**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА**

Факультет прикладної математики та інформатики

(повне найменування назва факультету)

Кафедра дискретного аналізу та інтелектуальних систем

(повна назва кафедри)

**Дипломна робота**

Автоматизація процесу взаємодії лікарів пацієнтів та клінік з використанням .net та Angular7 технологій

Виконав: студент групи ПМІ-43

спеціальності інформатика

(назва спеціальності)

Клименко В. В

(підпис) (прізвище та ініціали)

Керівник Коркуна Н. М

(підпис) (прізвище та ініціали)

Рецензент

(підпис) (прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Львів – 2019

**Зміст**

**Абстрактний опис 3**

**Вступ 4**

**Основна частина 5**

[**CHAPTER 1**](http://../Downloads/Telegram%20Desktop/Diploma_Andrew.docx#__RefHeading___Toc8545513) **5**

**П.1. Oгляд функціональності веб-застосування: 5**

**П.2. Інфраструктура серверної частини 15**

**П.3. Інфраструктура клієнтської Веб-аплікації: 19**

[**CHAPTER 2**](http://../Downloads/Telegram%20Desktop/Diploma_Andrew.docx#__RefHeading___Toc8545514) **24**

**П.1.Авторизація та автентифікація 24**

**П.2. Реєстрація та захист паролів 35**

[**CHAPTER 3**](http://../Downloads/Telegram%20Desktop/Diploma_Andrew.docx#__RefHeading___Toc8545514) **41**

**П.1. Збереження та обробка географічних даних клініки 41**

**П.3. Найкращий шлях до клініки (генетичний алгоритм) 43**

**Висновки 47**

**Список літератури**  **48**

Суть даного проекту – автоматизація процесу реєстрації в клініках та самого процесу лікування. Особливістю є те що проект не націлений на одну конкретну клініку, а має на меті розробити систему, програмними засобами, щоб об’єднати декілька, а в кращому випадку й усі відомі клініки, що дасть можливість пацієнтам в зручний спосіб знайти й зареєструватися у необхідну клініку без необхідності відвідувати кожну з них.

Вступ

Дана робота має на меті декілька завдань. Найголовнішим із них є процес автоматизації взаємодії між пацієнтами, адміністраторами клінік та лікарями. Також після кожного успішного лікувального процесу лікар здобуває певні нові знання і дуже часто ці знання ніде не зберігаються, оскільки для більшості досвідчених лікарів отримані знання не мають великої цінності, а для молодих стажерів ця інформація може бути дуже цінною. Тому важливо зберігати усі здобутті знання після лікування, зрозуміло що лише за згоди пацієнтів.

Дуже часто є ситуація коли пацієнти не знають або не впевненні до якої клініки чи до якого пацієнта їм варто звернутися у випадку необхідності. Тому дуже актуальним є завдання об’єднання даних про доступні клініки і лікарів в одній системі. Також пацієнти зможуть оцінювати якісь роботи клініки і лікарів що дозволить оцінити ту чи іншу клініку і того чи іншого лікаря. Це дозволить спростити пошук лікарів для кінік та підвищить рівень послуг що надаються пацієнтам.

Дуже важливою частиною даної роботи є дослідження в сфері захисту даних, оскільки в даному проекті я маю на меті зберігати цінну конфіденційну інформацію. Тому в даній роботі досліджено сучасні методи викрадення та захисту цифрової інформації та для організації процесу автентифікації й авторизації для веб застосувань. Також розглянуто найкращі методи для кодування/декодування та збереження паролів у базі даних.

Для того щоб ця система була популярною вона в першу чергу має бути зручною у використанні й містити найнеобхідніший функціонал для пацієнтів. В межах даної роботи я організував процес лікування й реєстрації пацієнта, розбивши його на декілька етапів, для того щоб перейти до наступного стану, користувач повинен ввести деякі дані або зачекати поки інший користувач зробить певні дії або введе певні дані. По завершенню лікування на сторінці пацієнта з’являться відповідне модальне вікно з можливість зберегти дані у спільній базі знань, куди збережуться дані аналізів, опис хвороби й висновки лікаря та файли з дослідженнями лікаря. Очевидно що у пацієнта буде можливість зберегти ці дані під фейковим ім’ям для конфіденційності.

Також для кожного пацієнта є персональна таблиця із даними про усі лікування в усіх клініках, що значно спрощує процес слідкування за станом здоров’я. Для зручності у використанні, застосування містить окрему сторінку для перегляду на інтерактивній карті усіх зареєстрованих в системі клінік та їх лікарів із можливість зробити запис до відповідного лікаря.**CHAPTER 1**

Основною метою даної роботи є автоматизація процесу лікування і реєстрації в клініках за допомогою програмних засобів а також аналіз сучасних технологій та підходів для розробки веб-застосувань. Для цього я розробив веб-аплікацію із використанням технологій .Net Web API та Angular7. Для цієї частини роботи я використав найновіші підходи і технології, для реалізації Веб-застосунку, який складається із Rest-full веб-сервера та веб-сервіса для клієнтської частини.

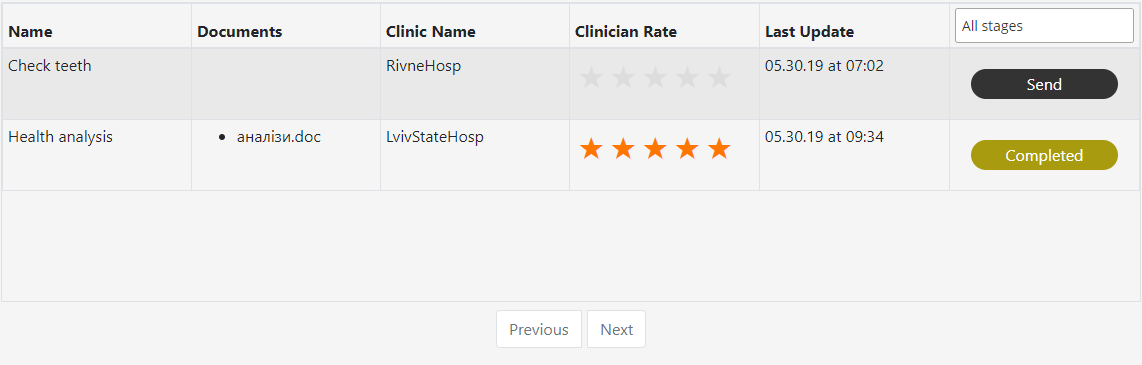
**П.1. Організація процесу лікування та огляд функціональності веб-застосування:**

В цьому розділі розглянуто функціональні можливості застосування. Користувачі аплікації поділяються на три ролі: ‘Пацієнти’, ‘Лікарі’ та ‘Адміністратори клініки’. Кожен із них може переглядати існуючі записи на прийом до відповідного лікаря. Проте для кожного із них відображаються різні дані і обмежуються деякі функції.

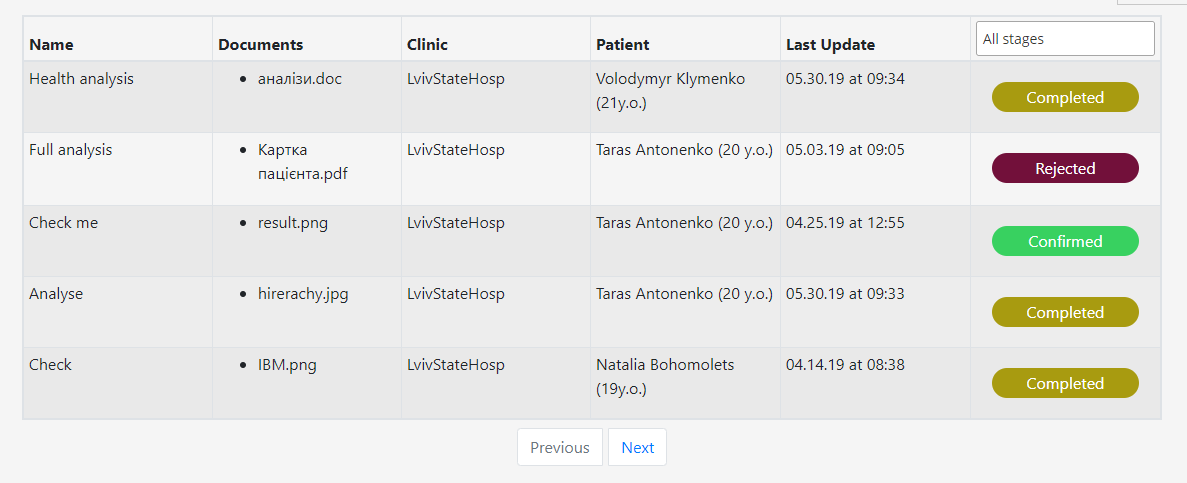
Веб-застосування включає можливість для пацієнта створювати і переглядати існуючі записи на прийом до відповідного лікаря. Відповідно лікар може також переглядати створенні для нього записи та відповідним чином реагувати на них. Адміністратор клініки може переглядати усіх лікарів клініки їх рейтинг, кількість пацієнтів та відгуки.

Дані про записи на прийом відображаються у табличному вигляді причому для пацієнтів та лікарів відображаються по різному.

Таблиця записів для пацієнта:



Таблиця записів для лікаря:



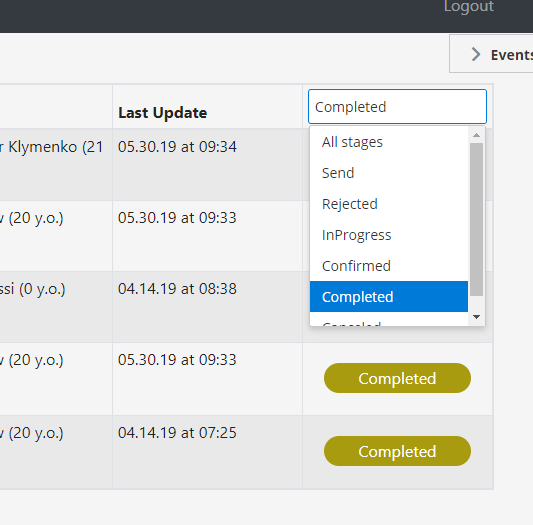


Дані відображаються не всі одразу а по 10 записів з можливістю пагінації, причому вони є посортовані по останній даті змін зроблених на них.

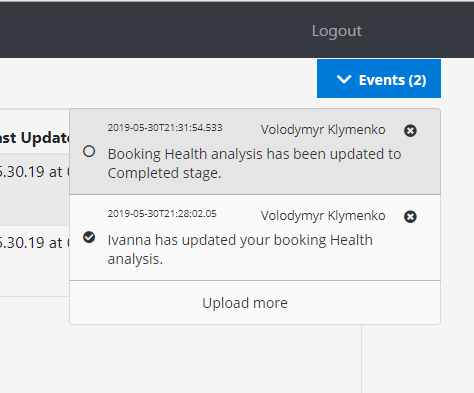
Кожен запис до лікаря (рядок таблиці) може перебувати в одному із наступних станів:

* Send – Пацієнт створив запис але лікар ще не відреагував на нього;
* Rejected – Лікар відмовився приймати запит на прийом від пацієнта;
* Confirmed – Лікар прийняв запит але ще не почав процес обстеження та лікування;
* In Progress – Лікар обстежив пацієнта, ввів необхідні дані та призначив лікування;
* Completed – Лікар завершив процес лікування пацієнта, пацієнт може оцінювати записи лише в цьому стані;
* Canceled – Лікар не зміг знайти правильного підходу у лікуванні пацієнта. Такі записи може бачити адміністратор клініки і призначати (редагувати) нового лікаря для пацієнта. Перевести запис у цей стан може як і пацієнт так і лікар.

Також користувачі застосування можуть сортувати колонки по зростанню чи спаданню та фільтрувати записи по станах, вибравши необхідний із відповідного випадаючого списку:



На кожну дію чи зміну лікаря для відповідного пацієнта відображається повідомлення про внесенні зміни у правому верхньому куті екрану:



Кожне повідомлення може перебувати у одному із двох станів:

* Прочитане
* Непрочитане

Біля кнопки для відкриття вікна із повідомленнями відображається кількість непрочитаних повідомлень. Також користувач може видаляти повідомлення.

*Як відбувається процес лікування:*

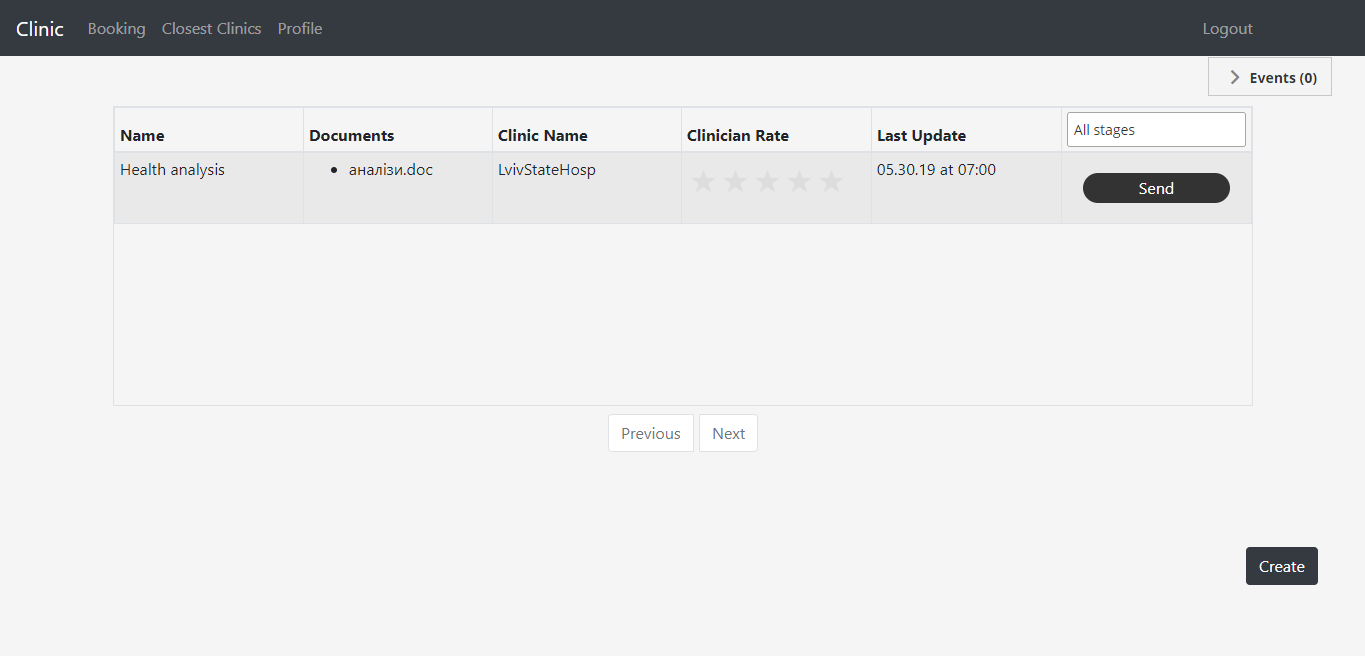
Процес лікування умовно розбитий на декілька етапів. Для цього для пацієнта на сторінці з його уже створеними записами на прийом існує спеціальна кнопка для запису на прийом. Пацієнту необхідно обрати відповідну клініку і лікаря та ввести необхідні дані. Модальне вікно виглядає наступним чином:



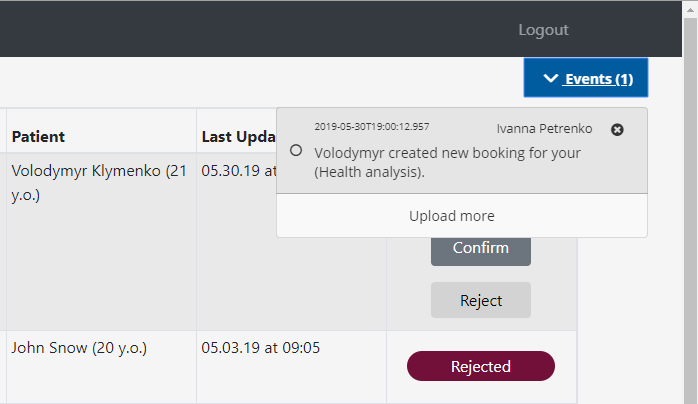
Для запису на прийом необхідно ввести

* Назву запису на прийом до лікаря (необов’язкове поле, може містити ключові слова та може бути використане для швидкого пошуку)
* Опис захворювання
* Вибрати із випадаючого списку необхідну клініку (клініки відображаються посортованими по відстані до пацієнта, та підтягуються не всі одразу із сервера а по десять із можливість відобразити наступні десять)
* Після чого обрати (також із випадаючого списку) необхідного лікаря (Відображаються лише лікарі для цієї клініки та також не всі одразу)
* Також можна завантажити файли із існуючими аналізами чи іншими необхідними документами (якщо такі у пацієнта є)

