**Cross-table parametrised SQL**

Cross table parametrised SQL is used throughout the WebAPI, with pieces of SQL code being used to update other tables. This is often done through two SQL statements (for example in accountsForHelper.cs in AddNewUser where the timetableId is found through one query and used to change the value in another table).

They are also used as part of joins in a couple of places. One such example is in reviews.cs in returnReviewBasedOnAppointmentId. Here, the contents of a review are returned using the appointment id of which the rating is made.

(These aren’t discussed at all very much in design as they felt rather unimportant and not very important in the scheme of things).

**Aggregate SQL functions**

Two forms of aggregate functions are used in the WebAPI. The first involves the use of inbuilt aggregate functions, with max(x) being used in particular to get data to change information in other parts of the database. One such example of this is in accountsForHelper.cs in AddNewUser where the timetableId is found through one query and used to change the value in another table). Another aggregate function which is used frequently is the count(x) function, which is often used to check for errors. One such example of this is in AddNewUser of accountsForElderly.cs and accountsForHelpers.cs where the functions are used to check whether accounts of the same telephone number and email address exist already.

The other form of aggregate function which exists is through the use of subqueries which are used to simplify the code. One such example of this is in addAreview in reviews.cs where the id to which the rating should be updated is found by using another aggregate function within this subquery to find the new review which has just been added.

**Complex data model in database**

Several tables with connections have been used to store all the data, with the Entity Relationship diagram showing all the tables and the connections (2.2.1). One part of the database which offers particular complexity is the undefined nature of the timetables (labelled weeklyTimetables in the ER diagram) where the columns are added and deleted based on when the custom weeks are required. This is discussed in the timetables part of 2.2.1.

Accessing the column names of the table occurs in the GetTimetable function of timetables.cs and altering the table (setting the column name which can be later parsed back into the names of the starting date of the week) occurs in the AddTimetable function of the WebAPI.

**User / CASE-generated DDL script**

Case generated DDL script is used throughout the WebAPI with the inputs to the HTTP GET, POST and PUT functions being used as a part of the SQL command to be executed.

One particular example is in the AddNewUser function of both the accountsForElderly.cs and accountsForHelpers.cs files with the SQL function being populated with the parts of the user which is POSTed to the function.

**List operations**

Lists are used throughout the WebAPI and the Android Application with them always being used where there are an unknown number of items to be added to a list. In the Android application, they are often used before being converted to an array to be used to populate a ListAdapter, for example in the bookAppointment.cs activity – in the OnCreate function.

An example of lists being used in the WebAPI is in the GetTimetable function is used where there are an unknown number of weeks in the custom timetable (using the names item). Another example is in the ratings.cs file in the function getAllRatingsForUser where there is an unknown number of ratings to be returned.

A large number of lists are also used in the mergeSort function of accountsForHelper when lists are split into two halves each time the function is called, using the Add(x) and RemoveAt(x) functions.

**Recursive Algorithms**

There are a number of recursive algorithms which are used in the WebAPI. One prime example of this is in the mergeSort algorithm of the accountsForHelper.cs file, where the first half and second half is merged after being recursively sorted.

Recursion is also used in the createNewAppointmentsForAndroidApp function of appointments.cs where if a new week must be added, once this has been found out, the new week is added and then the entire process is repeated now, knowing that the week already exists in the timetables table. Therefore, this function is recurred on, at most once in the system. The recursion is checked for, with a variable ensuring that the appointment information is not readded to the appointment database.

This is discussed in the ‘Create New Appointment From AndroidApp’ in 2.3.4 in the report.

**Mergesort or similarly efficient sort**

A MergeSort is used to sort the helpers by how near they are to the elderly person, being used in the mergeSort function of accountsForHelpers.cs of the WebAPI.

It is discussed in the design part of the report in 2.3.4 in ‘Returning List of Nearest Helpers’

**Dynamic generation of objects based on complex user-defined use of OOP model**

Multiple classes are used throughout both the Android Application and the WebAPI, with a number of processes used in the system. Inheritance is used for controllers with the hierarchy of this defined in 2.3.1.

Classes are also used for all the types that are used for the system, from the days, weeks and timetables and information about the users. For these composition is generally used. One example of composition being used widely (it is really used everywhere through the WebAPI) is in the GetTimetable function of timetables.cs.

It is also used in the Android Application, for example in the AppointmentsActivity.cs in the activities of the AndroidApp, where user, timetable and appointment items are used in the activity. (Again, composition is generally used in almost every activity…)

**Server-side scripting using request and response objects and server-side extensions for a complex client-server model**

A .NET core (REST) API is produced to deal with requests and acts as a server to return all the information which is required. This was discussed in 2.3.1 and throughout 2.3 discussing the design of the API. An example of dealing with requests is in any one of the controllers of the .NET core API, for example the ratings.cs which includes GET, POST, PATCH and DELETE HTTP request handlers.

Additionally, another WebAPI, this time using Python is also created using Flask. This is discussed in the design in 2.5 and can be seen as an example in app.py in the messagingServer system.

**Calling parameterised Web service APIs and parsing JSON / XML to service a complex client-server model**

In the course of the system, three different parametrised Web APIs are called and data returned from. The first is the messaging API which deals with JSON POSTed to it by the Android App (register function in the registerActivity of the Android Application) using the request library (for an example, see the verifyNumber function of the messagingServer API), parsing it and using it. Data is also sent to it (in JSON form) by the WebAPI in the Backend folder (SmsStuff.cs) using the System.Net.Http library.

The second WebAPI which is used is the .NET Core API which communicated in JSON (both for the incoming requests and for the return data from the system). An example of parsing JSON by the .NET core library is using the [FromBody] keyword in the server, for example in the AddDay function (picked entirely at random, basically every function uses it…) in the timetables.cs controller. On the Android Application side, the JSON from the WebAPI is parsed into a custom object using the Newtonsoft.JSON library, for example in the GetUserDetails function of helperDetails.cs activity of the Android Application.

Finally, the GoogleMapsAPI is used in the accountsForHelpers.cs controller of the WebAPI in the GetDistance function of the helper. The NewtonSoft.Json function is used to parse the information from a specific part of the response from the WebAPI.