# Technologies and tools used

* SQL Server for the persistent storage.
* Visual Studio 2019 community edition.
* PostMan, for initial checking of the web api calls.

# System Architecture

The central piece of the implementation is based around the ASP.NET Core Web Application project template set up to provide a REST API. The patient records to be added are sent via an HTTP POST and the system utilises the entity framework and SQL server to provide persistent storage. The source for the API is to be found in the Patient project in the solution.

As implemented the API accepts POST requests on the <http://localhost:5000/api/Patient> URL. With JSON body of the following form

{

"User":"Johns",

"ForeName":"Rob",

"Surname":"Smith",

"DateOfBirth":"1961-12-01",

"PrimaryContactNumber":"07700900083",

"PrimaryAddressLine1":"2203 Nowhere Street",

"PrimaryAddressLine2":"Notown",

"PrimaryAddressLine3":"Nowhereshire",

"PostCode":"ZZ15 5AA"

}

On receipt of the POST request the following validations are done

1. User is verified against User table in the database. If this fails a 400 Bad request message is returned with an invalid user error message in the body.
2. The following fields have basic validation done and if any of them fail a 400 Bad request message is returned with an error message in the body that covers the failings
   * Forename must be present.
   * Surname must be present.
   * Primary contact number must be present and consist of only numerals.
   * Primary Address line one must be present.
   * Postcode must be present and have two groups of uppercase and numeric characters separated by a space.
3. It is assumed that the combination of Forename, Surname and date of birth is unique for a patient regardless of other details. If a database row already exists with that combination a 400 Bad Request message is returned with the failing details in the message.

Note that Date of birth is not validated directly as it get deserialised prior to hitting the controller code for the POST request in non-user code. An error there generates an exception return that is not particularly user friendly.

Once past validation the Patient, Phone and Address are split out for insertion into separate tables as required.

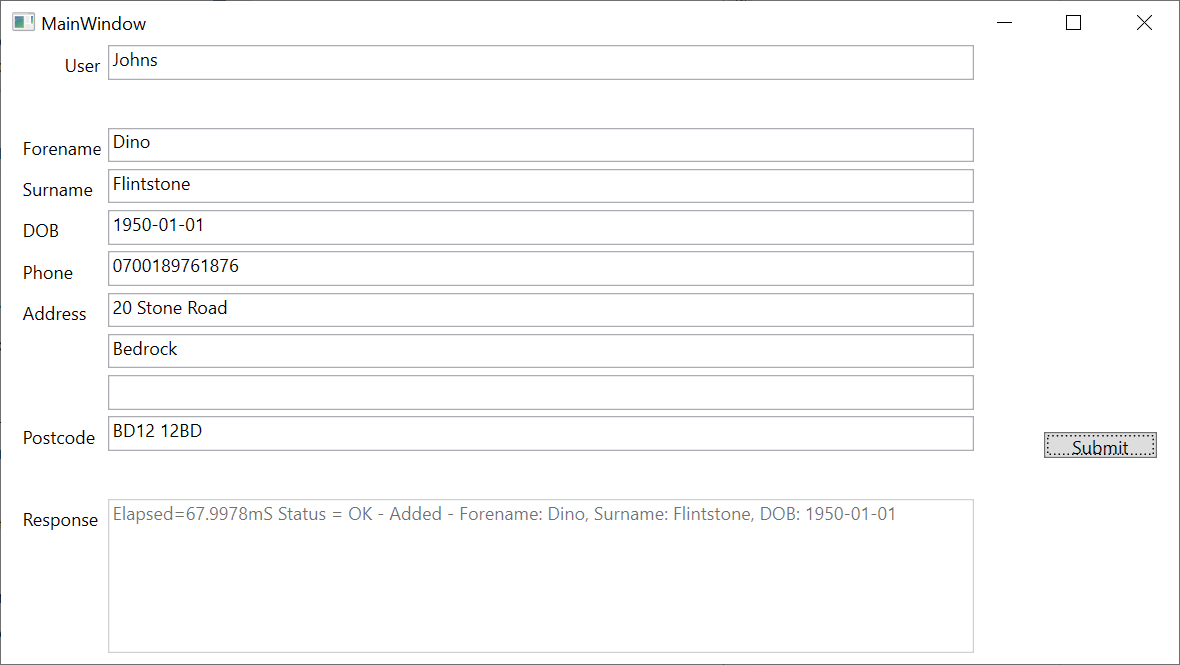
1. The assumption with the address is the first line of the address and postcode should uniquely identify the address. If already present the pre-existing row’s ID will be added to the patient data for its reference otherwise a new ID will be generated for the supplied address.
2. The contact phone number is dealt with in a similar manner.
3. The new Patient, Phone and Address rows are then inserted into the database.
4. An audit log record is inserted into the database, which details the user, patient and date/time along with an “Add” indication and Human readable notes.
5. Finally a 200 OK response with some identifying details in the message is returned.

# GUI Client

The GuiClient project in the solution contains a simple GUI application that utilises the API. The assumption is the API is running on the local machine.

All the entry fields just take text and there is no constraints placed on values entered.

Clicking the submit button sends the details to the API and once the response comes back the Response field shows the details.



# Further extensions to the functionality

Potential extensions include

* Patient details retrieval by record ID or search.
* Being able to update the patient details.
  + Name updates for name changes for whatever reason.
  + Address and Phone changes.
  + Possibly delete details.
* Support for reporting/search functions.
* HTTPS support should really be added.
* Proper authentication of the User entering the data.

# Setup

The SQL scripts from creating the local databases are to be found under the SQL directory.

* CreatePatientDB.sql – Can used to create the Patient database.
* CreatePatientTestDB.sql – Can used to create the PatientTest database.
* PatientDB.sql – creates the tables for the Patient database.
* PatientTestDB.sql - creates the tables for the PatientTest database.
* UserData.sql – populates the User table with a user in the Patient database.
* UserDataTest.sql - populates the User table with a user in the PatientTest database.

The “PATIENT\_API\_TEST\_MODE” environment can be set to change which database is used. If set to “TRUE” it will use the test database. The connections strings used can be altered in the appsettings.json file in the Patient project.

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=Patient;Trusted\_Connection=True;",

"TestConnection": "Server=localhost;Database=PatientTest;Trusted\_Connection=True;"

}