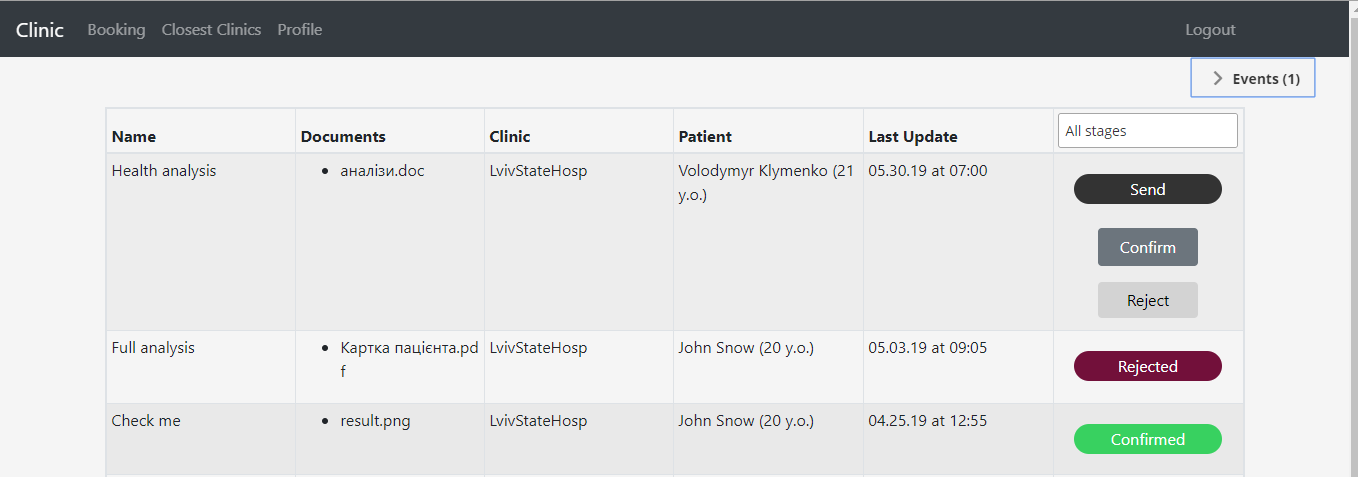
Після чого у таблиці із записами появиться новий рядок із щойно створеною реєстрацією (із позначкою ‘*Send*’):



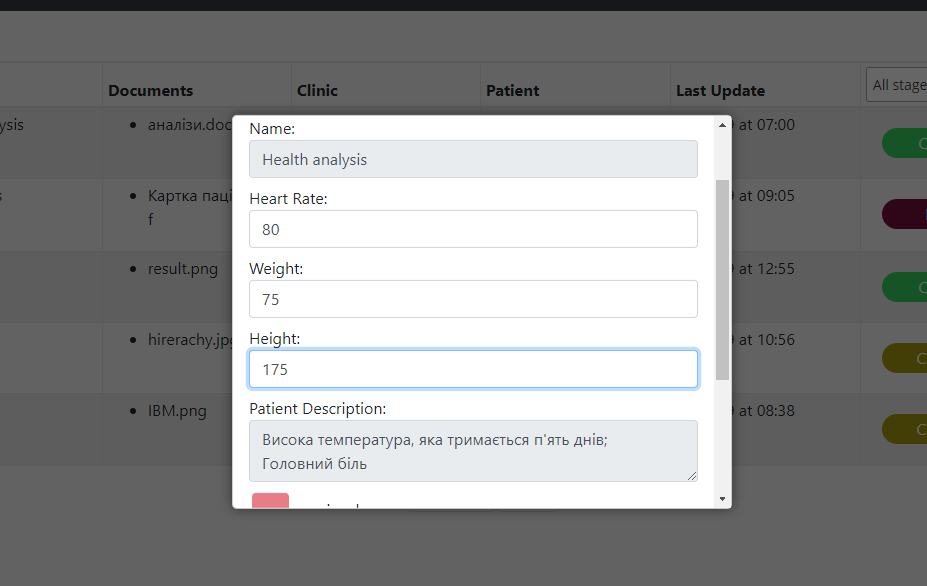
Також створений запис на прийом до лікаря буде відображатися для вказаного лікаря та він отримає відповідне повідомлення про нову реєстрацію (повідомлення буде показуватися у верхньому правому куту екрана)

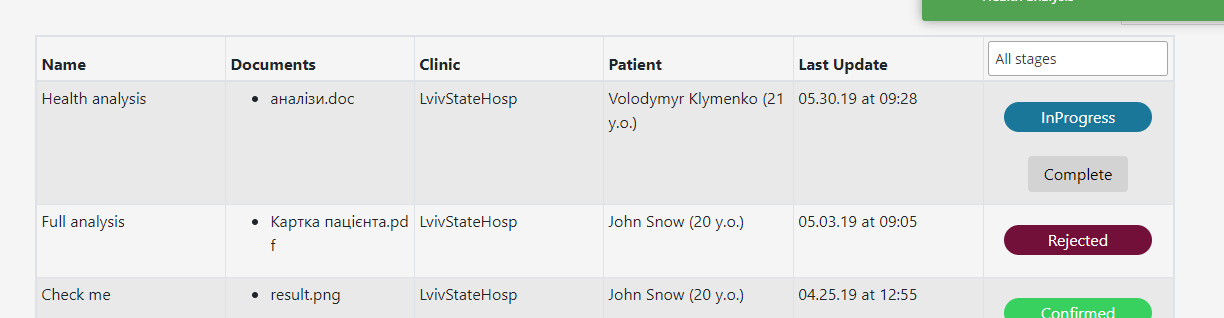


Після чого пацієнт повинен зачекати доки лікар підтвердить чи відмовиться у реєстрації. Для цього у відповідному рядку таблиці записів в останній колонці для лікаря є дві кнопки що даються можливість прийняти запит на прийом чи відмовитися від нього.

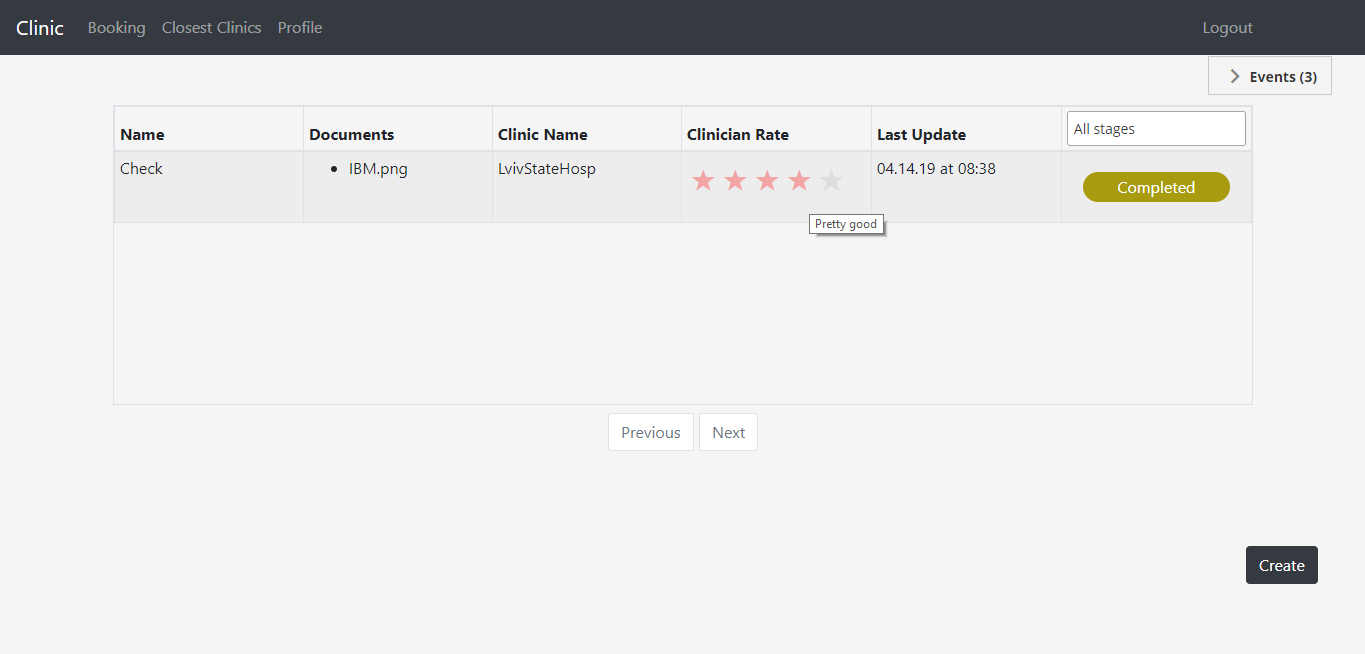


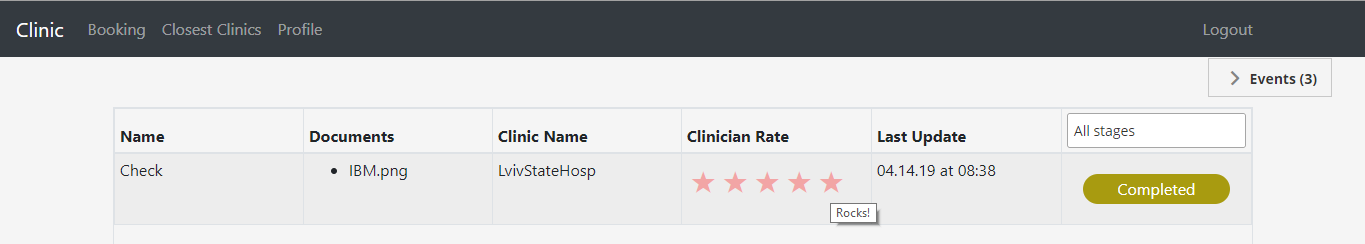
Після чого під час лікування лікар вводить результати аналізів і короткий опис обстеження та курс лікування. Що дозволить вести статистику фізичного стану пацієнта і можливість для недосвічених стажерів проаналізувати процес лікування. А запис переходить у наступний стан ‘In Progress’. Причому лікар не може змінювати значення введені пацієнтом.

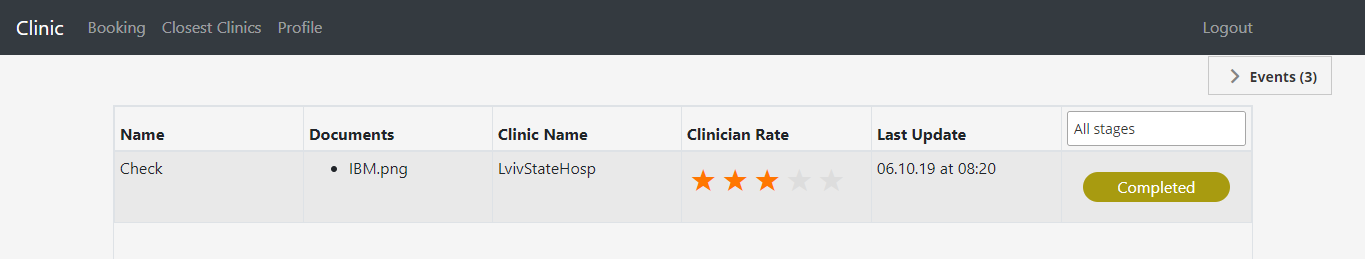




Після завершення лікування лікар нажимає кнопку ‘Complete’ і отримує повідомлення про можливість зберегти проведене лікування у спільну базу знань (лише при згоді пацієнта, пацієнт повинен підтвердити це на своїй сторінці). Також пацієнт може оцінити роботу лікаря (лише по завершені лікування) під час цього лікування. Для цього йому слід визначити якість роботи лікаря від одного до п’яти обравши відповідну зірочку із компонента для рейнтингу:

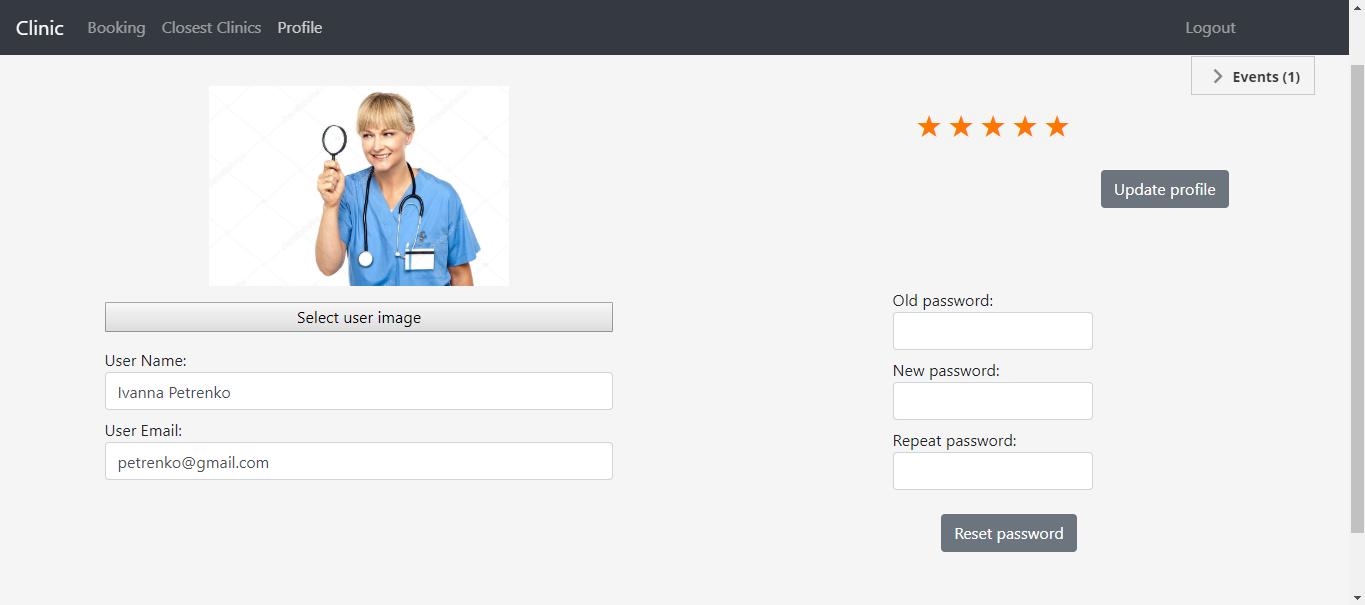






Відповідно загальний рейтинг лікаря визначається як середнє значення усіх його таких оцінок для записів пацієнтів.

