

## Faculty of Engineering and Computer Science Expectations of Originality

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1. Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2. Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation<sup>1</sup>. Note that engineering reports rarely contain direct quotations.
3. Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4. Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5. Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6. No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7. In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8. Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9. Your submissions cannot be edited or revised by any other student.
10. For lab reports, the data must be obtained from your own or your lab group's experimental work.
11. For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.

You must write one of the following statements on each piece of work that you submit:

For individual work: **"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"**, with your signature, I.D. #, and the date.

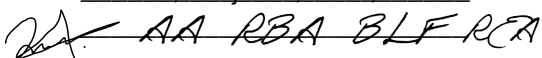
For group work: **"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"**, with the signatures and I.D. #s of all the team members and the date.

A signed copy of this form must be submitted to the instructor at the beginning of the semester in each course.

I certify that I have read the requirements set out on this form, and that I am aware of these requirements. I certify that all the work I will submit for this course will comply with these requirements and with additional requirements stated in the course outline.

Course Number: COMP353 - Databases

Name: Antoine, Benjamin, Rami, Ribelle

Signature: 

Instructor: SHIRI VARNAAMKHAASTI

I.D. # 40022745;40008156;40043011;29276726

Date: 16th April 2021

<sup>1</sup> Rules for reference citation can be found in "Form and Style" by Patrich MacDonagh and Jack Bordan, fourth edition, May, 2000, available at <http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf>.

# **COMP 353, Winter 2021 Team Project - Final Project**

## **Team 4 (ydc353\_4)**

April 18, 2021

Name	ID Number	GitHub Username
Antoine Assal	40022745	@As.Antoine
Benjamin Lofu Follo	40008156	@lamungu
Rami Bou Abboud	40043011	@Mr Robot
Ribelle El Ayoubi	29276726	@Belle-coder

# Contents

<b>1</b>	<b>Assumptions</b>	<b>6</b>
<b>2</b>	<b>E/R diagram</b>	<b>6</b>
2.1	Constraints captured by the diagram . . . . .	7
2.2	Constraints not captured by the diagram . . . . .	7
2.3	A different notation . . . . .	7
<b>3</b>	<b>Relational Database Schema</b>	<b>8</b>
3.1	Relational Model from Warm-up Project . . . . .	8
3.2	Relational Model with Updated Information . . . . .	8
3.3	Model's Normal Form . . . . .	9
3.4	Few Relation instances . . . . .	15
<b>4</b>	<b>Normalization</b>	<b>18</b>
4.1	Functional Dependency . . . . .	18
4.2	Normalization Steps . . . . .	18
4.3	Final Normalized Model . . . . .	20
4.4	Analysis of 3NF and BCNF . . . . .	21
<b>5</b>	<b>Functionalities Implemented to satisfy requirements</b>	<b>22</b>
5.1	Patient CRUD . . . . .	23
5.2	Public Health Worker CRUD . . . . .	25
5.3	Facility CRUD . . . . .	28
5.4	Region CRUD . . . . .	30
5.5	Group-Zone CRUD . . . . .	31
5.6	Recommendation CRUD . . . . .	32
5.7	Trigger for Alerts . . . . .	34
5.8	Trigger for Diagnostic Result . . . . .	35
<b>6</b>	<b>Scripts &amp; Queries</b>	<b>36</b>
6.1	Form Followup . . . . .	36
6.2	Symptoms Progress . . . . .	38
6.3	Messages during a given time . . . . .	38
6.4	People in an address . . . . .	40
6.5	Details of all facilities . . . . .	41
6.6	Details of all Regions . . . . .	42
6.7	Details of people's test result in a date ordered by result . . . . .	44
6.8	List of all workers in a Facility . . . . .	44
6.9	Workers with COVID at a facility on date and their reach . . . . .	45
6.10	Report for every region . . . . .	45

<b>7</b>	<b>User Interface</b>	<b>47</b>
7.1	Linked Resources . . . . .	47
7.2	Backend Component: Laravel . . . . .	47
7.3	Frontend Component: ReactJS . . . . .	48
7.4	Authentication . . . . .	48
7.5	CRUD Table . . . . .	48
7.6	Create/Update Entity . . . . .	49
7.7	Toast notification . . . . .	50
7.8	Region Update . . . . .	50
<b>8</b>	<b>Bonus: Additional feature</b>	<b>51</b>
8.1	Twilio SMS API Integration . . . . .	51
<b>9</b>	<b>Contribution &amp; Separation of work</b>	<b>55</b>
<b>10</b>	<b>References</b>	<b>56</b>

## List of Figures

1	E/R diagram . . . . .	6
2	A diagram that captures some more constraints (NOT AN ERD) . . . . .	7
3	Filling up a form . . . . .	37
4	Display all messages generate during in a given time . . . . .	40
5	Patients in an address . . . . .	41
6	Displaying details of all facilities . . . . .	42
7	Displaying details of all regions . . . . .	44
8	Landing Page of the User . . . . .	47
9	Home Page allows you to login with your medicare and dob . . . . .	48
10	List of all workers in a table . . . . .	49
11	This edit page provides feedback on what has not been filled out by the user . . . . .	50
12	A toast notification . . . . .	50
13	Edit Region Form . . . . .	51
14	Phone Configuration in the Twilio Console . . . . .	53
15	Our webapp sends text messages using the Twilio SMS API . . . . .	53
16	The command executes to process any unsent messages in the database . . . . .	53
17	Our webapp sends text messages using the Twilio SMS API . . . . .	54

## List of Tables

1	Un-normalized person relation instance . . . . .	15
2	Normalized person relation instance . . . . .	15
3	Un-normalized patient relation instance . . . . .	15
4	Normalized patient relation instance . . . . .	15
5	Un-normalized carer relation instance . . . . .	15
6	Normalized carer relation instance . . . . .	15
7	Un-normalized PublicHealthWorker relation instance . . . . .	16
8	Normalized PublicHealthWorker relation instance . . . . .	16
9	Normalized Position relation instance . . . . .	16
10	Un-normalized PublicHealthCenter relation instance . . . . .	16
11	Normalized PublicHealthCenter relation instance . . . . .	16
12	Normalized PostalCode relation instance . . . . .	16
13	Normalized City relation instance . . . . .	16
14	Un-normalized Region relation instance . . . . .	17
15	Normalized Region relation instance . . . . .	17
16	Normalized Province relation instance . . . . .	17
17	Follow-up form representative tuples . . . . .	37
18	Symptom progress representative tuples . . . . .	38
19	Messages generated in a given time representative tuples . . . . .	39
20	People living in the same address representative tuples. . . . .	41

# 1 Assumptions

Show your reasonable Assumptions made in the proposed database design.

