissions, and complications. *Medicine*, 97(3), May 2016.

Abbreviations

API	Application Programming Interface
CMS	Content Management System
DTL	Django Template Language
EHR	Electronic Health Record
etc	et cetera
ICD-10	International Classification of Diseases - 10th version
IT	Information Technology
MVT	Model View Template
NASA	National Aeronautics and Space Administration
OCR	Optical Character Recognition
OS	Operating System
REC	Regional Extension Centers
WHO	World Health Organization