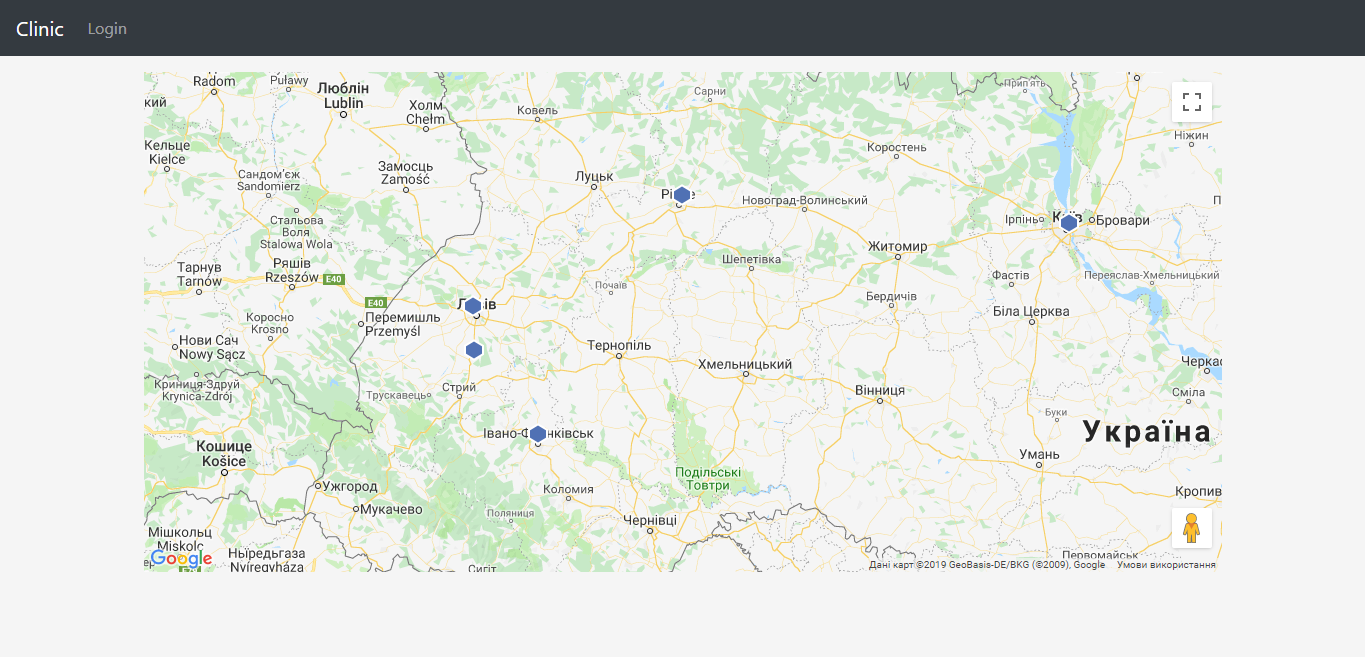
Для кожного користувача також є можливість редагувати свої персональні дані:



Сторінка з персональною інформацією для лікаря.

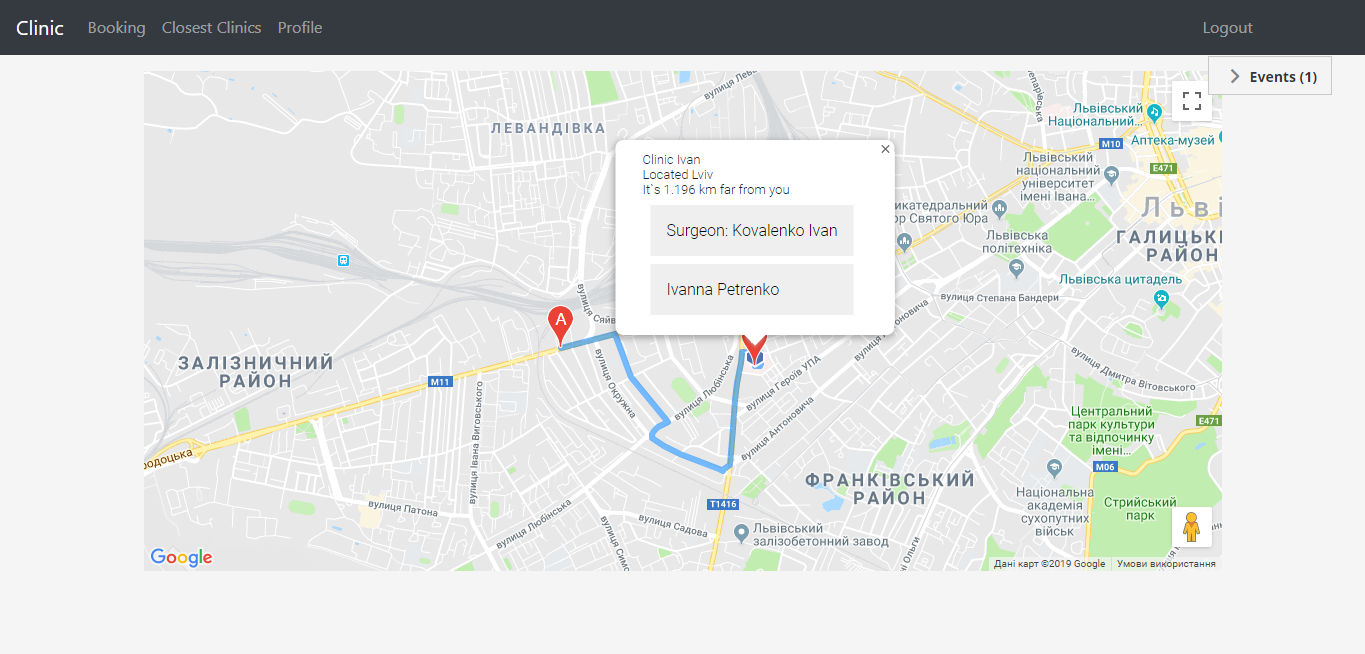
Користувач також має можливість змінити його існуючий пароль, заповнивши відповідні поля для існуючого та нового паролю.

Зрозуміло, що для користувача важливо візуально бачити найближчі клініки і лише потім переглядати її лікарів. Це застосування включає окрему частину для відображення найближчих клінік для користувача. Виглядає вона наступним чином:



Слід зазначити, що клініки відображаються не всі одразу, що зайняло б багато часу, а лише десять найближчих до нього. Значення розташування(геолокація) клінік зберігається у базі даних та вказується адміністратором клініки при реєстрації. Значення розташування користувача визначаються автоматично застосуванням. Користувач також має можливість вказати на карті для якого місця слід відображати найближчі клініки (натиснувши на відповідне місце на карті):

При виборі відповідної клініки користувач отримує інформацію про клініку, відстань до цієї клініки у кілометрах та список лікарів які в ній працюють із можливість переглядати їхній рейтинг. А також для користувача відображається найкоротший шлях до обраної клініки та є можливість зробити запис для відповідного лікаря цієї клініки:

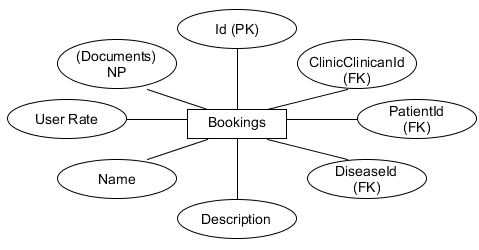


Детальніше про процес визначення відстані до клінік, найкоротшого шляху та його відображення розказується у розділі 3.

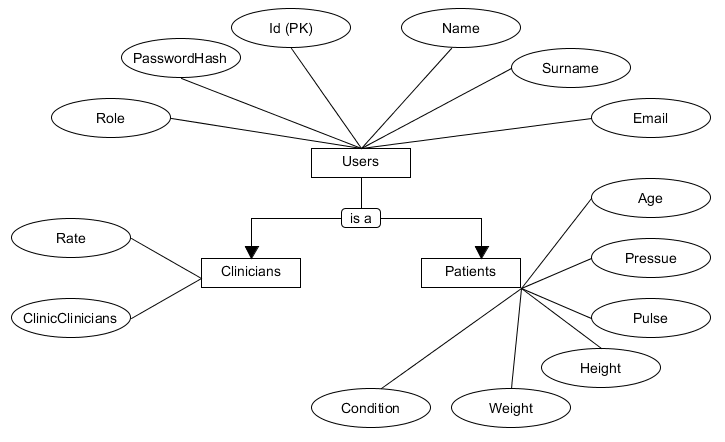
*Зберігання даних*

Розглянемо також як дані зберігаються у базі даних. Для цього використовуються наступні сутності:

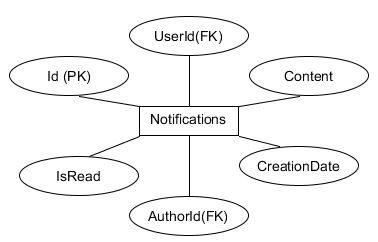
* Booking – представляє прийом у лікаря та містить наступні властивості:



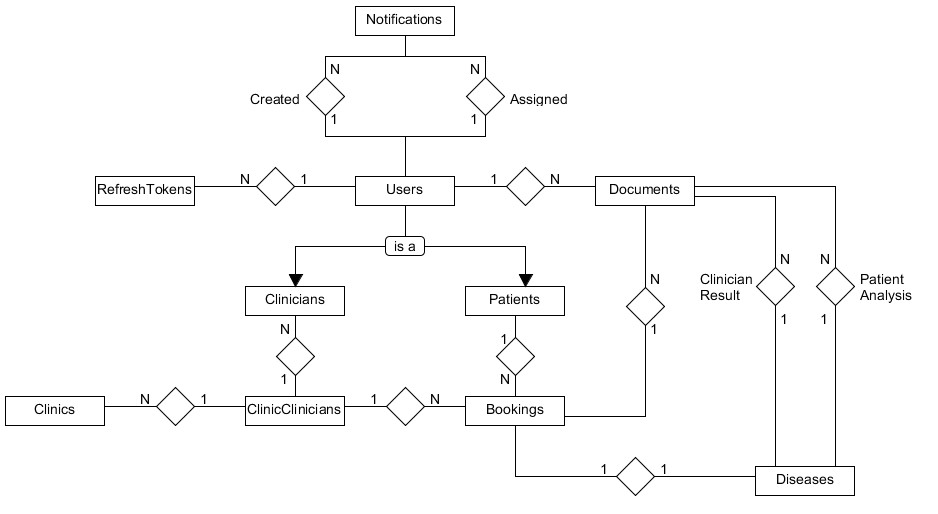
* User – абстрактна сутність із загальною інформацією для лікарів, пацієнтів та адміністраторів клінік
* Clinc – представляє дані про клініки
* Clinician – про лікарів
* Patient – про пацієнтів



* ClinicClinicians – встановлює відповідність між лікарями та клініками в яких вони працюють. Лікар може працювати в декількох клініках одночасно
* Notification – містить інформацію про певні зміни для сутності прийому у лікаря



ER - діаграма сутностей має наступний вигляд



**П.2. Інфраструктура серверної частини:**

Серверна частина проекту являє собою Rest-Full веб сервер, реалізований засобами .Net Web Api 2.0. Для зберігання даних було використано реляційну базу даних Microsoft Sql-Server. Комунікація між клієнтською та серверною частиною відбувається через протокол HTTP2.0.

*Архітектура серверної частини*

В даному веб-застосуванню я реалізував класичну трьох-рівневу архітектуру з наступними шрами:

1. Клієнтською логікою для обробки Http запитів (*назвемо його шаром представлення*)
2. Бізнес-логікою обробки вхідних даних (*шар логіки*)
3. Логіка для збереження та доступу до бази даних (*шар даних*)

Розглянемо більш детальніше, що являє собою розбиття програми на шари і підходи до реалізації архітектури застосування.

Концепція шарів – одна з загально використовуваних моделей, використовуваних розробниками програмного забезпечення для розділення складних систем на більш прості частини.

<< Layer >>

<< Subsystem >>

<< Subsystem >>

uses

<< Layer >>

<< Subsystem >>

Шар більш високого рівня користується послугами, що надає нижній шар, але той не знає про існування сусіднього верхнього рівня. Більше того, зазвичай кожен проміжний шар приховує нижній шар від верхнього.

Розділення системи на шари надає цілий ряд переваг:

* Окремий шар можна сприймати як єдине самодостатнє ціле, не піклуючись про наявність інших шарів;
* Можна обрати альтернативну реалізацію базових шарів;
* Залежність між шарами зводиться до мінімуму;
* Кожен шар є кандидатом на стандартизацію;
* Якісно створений шар може слугувати основою для декількох різних шарів більш високого рівня.

При переході від шару до шару сутності зазвичай піддаються трансформації з одного представлення в інше.

1. *Шар представлення, Філософія REST*

Шару представлення стосується усе, що пов’язане зі взаємодією користувача з системою*.* В моєму випадку шар представлення являє собою .Net WebApi 2.0 проект, який відповідає усім вимогам Rest-full філософії. Розглянемо більш детальніше ці принципи.

REST (Representational state transfer) початково був задуманий як простий і однозначний інтерфейс для керування даними. Передбачає лише деякі базові операції: вилучення даних (GET), збереження даних (POST), оновлення даних(PUT/PATCH), видалення (DELETE). Зрозуміло, що це все супроводжується такими опціями як: обробка помилок у запитах(чи запит правильно сформований), розділення доступу до даних, валідація вхідних даних.

Також це вимагає дотримання наступних архітектурних особливостей:

* Незалежність сервера від клієнта, що дозволяє замінити клієнтську частину без внесення змін до серверної частини або використання додаткового клієнта.
* Унікальність адреса для ресурсів – тобто кожна одиниця даних в базі даних має свій унікальний URL-адрес.
* Присутність у відповіді сервера усіх необхідних метаданих – окрім самих даних, що запитувалися, сервер повинен повертати також й деталі обробки запиту. В моєму сервері це повідомлення про помилки чи відбулося успішне оновлення чи видалення певних даних.

Розглянемо приклади URL – адресів для доступу і модифікації сутності ‘записів до лікаря’ розробленого застосування:

* GET: <http://serverpath/api/bookings?pageNumber=2&pageSize=10>

Повертає 10 посортованих значень ‘записів’ із бази починаючи з 21 які створені або записані на цього користувача.

Унікальний ідентифікатор користувача передається зашифрованим у Заголовках запиту (у такому вигляді: *eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VySUQiOiIxMDEwIiwiZW1haWwiOiJzb21lQGdtYWlsLmNvbSIsInJvbGUiOiJQYXRpZW50IiwibmJmIjoxNTU2NTYxMzA2LCJleHAiOjE1NTY1NjMxMDYsImlhdCI6MTU1NjU2MTMwNn0.s-UHIm8e3wOwGjL9t4GATYlKHMOt6Nz7IWHF1GuyT-Q*). Детальніше про це розказується у наступних розділах роботи.

* DELETE: <http://serverpath/api/bookings?id>=1001

Перевіряє чи унікальний ідентифікатор користувача, переданий у заголовках запиту, співпадає із значенням користувача що створив чи для якого створили цей запис. Після чого запис до лікаря із ідентифікатором 1001 - видаляється, при цьому.

* POST: <http://serverpath/api/bookings>

Створює нове значення запису, ідентифікатор користувача передається у заголовках запиту а значення властивостей створеної сутності передаються у тілі запиту у форматі ‘JSON’ наступним чином:

"Data": {

"ClinicId": 0,

"ClinicName": "string",

"PatientId": 0,

"PatientName": "string",

"PatientAge": "string",

"ClinicianId": 0,

"ClinicianName": "string",

"ClinicianRate": 0,

"BookingRate": 0,

"Id": 0,

"Stage": 1,

"Name": "string",

"Rate": 0,

"HeartRate": 0,

"Weight": 0,

"Height": 0,

"CreationDate": "2019-06-01T14:47:18.456Z",

"UpdateDate": "2019-06-01T14:47:18.456Z",

"PatientDescription": "string",

"Documents": [

{

"Id": 0,

"Name": "string",

"FilePath": "string",

"UserId": 0

}

]

}

* PUT: <http://serverpath/api/bookings>

Оновлює існуюче значення сутності запису до лікаря. Нові значення властивостей передаються у вище згаданому форматі.

* PATCH: <http://serverpath/api/bookings/rate>

Оновлює значення оцінки рейтингу роботи лікаря пацієнтом. Ідентифікатор та нове значення рейтингу передаються у тілі запиту наступним чином:

{

"Id": 0,

"Value": 0

}

1. *Шар логіки*

Містить специфічну бізнес-логіку та алгоритми для обробки даних. Також містить логіку для хешування паролів, та обробки ‘access token’ для автентифікації, що детально розглядається у Chapter 2.

1. *Шар даних*

Містить запити до бази даних, обмін повідомленнями, управління транзакціями тощо. В моєму випадку я використав патерни проектування *Unit of work* та *Repository* для організації доступу до бази даних.

Таким чином структуру створеного веб-застосування можна описати за допомогою наступної схеми:

Клієнт

http

request

Шар представлення

http

response

View Models

Шар логіки

DTO Models

База даних

Sql queries

Шар даних

**П.3. Інфраструктура клієнтської Веб-аплікації**:

Сьогодні веб технології швидко розвиваються. Платформо-незалежність та доступність веб сайтів роблять їх дуже привабливими для розробників. Розвиток веб технологій сьогодні дозволяє будувати не лише веб сайти у звичайному сенсі цього слова, коли сайт складається зі сторінок, а й дуже інтерактивні сайти – веб-сервіси які можна називати повноцінними додатками.