- Every user in our database is a Person. However, for a Person to exist they must be a Patient or a Health Care worker.
- A Healthcare worker can be a patient, not mutually exclusive.
- Auto-incrementing identifiers for our entities avoids user input which eliminates the possibility of errors of having multiple instance of an entity with the same ID.
- Our database system will be used to cover Canada and it's region/cities.
- Our data focuses on Quebec.
- Regions, GroupZones, and Recommendation do not need a delete call in CRUD.

# 2 E/R diagram

E/R diagram to represent the conceptual database design for the application.

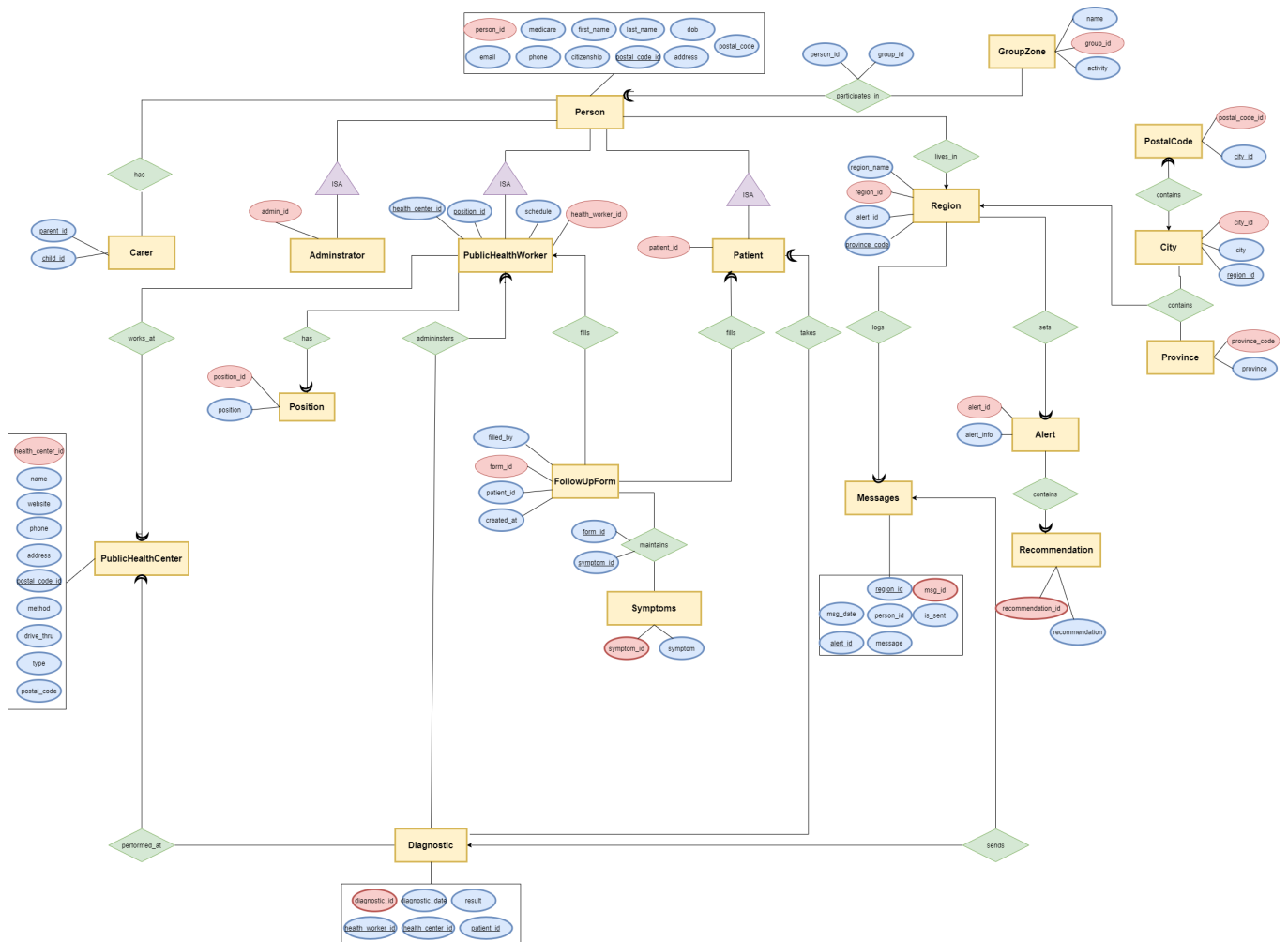


Figure 1: E/R diagram

## 2.1 Constraints captured by the diagram

- Multiplicity of relationships (pointed arrow)
- Inheritance (Triangle)
- Referential Integrity (Curved arrow)
- Primary keys (red bubble)
- Foreign keys (underlined attribute)

## 2.2 Constraints not captured by the diagram

One of the limitations we came across when using this notation is having referential integrity and multiplicity pointing to the same entity and we had cases where referential integrity and multiplicity of relationships conflicted.

- We prioritized showing the referential integrity constraint.
- We did not show referential integrity on the ERD when it comes to inherited entity sets.
- The diagram does not capture data types.
- The diagram does not capture Nullable attributes.
- The diagram does not show functional dependencies.