Односторінковий веб-додаток (SPA) – це концепція веб-сайту, що робить його дуже інтерактивним й дуже схожим на звичайний додаток, зберігаючи при цьому всі переваги веб-сайту. Для даної роботи я створив саме односторінковий веб-додаток з використанням технології Angular7.

Концепція односторінкового додатку виглядає дуже привабливо, проте вона теж має певні недоліки та переваги. Розглянемо основні переваги та недоліки.

*Переваги односторінкових додатків*:

1. Насичений функціоналом інтерфейс

2. Швидка реакція інтерфейсу, завдяки відсутній необхідності звертатися до серверу при кожній дії.

3. Значне зменшення навантаження на сервер.

4. Значне спрощення логіки та складності серверу.

5. Схожість додатків з нативними додатками операційної системи.

*Недоліки*:

1. Підвищення навантаження на клієнт завдяки великої кількості JavaScript.

2. Складність розробки.

Веб-додаток Angular 7 структурно складається з модулів, компонентів, директив та сервісів. Компоненти контролюють певну частину DOM-елементів та містять логіку їх відображення й поведінки. Директиви, на відміну від компонентів, містять лише поведінку. Сервіси створені для зручного впровадження одного екземпляра класу в компонентах та директивах додатку які від них залежать, тобто сервіси реалізують шаблон проектування «Одинак».

Розглянемо основні особливості реалізованого веб-додатку:

* *Компонентна структура*

Односторінковий додаток містить велику кількість скриптів та HTML. Це потребує гарної структуризації інакше підтримка та масштабування, можливо навіть й розробка, стануть неможливими. В традиційних сайтах структурної одиницею є сторінки. В односторінкових додатках доводиться робити структуризації іншими способами, зокрема розділенням на компоненти.

Компонент - це певний об’єкт, який має ресурси й поведінку, він може залежати він інших компонентів або містити їх. Розділення на компоненти – дуже популярний підхід який використовується у багатьох фреймворках графічного інтерфейсу. Розбиття на компоненти вважається одним із принципів бест-практісе для розробки додатків з використанням Angular7.

В моєму веб-сервісі я виділив п’ять основних компонентів. Кожен із яких в свою чергу складається із декількох залежних компонентів. А також я відділив декілька загальних компонентів (таких як для таблиці, модального вікна, …), які використовуються і можуть бути використані в інших вище згаданих.

Загальна структура має вигляд:

* Компонент аплікації (із спільною частиною для усіх інших)
* Компонент для авторизації/автентифікації
  + Авторизація
  + Реєстрація
    - Реєстрація клініки
    - Реєстрація лікаря
    - Реєстрація пацієнта
* Компоненту з існуючими записами в клініках
  + Компонент редагування запису
    - Компонент редагування документів
* Компонент із картою найближчих клінік
  + Компонент із інформацією про клініку
* Компонент із персональною інформацією
  + Для пацієнта
  + Для лікаря
  + Для адміністратора клініки
  + Для відновлення паролю
* Загальні компоненти
  + Компонент модального вікна
  + Компонент таблиці
    - Компонент пагінації
  + Компонент для оцінювання рейтингу

Слід зазначити що усі загальні компоненти є досить гнучким і розраховані на усі пристрої якими зможуть користуватися пацієнти і лікарі. В першу чергу це стосується таблички яка при маленьких розмірах екрану девайсу переформатовується у зручний для користувача вигляд:



* *Модульність*

Оскільки, як було зазначено раніше, одно-сторінкові додатки мають велику кількість ресурсів, скриптів й HTML, перше завантаження додатку може стати надто довгим. Для зменшення цього слід реалізувати динамічне підвантаження ресурсів, коли вони потрібні. Модуль дозволяє визначити які ресурси, модулі, компоненти й залежності повинні бути завантаженні на клієнт.

Створений додаток складається із наступних модулів:

* Модуль для авторизації з відповідними компонентами
* Модуль для інформації про записи на прийом у відповідних клініках і лікарів
* Модуль із найближчими клініками
* Модуль із персональною інформацією
* Загальні модулі (не відображаються а лише використовуються іншими вище згаданими)
  + Модуль загальної таблиці
  + Модуль модального вікна
  + Модуль із оцінкою якості роботи
* *Клієнтська маршрутизація*

Як відомо, одно-сторінкові додатки мають одну сторінку, тобто один URL. Всі інші зміни та стани сторінки відображуються станом JavaScript змінних та сховищ, таких як Cookie та LocalStorage. Перезавантаживши сторінку або відкривши її в іншому браузері чи на іншому пристрої, ми отримаємо нову сторінку в початковому стані. Щоб зберігати стани при перезавантаженні можна використовувати сховища згадані вище.

В традиційних сайтах для цього використовується різні сторінки кожна з яких має свій URL. Оскільки суть концепції односторінкового додатку полягає не в обмеженні сторінок, а в позбавленні перезавантаження сайту при змінах, ми можемо мати сторінки але щоб уникнути перезавантаження при зміні сторінки, треба використовувати відокремлену частину URL, наприклад символом «#». Зміни URL записуються в історію браузера й є основою для браузерної навігації. Таким чином використовуючи URL для стану сторінки ми можемо користуватися кнопкою «Назад» у браузері.

В Angular2+ існує спеціальний компонент «ComponentRouter» який дозволяє зробити маршрутизацію, що базується на компонентах.

Структура клієнтської маршрутизації в розробленій аплікації має наступний вигляд:

* [http://base-address/**booking**](http://base-address/booking) - відображається таблиця із зареєстрованими лікуваннями в усіх клініках
* [http://base-address/**clinic**](http://base-address/clinic) - відображається компонент із картою з усіма зареєстрованими клініками
* [http://base-address/**login**](http://base-address/login) - відображається компонент для автентифікації
* [http://base-address/**profile/clinic**](http://base-address/profile/clinic) - відображається компонент з персональною інформацією для адміністратора клініки
* [http://base-address/**profile/clinician**](http://base-address/profile/clinician) - для лікаря
* [http://base-address/**profile/patient**](http://base-address/profile/patient) - для пацієнта
* [http://base-address/**registration**](http://base-address/registration) - відображається компонент для авторизації

для пацієнтів, лікарів і адміністраторів клінік усі компоненти відображаються по різному.

* *Інверсія управління*

З ростом кількості бібліотек та фреймворків, що використовуються при створенні односторінкових додатків, все більше набуває сенсу застосування такого принципу побудови програми як «інверсія управління», який відокремлює створення залежностей клієнта від власної логіки клієнта, що дозволяє компонентам бути слабко зв'язаними.

Згідно принципу інверсії управління, якщо в нас є клієнт що використовує певний сервіс, то він має робити це не безпосередньо, а використовуючи посередника.

Клієнт

Сервіс

Агент

(Посередник)

Принцип інверсії управління

В моєму додатку прикладом цього принципу є реалізація інтеграції із бібліотекою (Сервіс) для відображення *карти* (із модулем AgmCoreModule від компанії Google). Який використовується у компоненті Client.component.ts(Клієнт) через загальний модуль Map.module.ts (Агент).

**CHAPTER 2**

В цьому розділі детально розглянуто процес надання доступу до збереженої інформації, авторизація та автентифікація користувачів та розділення можливостей для пацієнтів/лікарів/адміністраторів.

Спершу розглядаються теоретичні відомості про авторизацію та автентифікацію. Потім дослідження про те як зловмисники можуть обійти існуючі системи захисту. А також описано підхід використаний в даній роботі для забезпечення надійного збереження персональних даних користувачів.

**П1. Авторизація та автентифікація**

Спершу розглянемо теоретичні відомості про ідентифікацію користувачів та надання їм доступу до даних.

Без процесів автентифікації та авторизації не обходиться жодне сучасне

веб-застосування. Тому важливо розуміти як відбувалися ці процеси раніше і як вони відбуваються сьогодні. Спершу розглянемо з ключовими термінами.

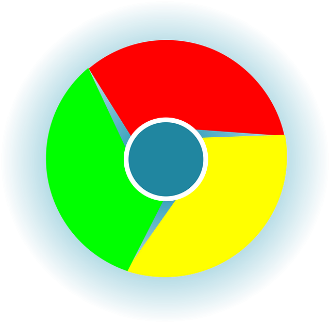
Ідентифікація – процес визначення користувача що хоче доступитися до тих чи інших даних. Автентифікація – процес підтвердження, що цей користувач саме той за кого він себе видає. Авторизація – процес прийняття рішення про те що саме цей автентифікований користувача можу робити із вказаними даними. Тобто це три різних, послідовних і взаємно незамінних поняття. Ідентифікацію часто розглядають як частину автентифікації. Найважливіше точно розрізняти авторизацію із автентифікацією.

*Способи автентифікації:*

* *Basic access authentication*

При використанні Http-протоколу, найпростішим способом автентифікації є ‘Basic access authentication’. В принципі цей протокол застарів і уже не часто використовується в інтернеті, але все ще зберігся у внутрішньо-кооперативних системах, лише через те, що деякі із них створенні досить давно. Тому варто розглянути, як він працює.

GET /home



401 Unauthorized

www-Authenticate: Basic realm=”localhost”

GET /home

Authorization: Basic YwaSAg2U6asfFSA

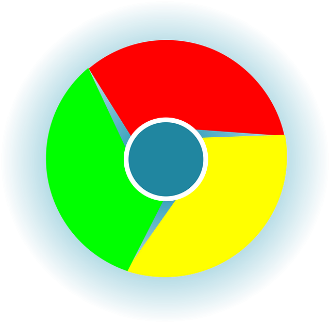
200 OK

При зверненні до захищеного ресурсу сервер поверне користувачу, що немає доступу, помилку 401 Unauthorized. При цьому відповідь також містить інформацію про тип автентифікації (в нашому випадку – Basic), який він можу отримати. Щоб отримати потрібні дані, користувач вводить логін та пароль, вони упаковуються в Basic64 і відправляються на сервер для перевірки. Тут існують різні небезпеки. Наприклад атака посередника, під час якої при використанні незахищеного з’єднання вказані дані можуть бути перехопленні в момент передачі від клієнта до сервера чи навпаки. Тому даний підхід є небезпечним та застарілм.

## *Forms Authentication*

GET: /login Http/1.1

Http/1.1: 200 OK





POST: /login Http1.1

user=’user’& password=’password’

Http/1.1: 302 Found

Set-Cookie: token=eFasg12fsqsaf

GET: /index Http/1.1

Cookie: token= eFasg12fsqsaf

Http/1.1: 200 OK

Пізніше з’явився процес Forms authentication, при якому автентифікація відбувається на більш високому рівні моделі абстракції. Http-сервер при цьому підході не повідомляє про помилку доступу, а перенаправляє неавтентифікованого користувача на іншу сторінку входу. Зазвичай на цій сторінці відображаються поля для вводу логіна та пароля, після заповнення яких формується Post-запит з даними і через захищений канал направляється на сервер. Серверна частина в свою чергу повертає токен або ідентифікатор сесії, який зберігається в Cookies і в подальшому використовується для доступу до захищених ресурсів.

* *Token authentication*

Identity provider

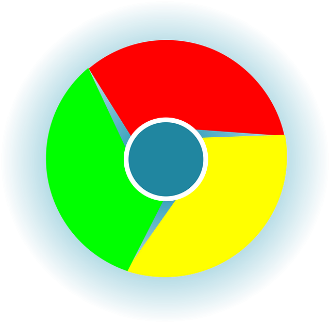
**

Get: /token

Authorization: basic: eGsa5fagasf

Http/1.1 200 OK

Token: s85Sassg21HJsafisakKfee

**

Trust

**

Get: /resource

Authorization: Bearer: s85Sassg21HJsafisakKfee

Http/1.1 200 OK

Server provider

Наступне спосіб автентифікації є Token Based Authentication. При його використанні сервіс делегує функцію перевірки інформацію про користувача іншому сервісу. Тобто перекладає відповідальність на видачу та перевірку необхідних для доступу токені на окремий сервіс (Token provider).

Наступна схема показує як і в якому порядку застосування обмінюються інформацією при використанні автентифікації по токенах.

Service provider

Identity provider

6. Auto submit

форми

5. Відсилає

токен в html

формі

4. Login

3. Запит

на дані

користувача

1. Запит

Ресурсів

7. Вертає

Ресурси

2. Перенаправляє

Перевірку токена

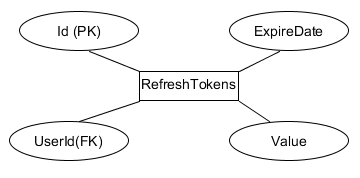
Client

## *OAuth2 & Open ID Connect*

[OAuth 2.0](http://tools.ietf.org/html/draft-ietf-oauth-v2) — новий протокол, що дозволяє надати одному сервісу права доступу до ресурсів користувача на іншому сервісі. Протокол позбавляє від необхідності довіряти застосуванню логін та пароль, а також дозволяє надавати обмежений набір прав, а не всі одночасно.

В межах даного проекту було використано саме цей підхід. Для цього спершу розглянемо ключові частини системи автентифікації:

1. Security Token Service (STS) – найважливіша частина усієї конструкції централізованого сервісу автентифікації. Являє собою окремий веб-сервіс який може приймати запити для генерації токенів доступу, токенів оновлення та перевірки токенів.
2. Client – в даному випадку це веб-аплікація на Angular 7. Намагається доступитися до даних веб-сервіса. Для цього спершу відсилає запит на Security Token Service для отримання відповідного токена доступу та токена оновлення.
3. Access Token — інформація, що конкретному користувачу надано доступ. Client запитує отримати Access Token і потім використовує його для доступу до ресурсів (Web API). Access Token також зберігає інформацію про користувача (унікальний ідентифікатор, ім’я та роль). Крім того для кожного токена вказуєтсья проміжок часу протягом якого його можна використовувати. Після цього проміжку з моменту створення токена користувач не зможе доступитися до ресурсів, використовуючи цей токен. І буде змушений або повторно автнетифікуватися або оновити існуючий access токен за допомогою токена іншого типу (refresh token) мові про яйкий піде дальше.
4. Refresh Token — токен, за допомогою можна отримати новий access токен. В залежності від режиму роботи Refresh Token може бути або ж багаторазовим або одноразовим. У випадку одноразового токена, при запиті на отримання нового access токена буде також сформований новий refresh токен а попередній стане недійсним. Очевидно що одноразові токени більш надійніші (в розробленому застосуванні використано саме цей підхід). Значення Refresh-токенів, в розробленому проекті зберігаються в базі даних у таблиці ‘RefreshTokens’, яка має зв’язок один-до-багатьох із таблицею користувачів (‘Users’). ER-діаграма цієї таблички має наступний вигляд:



# *Процес автентифікації*

При запиті користувача до клієнта той спершу перенаправляє його на Security Token Service, який запитує у користувача пароль та логін. У випадку успішного процесу перевірки вказаних параметрів автентифікації він повертає назад Access Token(з яким користувач може звертатися до серверу із необхідними даними) та Refresh Token.