## 2.3 A different notation

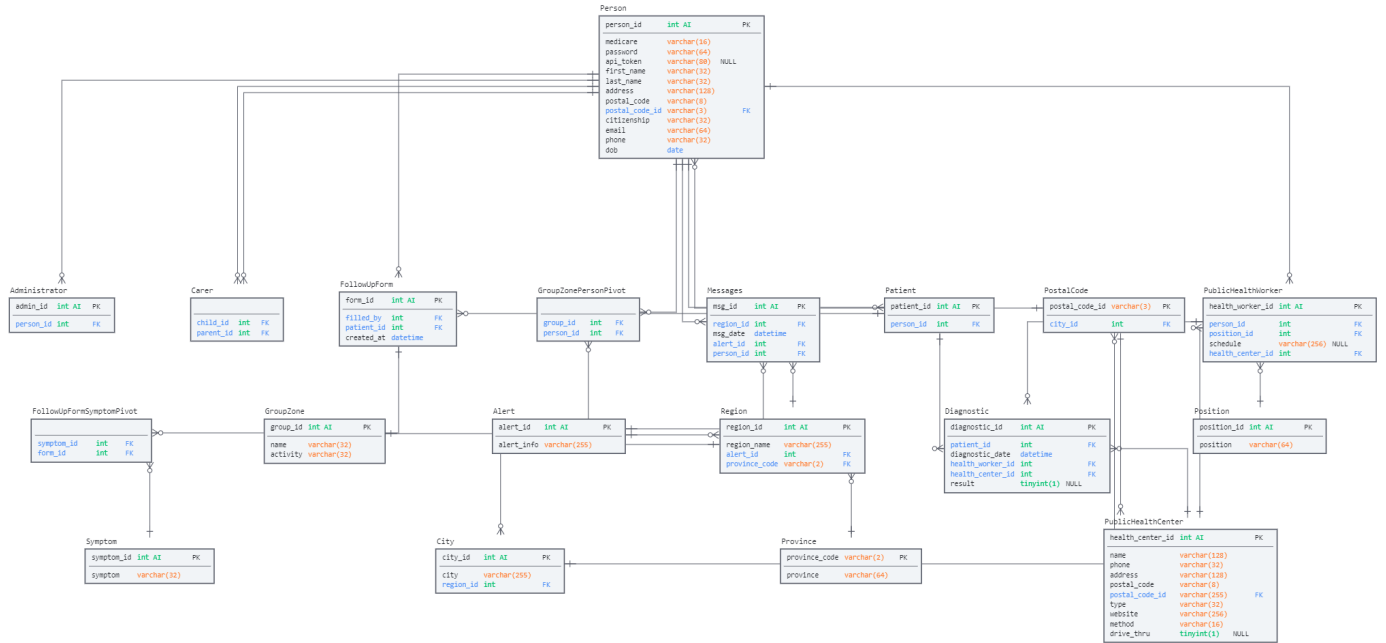


Figure 2: A diagram that captures some more constraints (NOT AN ERD)

### 3 Relational Database Schema

Convert your E/R diagram into a relational database schema. Make refinements to the DB schema if necessary. Identify various integrity constraints such as primary keys, foreign keys, functional dependencies, and referential constraints. Make sure that your database schema is at least in 3NF.

#### 3.1 Relational Model from Warm-up Project

This is the Relational schema we came up with for the warm-up project. Underline represents a primary key and # represents a foreign key.

- Person (person\_id, medicare, first\_name, last\_name, address, city, postal\_code, province, citizenship, email, phone, dob)
- Parental (parental\_id, #person\_id, parent\_1\_id, parent\_2\_id, guardian\_id, child\_id)
- PublicHealthWorker (health\_worker\_id, position, schedule, #health\_center\_id, #person\_id)
- PublicHealthCenter (health\_center\_id, phone, name, address, city, province, postal\_code, type, website)
- Diagnostic (diagnostic\_id, diagnostic\_date, result, #health\_worker\_id, #health\_center\_id, #person\_id)
- Patient (patient\_id, #person\_id)
- GroupZone (group\_id, activity, name)
- GroupZonePersonPivot (group\_id, person\_id)

#### 3.2 Relational Model with Updated Information

For this project, we read the instructions carefully and started building on top of our schema to have a new one that includes all the new required information for our system.

- Person (person\_id, medicare, first\_name, last\_name, address, city, postal\_code, province, citizenship, email, phone, dob, password, api\_token)
- Patient (patient\_id, #person\_id, symptoms, symptoms\_history)
- Carer (parental\_id, #person\_id, parent\_1\_id, parent\_2\_id, guardian\_id, child\_id)
- PublicHealthWorker (health\_worker\_id, position, schedule, #health\_center\_id, #person\_id)
- PublicHealthCenter (health\_center\_id, phone, name, address, city, province, postal\_code, type, website, drive\_thru)
- Diagnostic (diagnostic\_id, diagnostic\_date, result, #health\_worker\_id, #health\_center\_id, #patient\_id)
- FollowupForm (form\_id, #health\_worker\_id, #patient\_id)
- GroupZone (group\_id, activity, name)
- Region (region\_id, region\_name, city, postal\_code, province)
- Alert (alert\_id, alert\_name, alert\_level, alert\_region, alert\_status, alert\_date, alert\_time, current\_alert, past\_alert, message, #person\_id, recommendation)



### 3.3 Model's Normal Form

It is safe to say that our model is at least in **1NF** for all the relations since all attributes are atomic and cannot contain more than one value. The only exception is symptoms & symptoms\_history, which will be normalized later in this document.

#### 3.3.1 Person

For the Person Relation we have the following non-trivial dependencies:

- $\{person\_id\} \rightarrow first\_name$
- $\{person\_id\} \rightarrow last\_name$
- $\{person\_id\} \rightarrow medicare$
- $\{person\_id\} \rightarrow password$
- $\{person\_id\} \rightarrow address$
- $\{person\_id\} \rightarrow postal\_code$
- $\{person\_id\} \rightarrow citizenship$
- $\{person\_id\} \rightarrow email$
- $\{person\_id\} \rightarrow phone$
- $\{person\_id\} \rightarrow dob$
- $\{person\_id\} \rightarrow api\_token$
- $\{address\} \not\rightarrow first\_name$
- $\{email\} \not\rightarrow person\_id$

person\_id cannot be determined, is not on the RHS of any of the FDs, making it a super-key and the only candidate key.

**So our  $F$  is:**

$$F = \{ person\_id \rightarrow first\_name, last\_name, medicare, password, \\ address, postal\_code, citizenship, email, phone, dob, api\_token \} \quad (1)$$

$$person\_id^+ = \{ person\_id, first\_name, last\_name, medicare, password, \\ address, postal\_code, citizenship, email, phone, dob, api\_token \} \quad (2)$$

Key = *person\_id*

This relation is **in 2NF**.

This relation is **in 3NF**.

This relation is **in BCNF**.

### 3.3.2 Carer

For the Carer Relation we have the following non-trivial dependencies:

- $\{person\_id\} \rightarrow child\_id$
- $\{person\_id\} \rightarrow parent\_1\_id$
- $\{person\_id\} \rightarrow parent\_2\_id$
- $\{person\_id\} \rightarrow guardian\_id$
- $\{parent\_1\_id\} \rightarrow child\_id$
- $\{parent\_2\_id\} \rightarrow child\_id$
- $\{parent\_1\_id, parent\_2\_id\} \rightarrow child\_id$
- $\{parent\_1\_id, parent\_2\_id, guardian\_id\} \rightarrow child\_id$

So our  $F$  is:

$$F = \{ \{ person\_id \rightarrow child\_id, parent\_1\_id, parent\_2\_id, guardian\_id \}, \\ \{ parent\_1\_id \rightarrow child\_id \}, \{ parent\_2\_id \rightarrow child\_id \}, \\ \{ parent\_1\_id, parent\_2\_id, guardian\_id \rightarrow child\_id \} \} \quad (3)$$

This relation is **not in 2NF** because we have partial dependencies,  
For example  $person\_id \rightarrow parent\_1\_id$  and we can split it into more tables.

### 3.3.3 PublicHealthWorker

For the PublicHealthWorker Relation we have the following non-trivial dependencies:

- $\{health\_worker\_id\} \rightarrow position$
- $\{health\_worker\_id\} \rightarrow schedule$
- $\{health\_worker\_id\} \rightarrow health\_center\_id$
- $\{health\_worker\_id\} \rightarrow person\_id$
- $\{health\_center\_id\} \not\rightarrow health\_worker\_id$

$health\_worker\_id$  cannot be determined, is not on the RHS of any of the FDs, making it a super-key and the only candidate key.

So our  $F$  is:  $F = \{health\_worker\_id \rightarrow position, schedule, health\_center\_id, person\_id\}$

$$health\_worker\_id^+ = \{health\_worker\_id, person\_id, position, schedule, health\_center\_id\}$$

Key =  $health\_worker\_id$

This relation is **in 2NF**.

This relation is **in 3NF**.

This relation is **in BCNF**.

### 3.3.4 PublicHealthCenter

For the PublicHealthCenter Relation we have the following non-trivial dependencies:

- $\{\text{health\_center\_id}\} \rightarrow \text{phone}$
- $\{\text{health\_center\_id}\} \rightarrow \text{name}$
- $\{\text{health\_center\_id}\} \rightarrow \text{address}$
- $\{\text{health\_center\_id}\} \rightarrow \text{city}$
- $\{\text{health\_center\_id}\} \rightarrow \text{province}$
- $\{\text{health\_center\_id}\} \rightarrow \text{postal\_code}$
- $\{\text{health\_center\_id}\} \rightarrow \text{type}$
- $\{\text{health\_center\_id}\} \rightarrow \text{website}$
- $\{\text{health\_center\_id}\} \rightarrow \text{drive\_thru}$
- $\{\text{postal\_code}\} \rightarrow \text{city}$
- $\{\text{postal\_code}\} \rightarrow \text{province}$
- $\{\text{city}\} \rightarrow \text{province}$

So our  $F$  is:

$$F = \{ \{\text{health\_center\_id} \rightarrow \text{phone, name, address, city, province, postal\_code, type, website, drive\_thru}\}, \{\text{postal\_code} \rightarrow \text{city, province}\}, \{\text{city} \rightarrow \text{province}\} \} \quad (4)$$

Key = *health\_worker\_id*

This relation is **in 2NF**.

*postal\_code*  $\rightarrow$  *city, province* **violates 3NF rules.**

It is non-trivial, LHS is not a superkey and RHS contains a non-key attribute.

This relation is **not in 3NF**.

This relation is **not in BCNF**.

### 3.3.5 Diagnostic

For the Diagnostic Relation we have the following non-trivial dependencies:

- $\{\text{diagnostic\_id}\} \rightarrow \text{diagnostic\_date}$
- $\{\text{diagnostic\_id}\} \rightarrow \text{result}$
- $\{\text{diagnostic\_id}\} \rightarrow \text{health\_worker\_id}$
- $\{\text{diagnostic\_id}\} \rightarrow \text{health\_center\_id}$
- $\{\text{diagnostic\_id}\} \rightarrow \text{patient\_id}$
- $\{\text{patient\_id}, \text{health\_worker\_id}\} \not\rightarrow \text{diagnostic\_id}$
- $\{\text{patient\_id}, \text{health\_worker\_id}\} \not\rightarrow \text{result}$

**So our  $F$  is:**  $F = \{\text{diagnostic\_id} \rightarrow \text{diagnostic\_date}, \text{result}, \text{health\_worker\_id}, \text{health\_center\_id}\}$   
**Key =  $\text{diagnostic\_id}$**

$\text{diagnostic\_id}$  cannot be determined, is not on the RHS of any of the FDs, making it a super-key and the only candidate key.

This relation is **in 2NF**.

This relation is **in 3NF**.