Security Token

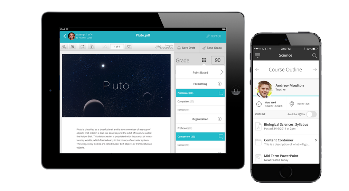
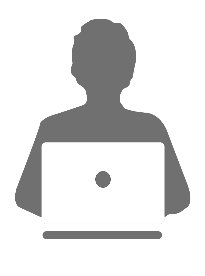
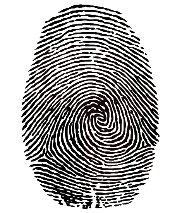
Service

Authentication /

Token request

Access token

Refresh token



Uses

Uses

Identity

Resources

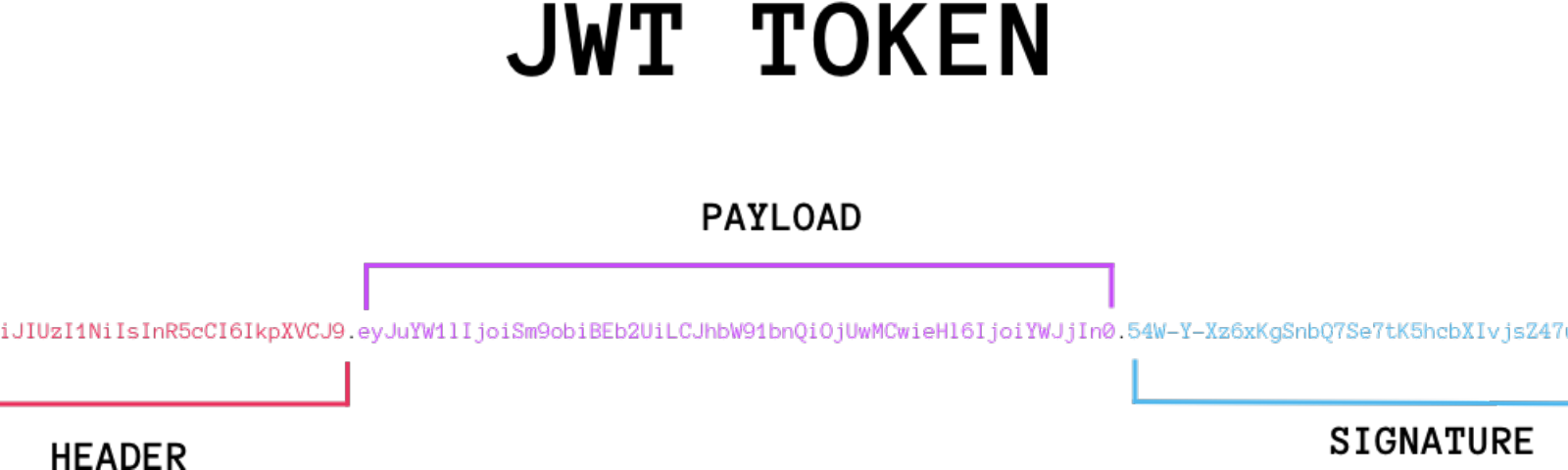
Scopes

Clients

Users

# *Структура токена*

Було використано JWT-token, який складається із трьох частин:



JSON Web Token (JWT) – це стандарт, що визначає як передавати інформацію про ідентифікацію користувача, у форматі JSON – об’єкта таким чином щоб його неможливо було розшифрувати чи змінити під час передачі.

Розглянемо його частини:

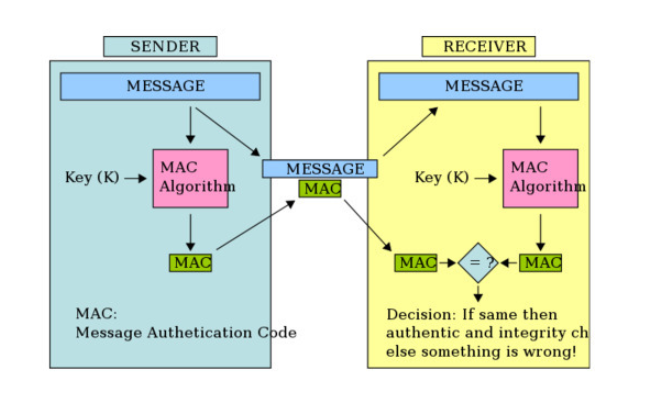
1. Header: JSON – об’єкт що визначає тип токена (JWT) та алгоритм за яким його було зашифровано.
2. Payload: JSON – об’єкт що містить інформацію про користувача (ідентифікатор, ім’я, роль)
3. Signature – ключова частина що забезпечує безпеку передачі вище згаданої інформації. Стрічка що утворюється із секретного ключа(стрічки), header частини та payload частини.

Ці частини при генерації нового JWT access token кодуються за допомогою [*base64Url*](https://brockallen.com/2014/10/17/base64url-encoding/)і розділяються крапкою. Таким чином це значення може бути лише розкодоване і закодоване в STS і відповідно доступ до нього має лише веб-сервер який відсилає отриманий із клієнта токен на STS та отримає закодовані в ньому значення, якщо це правильний токен і в якого ще не пройшов період протягом якого він є дійсним.

*Генерація та перевірка access-токена*

Для генерації та перевірки access токена використовується так званий MAC(message authentication code) механізм. Суть якого наступна:

Береться повідомлення і пропускається через спеціальну функцію що називається MAC-алгоритмом (це може бути довільна хеш-функція чи яка небудь інша). На вхід MAC-алгоритм отримує не тільки повідомлення але й ключ, відмінний від ключа шифрування.



Коли ми хочемо передати повідомлення, ми приєднаємо до нього результат виконання алгоритму. Отримувач повідомлення для перевірки чи повідомлення не було спотворене, виконує таку ж операцію (пропускає повідомлення черес MAC алгоритм) і порівнює отриманий результат із результатом переданим разом із повідомленням. Якщо вони співпадають – то повідомлення не було змінене. А якщо два отриманих результати (переданий в повідомленні і отриманий одержувачем при виконанні MAC алгоритму) не співпадають – то виникла проблема при передачі повідомлення і його не можна вважати достовірним. Схема цього процесу зображена вище.

Виникає питання що можна використати в якості MAC алгоритму.

Взагалі MAC – механізм це лише специфікація для безпечного передання повідомлень. І в залежності від типу MAC – алгоритму існує декілька реалізацій цього підходу:

1. HMAC (використаний в даному проекті): використовує в якості MAC – алгоритму хеш-функцію. Взагалі по своєму принципу ідея побудови хеш-функції похожа на алгоритм блочного шифрування. Іншими словами ми беремо який небудь блок даних і послідовно, декілька разів застосовуємо до нього набір примітивних операцій. Для прикладу для досить відомої хеш-функції MD5 це: множення по модулю 2 і циклічний зсув. Також важливою перевагою цього підходу є те, що при невеликих змінах на вхідному повідомленні хеш повинен забезпечувати великі зміни на виході. Тобто якщо у вхідному тексті замінити хоча б один біт, то в ідеалі хеш-результат повинен змінитися рівно на половину.
2. CBC-MAC: код автентифікації повідомлення ланцюгуванням шифроблоків. техніка побудови [кода автентифікації повідомлення](https://uk.wikipedia.org/wiki/MAC-%D0%BF%D1%96%D0%B4%D0%BF%D0%B8%D1%81) з [блочного шифру](https://uk.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D1%87%D0%BD%D0%B8%D0%B9_%D1%88%D0%B8%D1%84%D1%80). Повідомлення шифроване деяким алгоритмом блочного шифрування в [режимі CBC](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B6%D0%B8%D0%BC%D0%B8_%D0%B1%D0%BB%D0%BE%D1%87%D0%BD%D0%BE%D0%B3%D0%BE_%D1%88%D0%B8%D1%84%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F#CBC) для створення ланцюга блоків таких, що кожен блок залежить від правильного шифрування попередніх блоків. Ця взаємозалежність гарантує. що зміна в будь-якому біті відкритого тексту спричинить зміну в останньому зашифрованому блоці, яку не можна передбачити або запобігти без знання ключа блочного шифру.
3. IBC-MAC: MAC-функція має вигляд ***Hi = ((Mi mod p) + v) mod 2n***, де

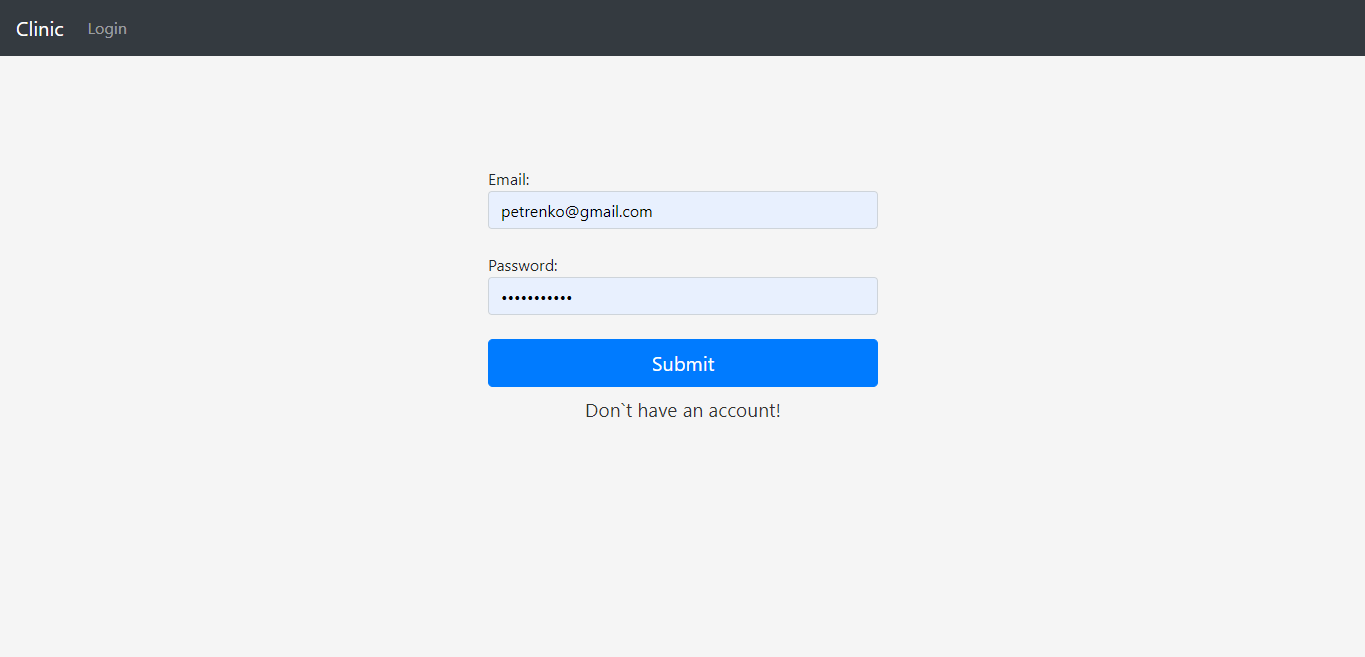
Секретний ключ складається із пари чисел p і v  
p — n-бітне просте число.   
v — випадкове число менше 2n.   
Mi — отримуємо за допомогою процедури доповнення.

*Тепер розглянемо як працює авторизація в розробленому застосуванні.*

В цьому проекті було використано OAuth 2.0 підхід на основі JWT токенів. Проте для зручності публікації проекту на зовнішньому сервері – веб-сервер та Security Token Service(STS) були створені в межах одного .Net Web API 2.0 проекту. Також передбачена можливість для оновлення access-токенів за допомогою токенів іншого типу – токенів оновлення (refresh – токени).

Для створення JWT access токенів було використано підхід HMAC(hash-based message authentication code) та у якості MAC-алгоритму – вбудовану в .Net Standard Library хеш функцію SHA-256, яка є надійною та ефективною для таких задач. Значення секретного ключа для алгоритму зберігається у файлі із конфігураціями аплікації та має приблизно такий вигляд: *WXwo4sqSDXxabIO8OOKb4qskm5X0Pk7v,* значення протягом якого створені access-token та refresh-token будуть валідними також зберігається у конфігураціях та має наступні значення 30 хвилин та 7 днів відповідно.

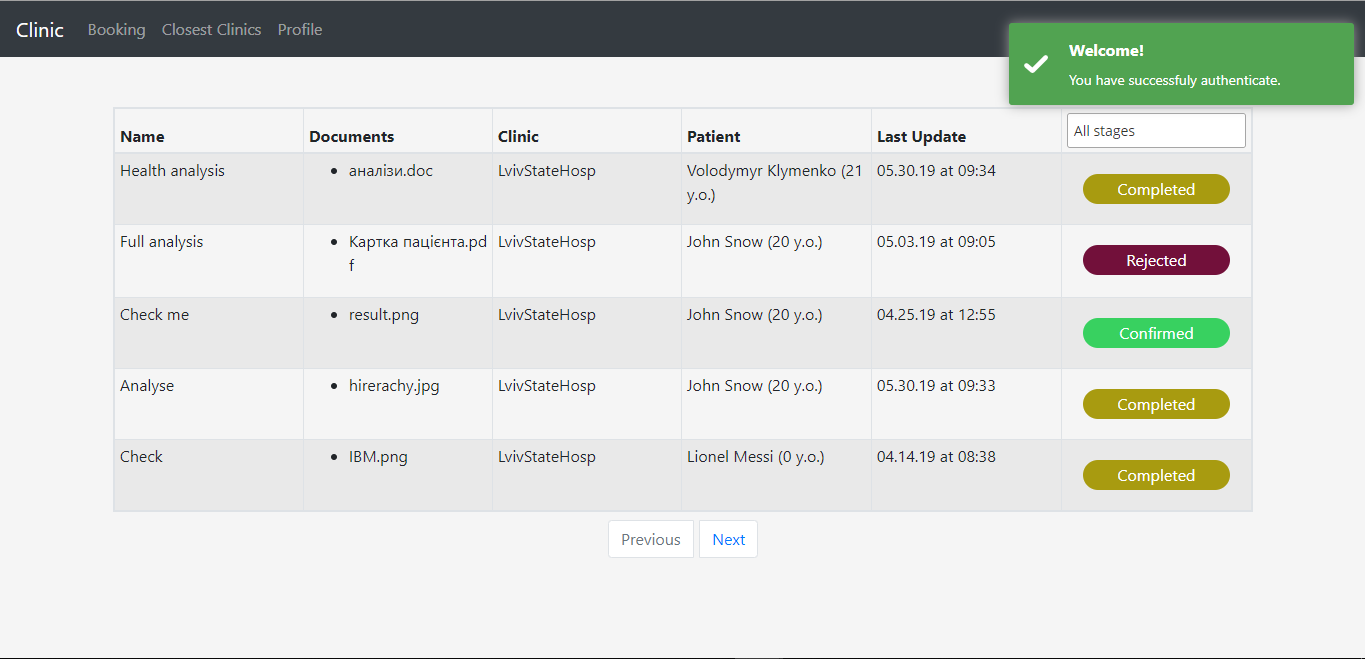
Процес авторизації/автентифікації буде відбуватися наступним чином:

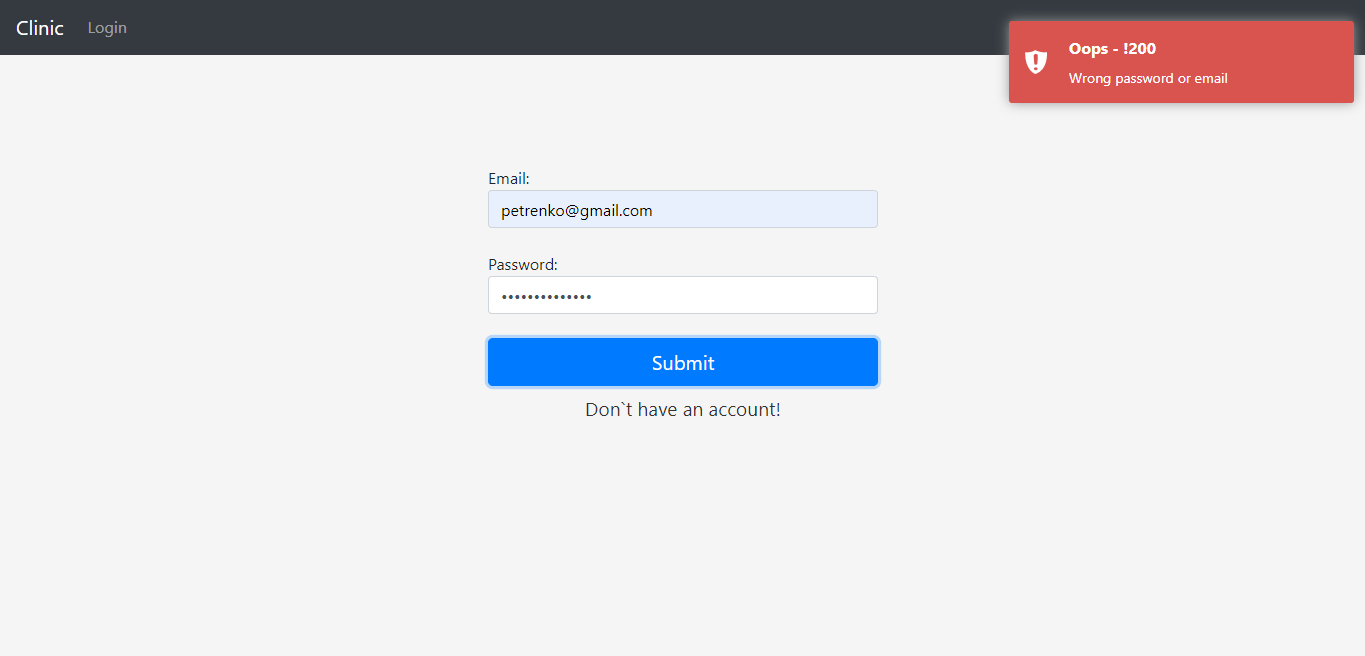
1. При вході в аплікацію клієнт відправляє запит на отримання даних про створені ним записи до лікаря на веб-сервер, передаючи при цьому значення access-token (збережене у локальній пам’яті клієнтського комп’ютера або ж пуста стрічко, це значення відсутнє) у заголовках запиту. Після чого за допомогою вище згаданого MAC-механізму перевіряємо чи переданий токен є достовірним. Якщо запит не містить значення access-token або хеш-функція вертає інше значення хешу повідомлення ніж передане значення або ж час протягом якого можна було скористатися цим токеном пройшов сервер повертає Http відповідь із кодом 301 – Un Authorized. Шлях до сторінки на якій виконувався запит зберігається в URL записі у вигляді [http://baseUrl?**returnUrl=%2Fbooking**](http://baseUrl?returnUrl=%2Fbooking)
2. Якщо клієнтська (Angular 7) веб аплікація отримує Http відповідь із кодом 301 – Un Authorized і значення access-токена чи refresh-токена відсутнє в локальній пам’яті компютера – то аплікація відображає для користувача сторінку для автентифікації:
3. Після введення відповідних значень автентифікації користувач відправляє відповідний запит на Security Token Service, який витягує із бази даних потрібного користувача та порівнює передані паролі, якщо вони співпадають то генерується нові значення JWT access-токена та refresh-токена. Та відповідно значення створеного refresh-токена зберігається у базі даних з відповідним значенням ідентифікатора користувача, якому належить даний токен та датою доки це можу бути використане. Та згенеровані значення відсилаються на клієнтську аплікацію у вигляді http відповіді із кодом 200. JWT access-токен має наступну структуру:

*Header*: { "alg": "HS256", "typ": "JWT"}

*Payload*: { "userId": "121", "userName": "NAME", "userRole": "Patient" }

*Signature*: Відповідно зашифроване за допомогою хеш функції SHA-256 значення секретного ключа разом із попередньою інформацією.

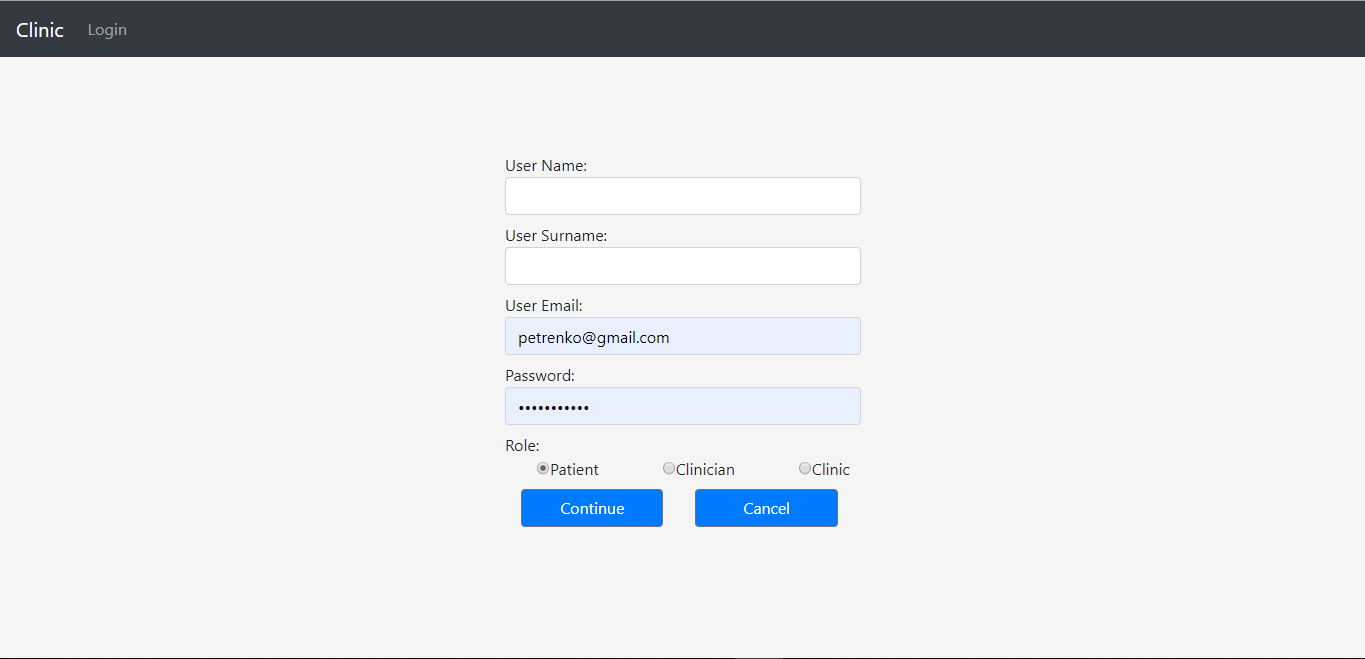
1. При отриманні http відповіді із кодом 200 клієнтська аплікація зберігає отримані дані в пам’яті та автоматично відсилає запит який повернув відповідь із кодом 301 (у нашому випадку це запит на отримання даних про створені записи для користувача) та на сервері відбувається аналогічна перевірка як і у першому кроці.
2. При успішній перевірці access-токена сервер розкодовує значення UserId із токена та витягує із бази даних усі створені ним записи до лікаря. Після чого повертає їх у форматі JSON в тілі Http відповіді із кодом 200.
3. Клієнтська аплікація при отриманні такої відповіді відображає користувачеві сторінку із таблицею повернутого масиву значень записів на прийом із відповідним повідомлення на екрані користувача:
4. У разі подальшого використання аплікації клієнтська аплікація додаватиме збережене значення access-токена до усіх запитів на сервер та у разі якщо користувач відправить такий після часу протягом якого можна було використовувати цей токен – він/вона отримає Http – відповідь із кодом 301. Після чого клієнтська аплікація обробить цю відповідь та сформує запит на Security Token Service для оновлення токена, який міститиме значення збережених access та refresh токенів.
5. При отриманні такого запиту сервер підтягне із бази даних значення користувача який надіслав запит та відповідне значення refresh-токена із відповідної таблиці. Та перевірить чи значення ідентифікатора користувача із access-токена та значення refresh-токена із бази даних співпадають із переданими. Після чого згенеруються, збережуться та відправляться нові значення. Які клієнтська аплікація зможе використовувати для доступу до необхідних ресурсів.
6. У разі якщо якийсь із переданих значень не співпадають із необхідним значенням - користувач буде перенаправлений на сторінку для автентифікації.
7. Якщо ж користувач введе невірні значення властивостей автентифікації – він/вона отримає відповідне повідомлення та залишиться на сторінці автентифікації:



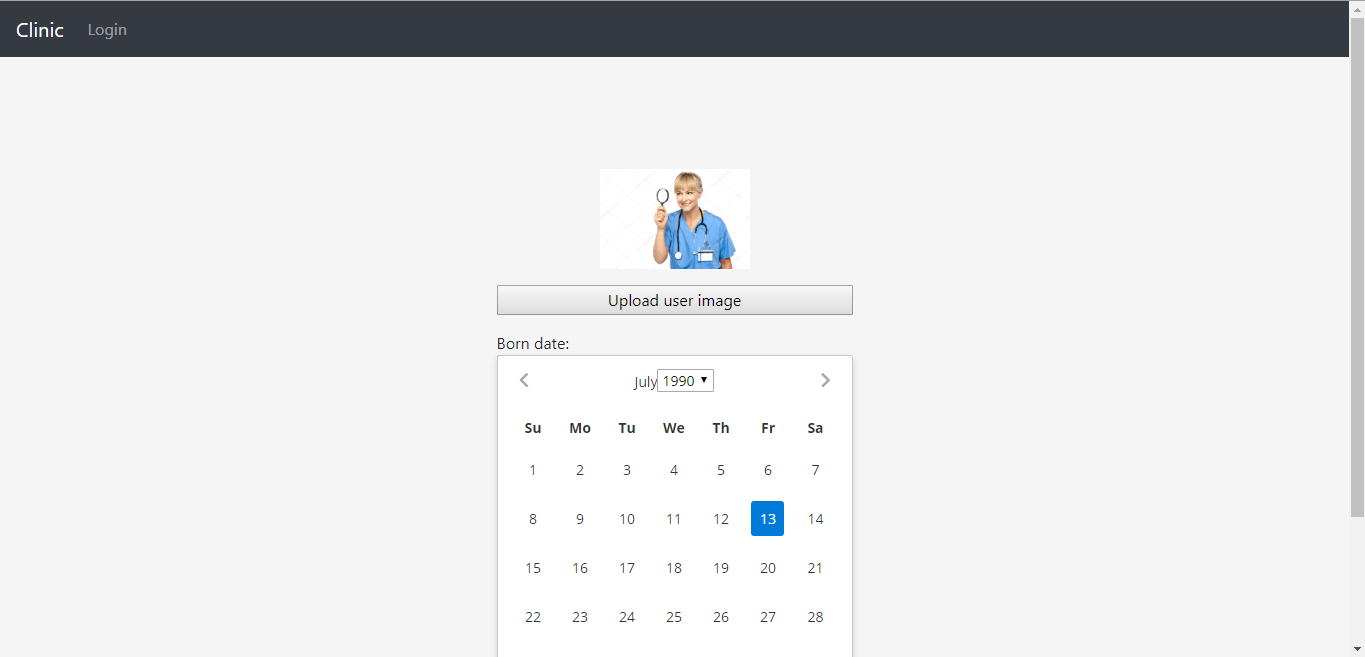
**П2. Реєстрація та захист інформації**

Якщо у користувача немає ще створеного акаунта він має можливість зареєструватися натиснувши відповідну кнопку на сторінці автентифікації.

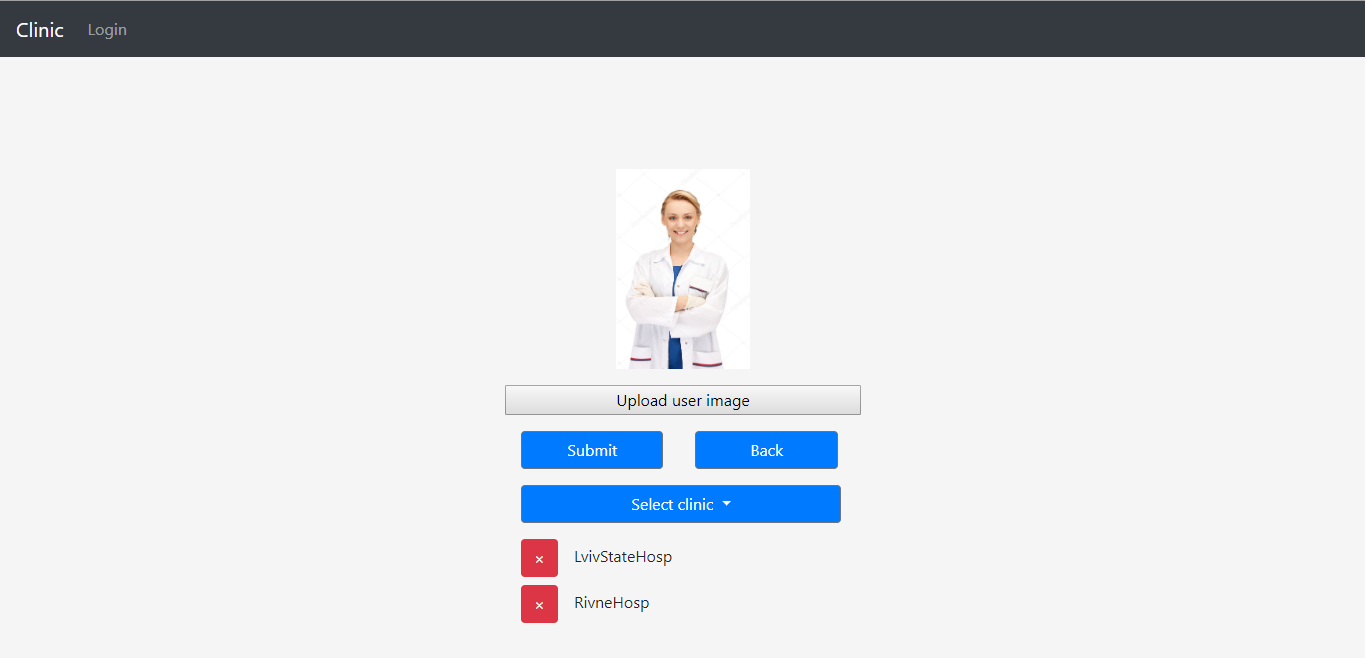
Реєстрація складається із двох етапів. Перший для усіх типів користувачів однаковий – потрібно ввести загальну інформацію про користувача (ім’я, електронна пошта, пароль … ), а також користувач повинен вказати свою роль (пацієнт, лікар чи адміністратор клініки). Відповідна форма має наступний вигляд:



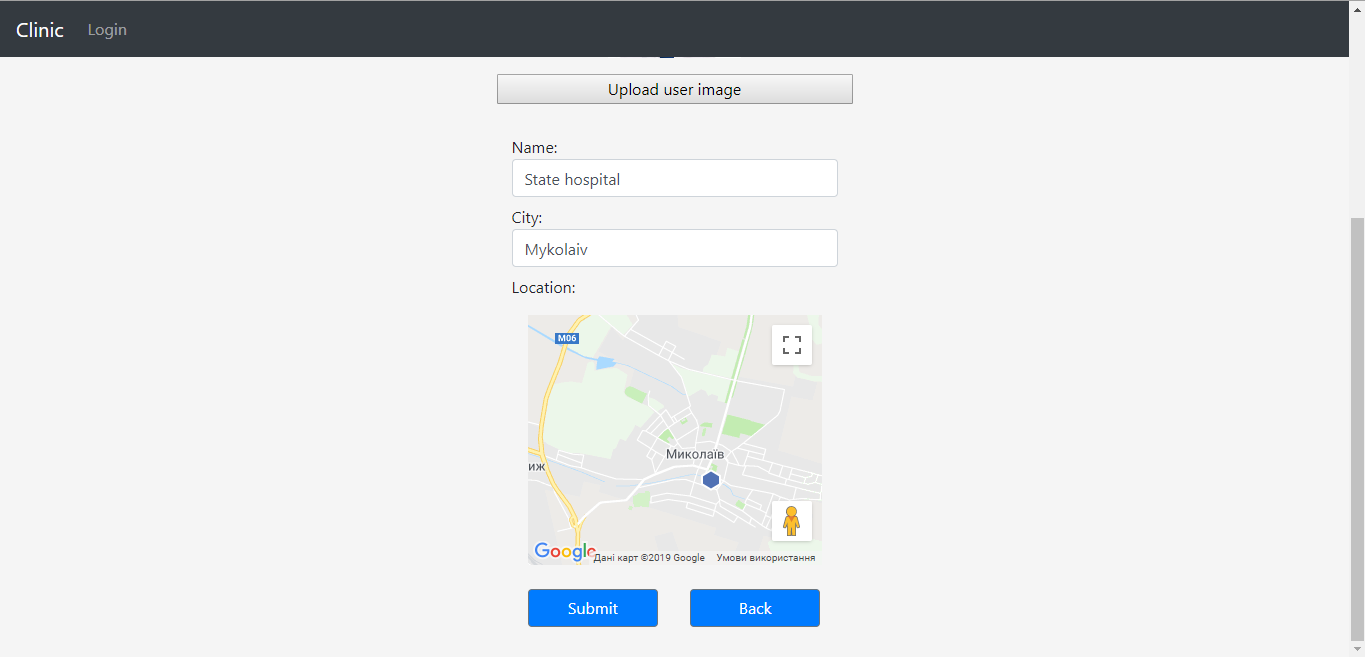
Наступний етап для кожного типу користувача специфічний і полягає у введені специфічних для кожного користувача властивостей. Для пацієнта це його зображення та дата народження:



Для лікаря це список лікарень в яких до нього можна записатися:



Для адміністратора клініки це назва клініки та місце її розташування на карті(геолокація):



*Захист інформації*

Також слід звернути увагу що даний проект має на меті зберігати цінні дані та файли із аналізами, тому слід звернути особливу увагу на захист цих даних а також паролів користувачів.