This relation is **in BCNF**.

### 3.3.6 FollowUpForm

For the FollowUpForm Relation we have the following non-trivial dependencies:

- $\{\text{form\_id}\} \rightarrow \text{health\_worker\_id}$
- $\{\text{form\_id}\} \rightarrow \text{patient\_id}$

$\text{form\_id}$  cannot be determined, is not on the RHS of any of the FDs, making it a super-key and the only candidate key.

**So our  $F$  is:**  $F = \{\text{form\_id} \rightarrow \text{health\_worker\_id}, \text{patient\_id}\}$

$\text{form\_id}^+ = \{\text{form\_id}, \text{health\_worker\_id}, \text{patient\_id}\}$

**Key =  $\text{form\_id}$**

This relation is **in 2NF**.

This relation is **in 3NF**.

This relation is **in BCNF**.

### 3.3.7 GroupZone

For the Groupzone Relation we have the following non-trivial dependencies:

- $\{group\_id\} \rightarrow activity$
- $\{group\_id\} \rightarrow name$

$group\_id$  cannot be determined, is not on the RHS of any of the FDs, making it a super-key and the only candidate key.

**So our  $F$  is:**  $F = \{group\_id \rightarrow activity, name\}$

$group\_id^+ = \{group\_id, activity, name\}$

Key =  $group\_id$

This relation is **in 2NF**.

This relation is **in 3NF**.

This relation is **in BCNF**.

### 3.3.8 Region

For the Region Relation we have the following non-trivial dependencies:

- $\{region\_id\} \rightarrow region\_name$
- $\{region\_id\} \rightarrow city$
- $\{region\_id\} \rightarrow postal\_code$
- $\{region\_id\} \rightarrow province$
- $\{postal\_code\} \rightarrow city$
- $\{postal\_code\} \rightarrow province$
- $\{city\} \rightarrow province$

**So our  $F$  is:**

$$F = \{ \{region\_id \rightarrow region\_name, city, postal\_code, province\}, \\ \{postal\_code \rightarrow city, province\}, \{city \rightarrow province\} \} \quad (5)$$

This relation is **in 2NF**.

$postal\_code \rightarrow city, province$  **violates 3NF rules**.

It is non-trivial, LHS is not a superkey and RHS contains a non-key attribute.

This relation is **not in 3NF**.

This relation is **not in BCNF**.

### 3.3.9 Alert

For the Alert Relation we have the following non-trivial dependencies:

- $\{\text{alert\_id}\} \rightarrow \text{alert\_name}$
- $\{\text{alert\_id}\} \rightarrow \text{alert\_level}$
- $\{\text{alert\_id}\} \rightarrow \text{alert\_region}$
- $\{\text{alert\_id}\} \rightarrow \text{alert\_status}$
- $\{\text{alert\_id}\} \rightarrow \text{alert\_date}$
- $\{\text{alert\_id}\} \rightarrow \text{alert\_time}$
- $\{\text{alert\_id}\} \rightarrow \text{current\_alert}$
- $\{\text{alert\_id}\} \rightarrow \text{past\_alert}$
- $\{\text{alert\_id}\} \rightarrow \text{message}$
- $\{\text{alert\_id}\} \rightarrow \text{person\_id}$
- $\{\text{alert\_id}\} \rightarrow \text{recommendation}$
- $\{\text{alert\_region}\} \rightarrow \text{alert\_status}$
- $\{\text{alert\_region}\} \rightarrow \text{alert\_level}$
- $\{\text{alert\_region}\} \rightarrow \text{message}$
- $\{\text{message}\} \rightarrow \text{recommendation}$
- $\{\text{person\_id}\} \rightarrow \text{message}$

So our  $F$  is:

$$F = \{ \{ \text{alert\_id} \rightarrow \text{alert\_name}, \text{alert\_level}, \text{alert\_region}, \text{alert\_status}, \text{alert\_date}, \text{alert\_time}, \text{current\_alert}, \text{past\_alert}, \text{message}, \text{person\_id}, \text{recommendation} \}, \\ \{ \text{alert\_region} \rightarrow \text{alert\_status}, \text{alert\_level}, \text{message} \}, \\ \{ \text{message} \rightarrow \text{recommendation} \}, \{ \text{person\_id} \rightarrow \text{message} \} \} \quad (6)$$

Key = *alert\_id*

This relation is **in 2NF**.

*alert\_region*  $\rightarrow$  *alert\_status, alert\_level, message* **violates 3NF rules.**

It is non-trivial, LHS is not a superkey and RHS contains a non-key attribute.

This relation is **not in 3NF**.

This relation is **not in BCNF**.

### 3.4 Few Relation instances

#### 3.4.1 Person

api\_token and password are randomly generated numbers or hashed strings. Shortened them for the report.

person_id	medicare	first_name	last_name	address	city	postal_code	province	citizenship	email	phone	dob	password	api_token
4	GBUJ46611976	Anne	Paquette	6577 Sébastien Plains	Sainte-Catherine	G3N 0O9	QC	KN	bstpierre@lapierre.com	1-882-823-4013,	1977-02-08	KxsFhruN	gwh0XS
16	EEKJ96363064	Marc	Gauthier	756 Tessier Prairie	Montreal	H1H 7D2	QC	TF	brigitte92@lambert.com	470-997-1115,	1990-06-29	zd9Znke	xO3TUV

Table 1: Un-normalized person relation instance

person_id	medicare	first_name	last_name	address	postal_code	postal_code_id	citizenship	email	phone	dob	password	api_token
4	GBUJ46611976	Anne	Paquette	6577 Sébastien Plains	G3N 0O9	G3N	KN	bstpierre@lapierre.com	1-882-823-4013,	1977-02-08	KxsFhruN	gwh0XS
16	EEKJ96363064	Marc	Gauthier	756 Tessier Prairie	H1H 7D2	H1H	TF	brigitte92@lambert.com	470-997-1115,	1990-06-29	zd9Znke	xO3TUV

Table 2: Normalized person relation instance

#### 3.4.2 Patient

patient_id	person_id	symptoms	symptoms_history
1	11	cough,fever	cough,fever,loss of taste
6	16	NULL	NULL

Table 3: Un-normalized patient relation instance

patient_id	person_id
1	11
6	16

Table 4: Normalized patient relation instance

#### 3.4.3 Carer

parental_id	person_id	parent_1_id	parent_2_id	guardian_id	child_id
1	4	5	7	NULL	4
2	9	NULL	NULL	11	9

Table 5: Un-normalized carer relation instance

child_id	parent_id
4	5
4	7
9	11

Table 6: Normalized carer relation instance

### 3.4.4 PublicHealthWorker

health_worker_id	person_id	position	schedule	health_center_id
1	18	Doctor	Mon: 19:00-03:00, Tue: 07:00-15:00	4
2	12	Nurse	Thu: 04:00-12:00, Fri: 17:00-01:00, Sat: 09:00-17:00	5

Table 7: Un-normalized PublicHealthWorker relation instance

health_worker_id	person_id	position	schedule	health_center_id
1	18	1	Mon: 19:00-03:00, Tue: 07:00-15:00	4
2	12	2	Thu: 04:00-12:00, Fri: 17:00-01:00, Sat: 09:00-17:00	5

Table 8: Normalized PublicHealthWorker relation instance

### 3.4.5 Position

position_id	position
1	Doctor
2	Nurse
3	Intern

Table 9: Normalized Position relation instance

### 3.4.6 PublicHealthCenter

health_center_id	phone	name	address	city	province	postal_code	type	website	drive_thru
4	(828) 624-4739	Boucher-Jacques	1717 Boucher Well	Saint-Laurent	QC	H4T 4V7	Hospital	www.simard.biz	0
5	+1.903.423.4085	Guay Ltd	T448 Girard Lane Suite 134	Laval	QC	H7N 5N3	Clinic	www.gangon.info	1

Table 10: Un-normalized PublicHealthCenter relation instance

health_center_id	phone	name	address	postal_code_id	postal_code	type	website	method	drive_thru
4	(828) 624-4739	Boucher-Jacques	1717 Boucher Well	H4T	H4T 4V7	Hospital	www.simard.biz	appointment	0
5	+1.903.423.4085	Guay Ltd	T448 Girard Lane Suite 134	H7N	H7N 5N3	Clinic	www.gangon.info	walk-in	1

Table 11: Normalized PublicHealthCenter relation instance

### 3.4.7 PostalCode

postal_code_id	city_id
V1R	229
V2K	244
V7A	304

Table 12: Normalized PostalCode relation instance

### 3.4.8 City

city_id	city	region_id
229	Trail	95
244	Prince George North	101
304	Richmond South	126

Table 13: Normalized City relation instance



### 3.4.9 Region

region_id	region_name	city	postal_code	province
95	Trail	Trail	V1R	British Columbia
101	Prince George	Prince George North	V2K	British Columbia
126	Richmond	Richmond South	V7A	British Columbia

Table 14: Un-normalized Region relation instance

region_id	region_name	alert_id	province_id
95	Trail	1	BC
101	Prince George North	1	BC
126	Richmond South	1	BC

Table 15: Normalized Region relation instance

### 3.4.10 Province

province_id	province
BC	British Columbia
SK	Saskatchewan
QC	Quebec

Table 16: Normalized Province relation instance

## 4 Normalization

Simple definitions for deriving normal forms :

- 1NF : The key (remove repeating groups)
- 2NF : The whole key (remove partial dependence)
- 3NF : And nothing but the key (remove indirect dependence)

### 4.1 Functional Dependency

Already analysed each relation's *FDs* in section 3.3

### 4.2 Normalization Steps

#### 4.2.1 2NF

**Second Normal Form 2NF:** achieved by first making sure the table is in first normal form, and then removing attributes that only partially depend on the key by creating new tables. So the steps are :

- The Database is in **1NF**
- For every non-trivial dependency  $X \rightarrow A$  either
  - $X$  is not a proper subset of any candidate key OR
  - $A$  is a key attribute

**Key attribute:** An attribute that is part of some candidate key.

**Super key:** A combination of fields that can uniquely determine a row

**Candidate key:** A Super key that is also minimal. Meaning that :

- All candidate keys are super keys.
- If we remove any field it's not a super key anymore.

We have already shown that the Carer relation is not in 2NF and needs to be normalized.

**4.2.1.1 Carer** We have redundant attributes, and we can simplify it by getting rid of *parent\_1\_id*, *parent\_2\_id* and *guardian\_id*.

Now our Carer relation is

Carer (#parent\_id, #child\_id)

This relation is now **in 2NF**.

This relation is now **in 3NF**.

This relation is now **in BCNF**.

#### 4.2.2 3NF

**Third Normal Form 3NF:** achieved by first making sure the table is in second normal form, and making sure that there are no non-key attributes that are transitively dependent on any candidate key. So the steps are :

- The Relation is in **2NF**
- If it's in 2NF and its non-key attributes are fully and directly dependent on the key, then it is also in **3NF**
- For every non-trivial dependency  $X \rightarrow A$  either
  - $X$  is a superkey or
  - $A$  is a key attribute

We have already shown that the **PublicHealthCenter**, **Region** and **Alert** relations are not in 3NF and need to be normalized.

##### 4.2.2.1 PublicHealthCenter

- The set of candidate keys for this relation is  $\{health\_center\_id\}$ .
- Merge the *FDs* we provided in 3.3.4, combining the same LHS whose RHS are non-key attributes.
- $R_1 = health\_center\_id \rightarrow drive\_thru, website, type, postal\_code, address, name, phone$
- $R_2 = postal\_code \rightarrow city$
- $R_3 = city \rightarrow province$
- Now  $R_1$  is in 3NF.
- $R_2$  &  $R_3$  violate 3NF since their LHS is not a superkey and RHS is a set of non-key attributes.
- We will make a new table for them:
- Province(province\_code, province)
- City(city\_id, city, #region\_id)
- Now PublicHealthCenter(health\_center\_id, phone, name, address, postal\_code, #postal\_code\_id, type, website, drive\_thru)

This relation is **in 2NF**.