При реєстрації користувача ми отримуємо введений ним пароль. Виникає логічне питання: яким чином його зберігати? Перше що можна зробити це зберігати даній у відповідній таблиці бази даних у такому вигляді в якому ми їх отримали. Зрозуміло що це не надто хороший варіант. В процесі подальшої розробки проекту хакери зможуть знайти спосіб доступитися до них, оскільки веб-сервер напряму доступається до бази даних. Або ж можуть виникнути проблеми із безпекою на сервері де буде розгортатися розроблений проект. А в питаннях безпеки краще завжди очікувати гіршого. Тоді розглянемо випадок, що зловмисник знайшов спосіб отримати дані про паролі користувачів. Що він може із ними зробити? В основному його дії можуть бути наступними:

* Виконання нелегальних дій від імені користувача профілю.
* Використати отриманий пароль на інших аплікаціях, додатках. Користувачі далеко не завжди придумують різні паролі для різних серверів.
* Дане веб-застосування передбачає можливість оновлення пароля. Для цього слід ввести попередній використовуваний пароль. В такому разі справжній користувач профілю втратить доступ до своїх ресурсів та даних.
* Доступ до персональних даних пацієнтів. Дана аплікація зберігає конфіденційну інформацію про стан здоров’я користувачів.

Як бачимо зберігати паролі у відкритому вигляді далеко не найкращий варіант для нашого застосування.

Так що ж тоді робити? Зрозуміло що найкращим варіантом було б зберігання паролів у зашифрованому вигляді. Тоді навіть, якщо зловмисник зможе доступитися до таблички із паролями він/вона не зможе ним скористатися для автентифікації або принаймні витратить на це надто багато часу.

Тут можливі два шляхи для розвитку: шифрувати паролі чи хешувати? Тому спершу порівняємо їх за різними характеристиками:

1. *Ефективність*: Шифрування займає більше часу і його потрібно буде застосовувати кожного разу для перевірки паролю. Одним із вимог до хеш-функцій є швидкість виконання.
2. *Довжина вхідних даних*: Результат шифрування має непостійну довжину, результат хешування є завжди видає результат із однаковою довжиною. А зберігати дані однакової довжини зручніше в базі даних. Крім того у випадку шифрування, довжина зашифрованого слова буде давати додаткову інформацію про дожину пароля. З іншого боку однакова довжина приводить до виникнення колізій але про це пізніше.
3. *Керування ключами*: Для шифрування потрібним є також секретний ключ, який потрібно також десь зберігати. А це вимагає додаткових засобів для зберігання ключів.
4. *Можливість колізій*: При шифруванні отримані значення від різних вхідних паролів завжди будуть різними. Постійна ж довжина хешу зумовлює обмеженість можливих захешованих значення паролів. Для прикладу користувач може приділити достатню кількість часу, щоб придумати дійсно складний пароль із літерами на різних мовах із цифрами та специфічними символами. Зловмисник вводить в значення пароль не менш складний пароль – ‘password’ і отримує таке ж захешешоване значення у базі даних, що не дуже добре.

Як бачимо для варіанту із хешуванням більше переваг, тому зупинимося на цьому варіанті. Але чи буде цього достатньо для захисту паролів, напевно що ні.

### *Атаки на захешовані паролі*

Розглянемо тепер які можливі дії можуть бути здійснені із захещованими паролями для того щоб отримати дійсні значення і як ускладнити цей процес.

Можливі дії:

1. Якщо із складним паролем адміністраторів клінік можуть бути то із інколи дуже простими (стандартними) паролями користувачів можна перебрати найчастіші паролі та для різних способів хешування порівняти чи існують відповідні значення у вкрадених даних.
2. Існуючі таблиці: Також слід звернути увагу, що в деяких випадках навіть не потрібно хешувати і перебирати. В мережі інтернет існує велика кількість готових таблиця для часто використовуваних методі хешування, таких як MD5, SHA1, SHA256.
3. Повний перебір: якщо ж таких таблиць не існує тоді єдиним варіантом залишається перебирати підряд усі можливі паролі поки обчислені хеші не співпадають із значеннями із таблиці паролів.

В найзагальнішому випадку для отримання реальних паролів потрібно хешувати різні значення і перебирати із даними із таблиці. В такій ситуації успіх залежить від часу визначення хешу. Для цього порівняємо популярні методи хешування за середнім часом виконання. Розглянуто наступні методи MD5, SHA-1, SHA-256, SHA-512. Для визначення середнього значення кожним із цих методів було захешовано однаковий набір паролів, різної довжини та зафіксовано час виконання. Отримані результати є наступними:

* MD5 – 611 мс
* SHA-1 – 297 мс
* SHA-256 – 776 мс
* SHA-512 – 1152 мс

Зрозуміло, що найкращим варіантом для збереження паролів є використання хеш-функції SHA-512. Проте чи буде цього достатньо для безпечного зберігання. Існує велика кількість технологій, що дозволяються розпаралелити так складні операції використовуючи GPU, що значно підвищують шанси на успіх.

Я розгляну інший підхід, який полягає у хешуванні паролів декілька разів. Тобто визначаємо хеш від хешу декілька разів рекурсивно.

Що це дає:

1. В першу чергу збільшується час хешування паролів, що зменшує шанси зловмисників при повному переборі.
2. При такому методі зберігання паролів, просто використати існуючі в мережі інтернет таблиці не вийде.

Проте слід зважати також що для звичайних користувачів паролі зберігатимуться і відповідно і перевірятимуться таким же чином, що може знизити час для автентифікацї.

Проте досі залишається варіант хешувати найпоширеніші паролі різними методами і порівнювати із даними із таблиці. Для того щоб ускладнити і цей варіант було використано наступний підхід. Зрозуміло, що найочевидніший варіант – добавляти довільне згенероване значення до введеного користувачем паролю і отримане внаслідок конкатенації значення хешувати. Це дає наступні преваги:

1. Уникнути хешів для простих, стандартних паролів користувачів
2. Робить хеш кожного паролю унікальним, тобто для двох однакових паролів будуть різні захешовані значення.

Тоді виникає логічне запитання як же ж зберігати ці унікальні додаткові значення. Можливі два варіанти:

1. В окремій колонці таблички. Напевно цього не достатньо оскільки зловмисник отримає ці значення разом із паролями.
2. У файлі із конфігураціями застосування. Про те кожне таке значення повинне бути унікальним, зберігати ж велику кількість конфігурацій – не дуже хороша ідея.

Найкращий же ж варіант для кожного паролю зберігати два таких значення. Одне спільне для всіх у фалі із конфігураціями, інше унікальне для кожного паролю в окремій колонці таблички або ж у тій же ж самій колонці. Це забезпечує унікальність з однієї сторони а з іншої сторони не все значення використане для хешування зберігається у базі даних, що є досить надійним способом.

В результаті якщо зловмиснику вдасться отримати значення таблиці із паролями він стикнеться із наступними перешкодами:

1. Йому невідома спільна додаткова частина використана для конкатенації із паролем перед хешуванням. Тому йому/їй прийдеться перебирати велику кількість паролів щоб дізнатися цю спільну частину.
2. Йому відомі значення хешів паролів і додаткових унікально-згенерованих частин проте він не зможе використати готових таблиць з мережі інтернет оскільки вони містять лише значення для досить часто використовуваних паролів.
3. Час для хешування кожного паролю є досить значним, оскільки хешування відбувається рекурсивно.

**CHAPTER 3**

Важливою частиною даної роботи є також зробити це веб застосування зручним та корисним для користувачів. Для мене як і для більшості людей було б дуже зручно мати можливість переглядати найближчі від мого місця розташування клініки та створювати записи в них. Розроблений проект включає таку можливість: переглядати клініки на інтерактивній карті. На якій кожна клініка відмічається відповідною міткою та користувач має можливість переглянути інформацію про клініку та лікарів які в ній працюють натиснувши на відповідну мітку та створити для відповідного лікаря запис. Іншим способом створення запису є вибір потрібної клініки та лікаря з випадаючого списку клінік та лікарів а сторінці із табличкою записів. Для зручності користувачів клінки із цього списку відображаються у порядку зростання відстані до нього. Інтерактивна карта також включає можливість переглянути візуально найкоротший шлях до обраної клініки, який визначається за допомогою «Генетичного алгоритму».

**П.1** **Збереження та обробка географічних даних клініки**

Розташування клінік та пацієнтів зберігається та обробляється у форматі географічних координат. Система географічних координат (географічні координати) застосовується для визначення положення точок земної поверхні відносно [екватора](https://uk.wikipedia.org/wiki/%D0%95%D0%BA%D0%B2%D0%B0%D1%82%D0%BE%D1%80) й початкового (нульового) [меридіану](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D0%B8%D0%B4%D1%96%D0%B0%D0%BD). Координатами є кутові величини: [географічна широта](https://uk.wikipedia.org/wiki/%D0%A8%D0%B8%D1%80%D0%BE%D1%82%D0%B0) й [географічна довгота](https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D0%B2%D0%B3%D0%BE%D1%82%D0%B0).

[Довгота](https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D0%B2%D0%B3%D0%BE%D1%82%D0%B0) - кут між площиною [меридіану](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D0%B8%D0%B4%D1%96%D0%B0%D0%BD) в точці спостереження та нульовим ([Гринвіч](https://uk.wikipedia.org/wiki/%D0%93%D1%80%D0%B8%D0%BD%D0%B2%D1%96%D1%87)) меридіаном.

[Широта](https://uk.wikipedia.org/wiki/%D0%A8%D0%B8%D1%80%D0%BE%D1%82%D0%B0) - кут між [прямовисною лінією](https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B1%D0%B5%D1%81%D0%BD%D0%B0_%D1%81%D1%84%D0%B5%D1%80%D0%B0#.D0.9F.D1.80.D1.8F.D0.BC.D0.BE.D0.B2.D0.B8.D1.81.D0.BD.D0.B0_.D0.BB.D1.96.D0.BD.D1.96.D1.8F) та площиною [екватора](https://uk.wikipedia.org/wiki/%D0%95%D0%BA%D0%B2%D0%B0%D1%82%D0%BE%D1%80).

Визначають положення точки на поверхні [Землі](https://uk.wikipedia.org/wiki/%D0%97%D0%B5%D0%BC%D0%BB%D1%8F). Вимірюються в градусах (°), [довгота](https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D0%B2%D0%B3%D0%BE%D1%82%D0%B0) — від 0° до 180° на [захід](https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D1%85%D1%96%D0%B4) та на [схід](https://uk.wikipedia.org/wiki/%D0%A1%D1%85%D1%96%D0%B4) від Гринвіча, [широта](https://uk.wikipedia.org/wiki/%D0%A8%D0%B8%D1%80%D0%BE%D1%82%D0%B0) — від 0° до 90° на [північ](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B2%D0%BD%D1%96%D1%87), від 0° до −90° на [південь](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B2%D0%B4%D0%B5%D0%BD%D1%8C) від [екватора](https://uk.wikipedia.org/wiki/%D0%95%D0%BA%D0%B2%D0%B0%D1%82%D0%BE%D1%80).

При реєестрації клініки користувач повинен вказати геолокацію (точку на карті). Для цього було використано сервіс ‘Google Maps Server’ що надає можливість відображати інтерактивну карту та взаємодіяти із нею.

Із розташуванням клінік усе зрозуміло. При реєстрації клініки користувач повинен вказати геолокацію (точку на карті). Для цього було використано сервіс ‘Google Maps Server’ що надає можливість відображати інтерактивну карту та взаємодіяти із нею. Після чого відповідні значення довготи та широти зберігаються у відповідній табличці бази даних (‘clinics’). Для визначення актуального розташування користувача застосування використовується функція браузера – «Географічне розташування». Виникає питання яким же ж чином браузер визначає ці координати. Коли ви активуєте цю функцію, в браузері відображається повідомлення про використання вашого розташування в картах Google. Якщо ви приймаєте запит, браузер намагатиметься визначити ваше розташування. Виконується аналіз точок безпровідного доступу Wi-fi навколо вашого місця розташування і вашої IP-адреси, після чого ці дані передаються на сервер для їх переведення у місцезнаходження, яке і буде відображатися на карті та використовуватися для обчислення відстані до клінік.

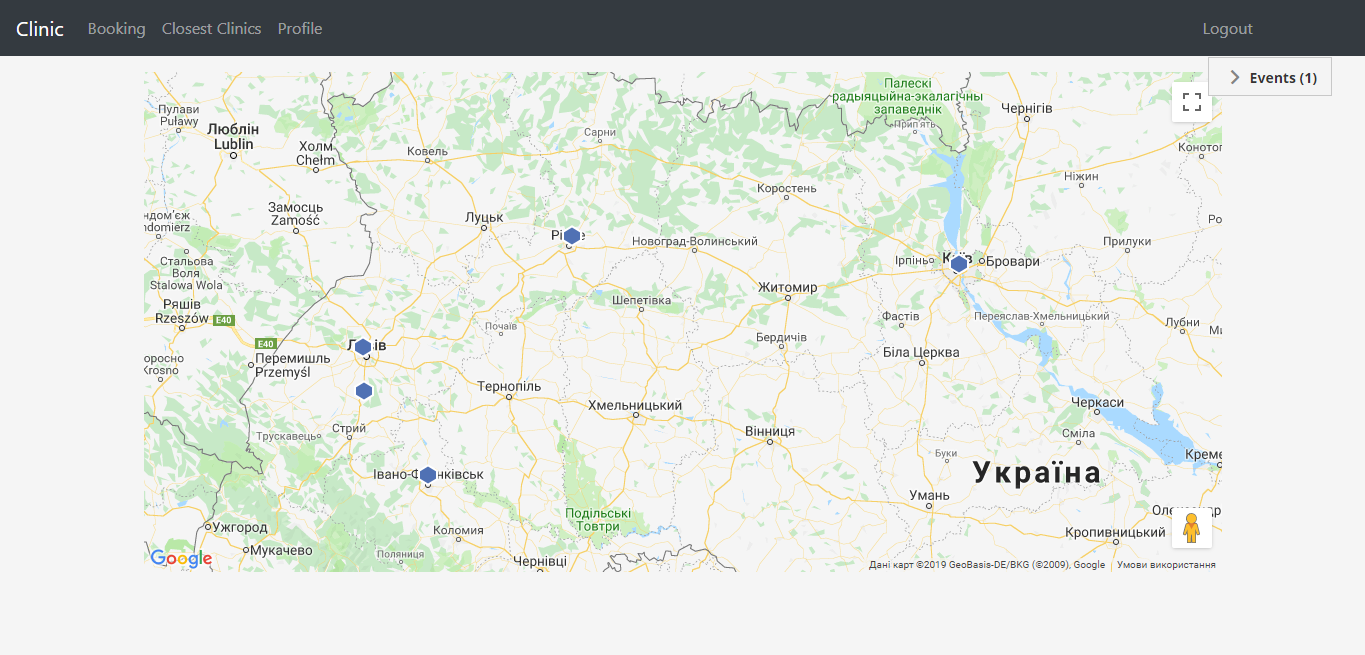
Для визначення відстані (d) між точками, знаючи їх географічні координати, використовувалася наступна формула:

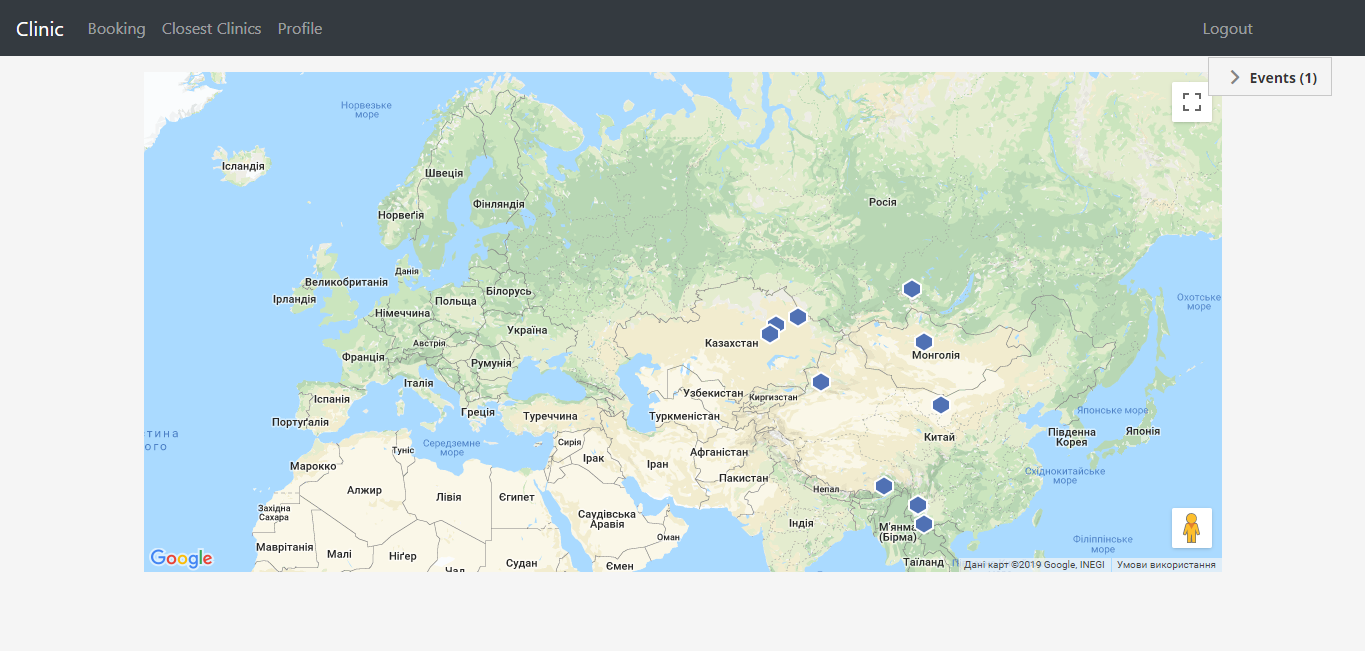
d = R \* c

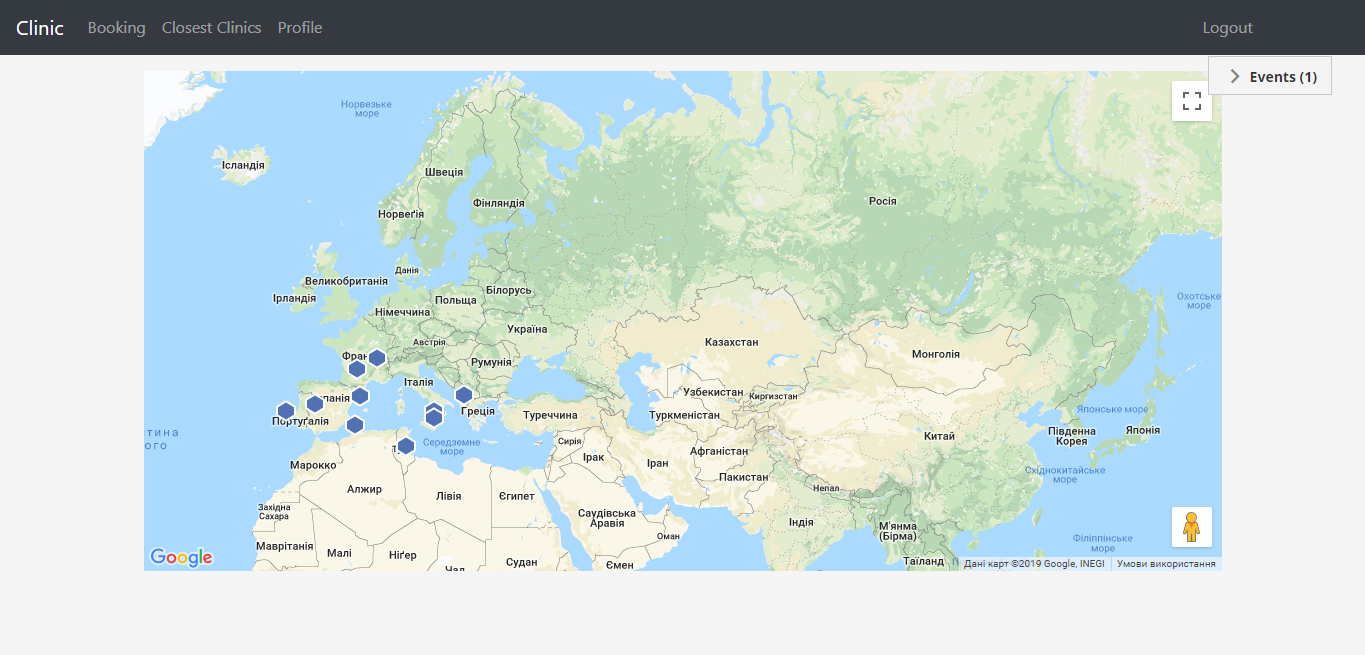
де:

R – радіус Землі (6371 км)

Приклади відображення найближчих (на карті відображаються десять найближчих та можна вказати найближчі до якої точки)







**П.2** **Знаходження найкоротшого шляху до клініки (генетичний алгоритм)**

Найближчий шлях знаходиться із використанням генетичного алгоритму.

Генетичний алгоритм – це в першу чергу еволюційний алгоритм, іншими словами, основна фішка алгоритму – схрещування(комбінування). Як не складно здогадатися – така ідея взяти з природи. Шляхом перебору і найважливіше перебору отримується правильна комбінація.

Алгоритм ділиться на три етапи:

* Схрещування
* Селекція (відбір)
* Формування нового покоління

Якщо результат не влаштовує, ці етапи повторюються до тих пір, поки результат не буде оптимальним або :

* Кількість поколінь досягне якось визначеного максимуму
* Витрачено час на мутацію

Розглянемо ці кроки детальніше

1. Створення нової популяції: На цьому кроці створюється початкова популяція. Мається на увазі декілька можливих шляхів між точками на карті за допомогою загально-доступного сервісу Google API Directions Service. Деякі з них можуть бути не оптимальні але алгоритм це виправить.
2. Розмноження/Мутації: Модифікуємо отримані можливі шляхи, модифікуючи їх використовуючи значення окремих частин шляху із інших можливих варіантів.
3. Відбір(селекція): На цьому етапі із сформованих варіантів вибираємо той який підходить краще. Кількість варіантів, отриманих після відбору, визначаємо завчасно.

Цей алгоритм застосовується для вирішення багатьох задач. Розглянемо приклад застосування цього алгоритму для розв’язування рівнянь.

Нехай потрібно розв’язати наступне рівняння: a + 2b + 3c + 4d = 30

Зрозуміло що корені цього рівняння лежать на проміжку [1;30], томі для початкової популяції візьмемо 5 довільних можливих роз’язків:

1. (1,28,15,3)
2. (14,9,2,4)
3. (13,5,7,3)
4. (23,8,16,19)
5. (9,13,5,2)

Для того щоб обчислити коефіцієнти виживання (що будуть враховуватися на етапі відбору), підставимо кожне значення у рівняння. Відстань отриманого значення до 30 і буде потрібним значенням:

1. |114-30|=84
2. |54-30|=24
3. |56-30|=26
4. |163-30|=133
5. |58-30|=28

Менші значення – ближчі до 30, відповідно вони ближчі до оптимального розв’язку. Тобто більші значення будуть мати менший коефіцієнт виживання. Для створення системи обчислимо імовірність вибору кожного. Тобто візьмемо суму обернених коефіцієнтів та обчислити із цього відсотки (*0.135266 – сума обернених коефіцієнтів):*

1. (1/84)/0.135266 = 8.80%
2. (1/24)/0.135266 = 30.8%
3. (1/26)/0.135266 = 28.4%
4. (1/133)/0.135266 = 5.56%
5. (1/28)/0.135266 = 26.4%

Далі будемо обирати п’ять пар батьків, у яких буде рівно по одному нащадку. Шанс для певного набору чисел стати батьківським еквівалентний коефіцієнту виживання: 3-1, 5-2, 3-5, 2-5, 5-3

Нащадок міститиме інформацію про гени батька й матері. Є багато варіантів передачі інформації потомку. Використаємо підхід “кросс-овер”, що полягає в розділі генів на дві частини:

* **Х.-батько:** a1 | b1,c1,d1 **Х.-мати:**a2 | b2,c2,d2 **Х.-нащадок:** a1,b2,c2,d2 or a2,b1,c1,d1
* **Х.-батько:** a1,b1 | c1,d1 **Х.-мати:**a2,b2 | c2,d2 **Х.-нащадок:** a1,b1,c2,d2 or a2,b2,c1,d1
* **Х.-батько:** a1,b1,c1 | d1 **Х.-мати:**a2,b2,c2 | d2 **Х.-нащадок:** a1,b1,c1,d2 or a2,b2,c2,d1

Зробимо це ж саме із заданими значеннями:

* **Х.-батько:** (13 | 5,7,3) **Х.-мати:**(1 | 28,15,3) **Х.-нащадок:** (13,28,15,3)
* **Х.-батько:** (9,13 | 5,2) **Х.-мати:**(14,9 | 2,4) **Х.-нащадок:** (9,13,2,4)
* **Х.-батько:** (13,5,7 | 3) **Х.-мати:**(9,13,5 | 2) **Х.-нащадок:** (13,5,7,2)
* **Х.-батько:** (14 | 9,2,4) **Х.-мати:**(9 | 13,5,2) **Х.-нащадок:** (14,13,5,2)
* **Х.-батько:** (13,5 | 7, 3) **Х.-мати:**(9,13 | 5, 2) **Х.-нащадок:** (13,5,5,2)

Тепер обчислимо коефіцієнти виживання потомків:

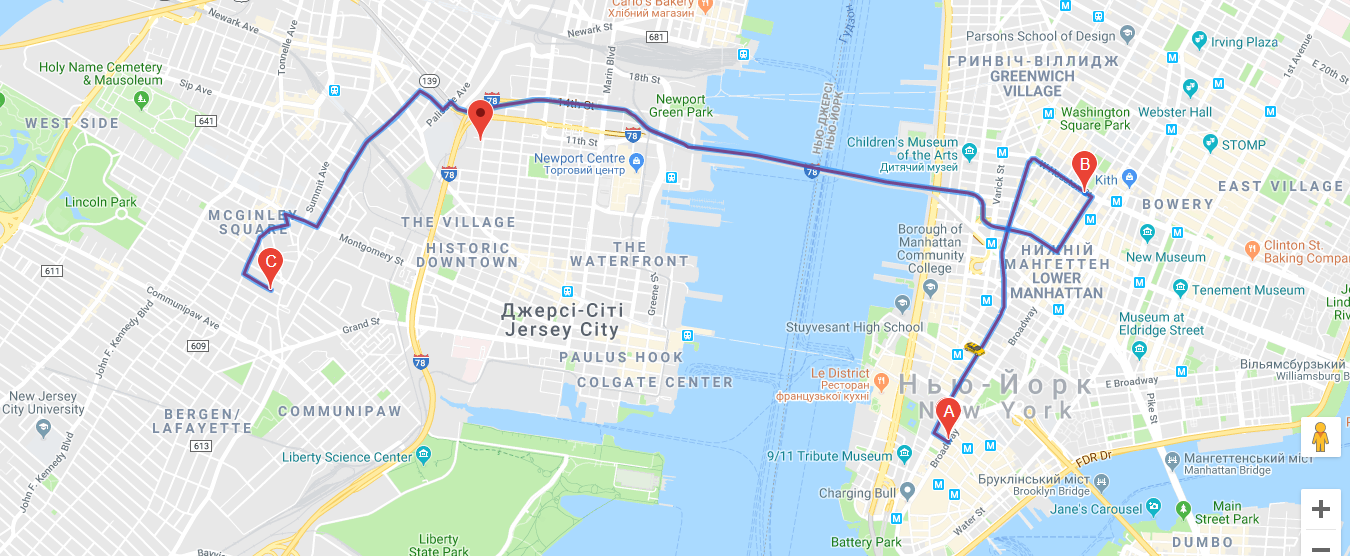
* (13,28,15,3) — |126-30|=96(9,13,2,4) — |57-30|=27
* (13,5,7,2) — |57-30|=22
* (14,13,5,2) — |63-30|=33
* (13,5,5,2) — |46-30|=16

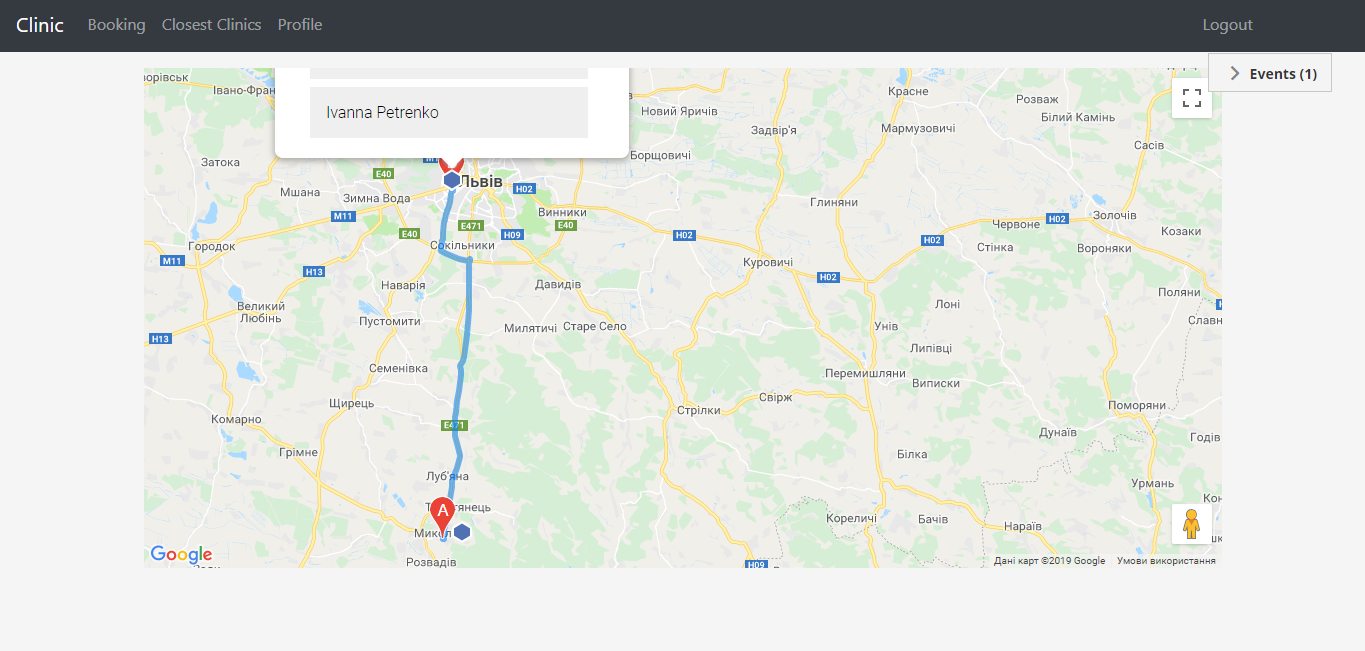
Замітимо що середнє значення коефіцієнту виживання в утвореного покоління менше ніж у батьківського, тому для вирішення цієї проблеми проводиться мутація: заміна одного чи більше генів (чисел) на інше із діапазону [1,30]

Алгоритм буде працювати поки коефіцієнт виживання не буде рівний нулю. В цьому випадку отримані покоління із таким коефіцієнтом будуть розв’язками даного рівняння.

В моєму завданні (визначенні найкращого шляху) генами поколінь є значення проміжних точок шляху. Початкова ініціалізація шляхів відбувається використовуючи загально-доступний сервіс від компанії Google: Directions Service, що повертає масив можливих шляхів. При мутації поколінь добавляються нові точки на карті через які потрібно пройти на шляху між заданими точками. При схрещуванні (утворенні нових поколінь, варіантів шляху між заданими точками) можливі шляхи обмінюються деякими точками на карті, які вони включають. Також слід зазначити що програмна реалізація включає можливість вказання декількох точок на карті, але у застосуванню в основному буде застосовуватися для двох (пацієнт – клініка)

Отримані результати є наступними:





Висновки:

Список літератури