This relation is now **in 3NF**.

This relation is now **in BCNF**.

##### 4.2.2.2 Region

- Same process is used as PublicHealthCenter so we get:
- Province(province\_code, province)
- City(city\_id, city, #region\_id)
- Region(region\_id, region\_name, #province\_code)

This relation is **in 2NF**.

This relation is now **in 3NF**.

This relation is now **in BCNF**.

### 4.2.3 BCNF

**Boyce-Codd Normal Form BCNF:** achieved by first making sure the table is in third normal form. So the steps are :

- The Relation is in **3NF**
- For every non-trivial dependency  $X \rightarrow A$ ,  $A$  is a superkey.

We have already showed that Person, PublicHealthWorker, PublicHealthCenter, Region, Diagnostic, FollowUpForm, GroupZone and Alert are all in **BCNF**.

### 4.3 Final Normalized Model

**This is our final normalized model of the system. All**

- Person (person\_id, medicare, first\_name, last\_name, address, postal\_code, #postal\_code\_id, citizenship, email, phone, dob, password, api\_token)
- Patient (patient\_id, # person\_id)
- Carer (#child\_id, #parent\_id)
- Administrator(admin\_id, #person\_id)
- PublicHealthWorker (health\_worker\_id, #position\_id, schedule,#health\_center\_id, #person\_id)
- PublicHealthCenter (health\_center\_id, phone, name, address, postal\_code,#postal\_code\_id, type, website, method, drive\_thru)
- Position (position\_id, position)
- Diagnostic (diagnostic\_id, diagnostic\_date, result,#health\_worker\_id, #health\_center\_id, #patient\_id)
- FollowUpForm (form\_id, #filled\_by, #patient\_id, created\_at)
- Symptom (symptom\_id, symptom)
- FollowUpFormSymptomPivot(#symptom\_id, form\_id)
- GroupZone (group\_id, activity, name)
- GroupZonePersonPivot(#group\_id, #person\_id)
- Region (region\_id, region\_name, #alert\_id, #province\_code)
- Alert (alert\_id, alert\_info)
- Province(province\_code, province)
- City(city\_id, city, #region\_id)
- PostalCode (postal\_code\_id, #city\_id)
- Recommendation (recommendation\_id, recommendation)
- Messages (msg\_id,#region\_id, msg\_date, #alert\_id, #person\_id)

## 4.4 Analysis of 3NF and BCNF

The database design has been normalised to comply with the normalisation rules up to Third Normal Form (3NF). There are no redundant data that are related, as we split them all into their own tables. These tables are each identified by their own ID (primary key). If some data is related to more than one record, there has been a separate table created for it. Such tables are connected through foreign keys or, in some **special cases**, a separate pivot table has been created. Additionally, there are no values in the tables that do not depend on the key and we did not need any composite keys.

Below is a list of all the **special cases** that required pivot tables.

- **FollowupFormSymptomPivot**: Used to map a specific record of followup form to multiple records of symptom.
- **GroupZonePersonPivot**: Used to map a specific record of person form to multiple records of person.

We have shown in earlier sections that every relation has its own single super key and candidate key. There are no attributes that depend on non-key attributes in our tables. All dependent non-key attributes, do depend on the table's primary key. In other words, for every non-trivial functional dependency listed there is no dependency of the form  $A \rightarrow B, B \rightarrow C, C \rightarrow A$ , and for every functional dependency  $X \rightarrow Y$ , X will be the superkey of the table. As we have proved that our schema is in BCNF form, all data redundancy based on functional dependencies have been removed.

## 5 Functionalities Implemented to satisfy requirements

For each of the required entities we have implemented PHP functions to Create, Read, Update and Delete a resource. These functions are placed in our Ressource controllers which handle the requests coming from the front end to our API. To make our live's simpler we have implemented the following helper functions accessible by all the rest of our controllers.

```
<?php
public function doInsertAndGetId($tableName, Collection $parameters){
    $keys = $parameters->keys()->join(',');
    $placeholders = $parameters->map(function () { return '?';})->join(',');
    $result = DB::insert("INSERT INTO $tableName (
        $keys
    ) VALUES ($placeholders)",
    [
        ...$parameters->values(),
    ]
    );
    if ($result) {
        $id = DB::getPdo()->lastInsertId();
        return $id;
    }
    return null;
}
```

```
<?php
public function doUpdate($tableName, $key, $id, Collection $fieldsToUpdate){
    $values = $fieldsToUpdate->values();
    $values->push($id);
    return DB::update("UPDATE $tableName SET {$fieldsToUpdate->keys()->join(',')}"
        WHERE $key = ?", $values->toArray());
}
```

```
<?php
public function syncGroupIds($personId, $groupZones)
{
    $toDelete = collect();
    $toAdd = collect();

    if (!$personId) {
        abort(500);
    }

    // delete anything that isn't in the group_zones
    $dbGroupIds = collect(DB::select("SELECT gzp.group_id FROM
        GroupZonePersonPivot gzp
        WHERE person_id = '{$personId}'"))->pluck('group_id')->toArray();
    foreach ($dbGroupIds as $dbGroupId) {
        if (!in_array($dbGroupId, $groupZones)) {
            $toDelete->push($dbGroupId);
        }
    }
}
```

```

    }
}
foreach ($groupZones as $groupZone) {
    if (!in_array($groupZone, $dbGroupIds)) {
        $toAdd->push($groupZone);
    }
}
if ($toAdd->isNotEmpty()) {
    $stringAdd = $toAdd->map(function ($groupId) use (&$personId) {
        return "($personId,$groupId)";
    })->join(',');
    DB::insert("INSERT INTO GroupZonePersonPivot (`person_id`, `group_id`)
        VALUES $stringAdd");
}
if ($toDelete->isNotEmpty()) {
    $toDelete->map(function ($id) use (&$personId) {
        DB::insert("DELETE FROM GroupZonePersonPivot WHERE
            person_id=$personId and group_id=$id");
    });
}
}
}

```

## 5.1 Patient CRUD

### 5.1.1 Create

```

<?php
public function create(Request $request){
    $parameters = collect($request->only([
        'medicare',
        'first_name',
        'last_name',
        'address',
        'postal_code',
        'postal_code_id',
        'citizenship',
        'email',
        'phone',
        'dob'
    ]));
    $parameters['medicare'] = str_replace(' ', '', $parameters['medicare']);
    $parameters->put('password', bcrypt(str_replace('-', '', $parameters['dob'])));
    $parameters->put('api_token', Str::random(60));
    $id = $this->doInsertAndGetId('Person', $parameters);
    $this->syncGroupIds($id, $request->group_zones);
    DB::insert("INSERT INTO Patient (person_id) VALUES (?)", [$id]);
    return response()->json(['patient_id' => $id], $id ? Response::HTTP_CREATED :
        Response::HTTP_BAD_REQUEST);
}

```

### 5.1.2 Read

```
<?php
public function readAll(Request $request){
    return response()->json(DB::select("SELECT p.patient_id, ps.*
        FROM Patient p
        JOIN Person ps ON p.person_id = ps.person_id"));
}

public function readOne($id){
    $result = DB::select("SELECT
        p.patient_id,
        ps.*,
        c.city,
        pv.province,
        r.region_name,
        GROUP_CONCAT(gzp.group_id) as 'group_zones'
    FROM Patient p
    JOIN Person ps ON p.person_id = ps.person_id
    LEFT JOIN GroupZonePersonPivot gzp ON gzp.person_id = p.person_id
    JOIN PostalCode pc ON ps.postal_code_id = pc.postal_code_id
    JOIN City c ON pc.city_id = c.city_id
    JOIN Region r ON c.region_id = r.region_id
    JOIN Province pv ON pv.province_code = r.province_code
    WHERE p.patient_id = '{$id}'
    GROUP BY p.patient_id");

    return response()->json((count($result) > 0 ? $result[0] : null),
        count($result) > 0 ? 200 : 404
    );
}
```

### 5.1.3 Update

```
<?php
public function update(Request $request, $id){
    $personFieldsToUpdate = collect();

    if ($request->filled('password')) {
        $personFieldsToUpdate->put('password = ?', $request->password);
    }
    if ($request->filled('first_name')) {
        $personFieldsToUpdate->put('first_name = ?', $request->first_name);
    }
    if ($request->filled('last_name')) {
        $personFieldsToUpdate->put('last_name = ?', $request->last_name);
    }
    if ($request->filled('address')) {
        $personFieldsToUpdate->put('address = ?', $request->address);
    }
}
```



```

        if ($request->filled('postal_code_id')) {
            $personFieldsToUpdate->put('postal_code_id = ?',
                $request->postal_code_id);
        }
        if ($request->filled('citizenship')) {
            $personFieldsToUpdate->put('email = ?', $request->email);
        }
        if ($request->filled('email')) {
            $personFieldsToUpdate->put('phone = ?', $request->phone);
        }
        if ($request->filled('phone')) {
            $personFieldsToUpdate->put('phone = ?', $request->phone);
        }
        if ($request->filled('dob')) {
            $personFieldsToUpdate->put('dob = ?', $request->dob);
        }
        $personId = DB::select("SELECT person_id FROM Patient WHERE patient_id =
            '{$id}'")[0]->person_id ?? null;
        if (!$personId) {
            abort(500);
        }
        $this->syncGroupIds($personId, $request->group_zones);
        $this->doUpdate('Person', 'person_id', $id, $personFieldsToUpdate);
        $fieldsUpdated = $personFieldsToUpdate->count();
        return response()->json(['message' => $fieldsUpdated . " field(s) updated
            successfully!"], 200);
    }
}

```

#### 5.1.4 Delete

```

<?php
public function delete($id){
    $status = DB::delete("DELETE FROM Patient WHERE patient_id = ?", [$id]);
    return response()->json(['status' => "Deleted successfully!"], 200);
}

```

## 5.2 Public Health Worker CRUD

### 5.2.1 Create

```

<?php
public function create(Request $request){
    $parameters = collect($request->only([
        'medicare',
        'password',
        'first_name',
        'last_name',
        'address',

```

```

        'postal_code_id',
        'citizenship',
        'email',
        'phone',
        'dob',
        'region_id'
    ]));
    $pid = $this->doInsertAndGetId('Person', $parameters);

    $this->syncGroupIds($pid, $request->group_zones);
    $schedule = $request->input('schedule');
    $position_id = $request->input('position_id');
    $center_id = $request->input('health_center_id');

    DB::insert("INSERT INTO PublicHealthWorker (person_id, position_id, schedule,
        health_center_id) VALUES (?, ?, ?, ?)", [$pid, $position_id, $schedule,
        $center_id]);
}

```

### 5.2.2 Read

```

<?php
public function readAll(Request $request){
    $stringSearch = "SELECT w.health_worker_id, pst.position, w.schedule, ps.*
        FROM PublicHealthWorker w
        JOIN Position pst ON w.position_id = pst.position_id
        JOIN Person ps ON w.person_id = ps.person_id
        JOIN PublicHealthCenter phc ON w.health_center_id =
        phc.health_center_id";

    if ($request->filled("health_center_id")) {
        $stringSearch .= " WHERE w.health_center_id = " .
            $request->health_center_id;
    }
    return response()->json(DB::select($stringSearch));
}

public function readOne($id){
    $result = DB::select("SELECT
        w.health_worker_id,
        pst.position,
        w.schedule,
        ps.*,
        c.city,
        p.province,
        r.region_name,
        GROUP_CONCAT(gzp.group_id) as 'group_zones'
    FROM PublicHealthWorker w
    JOIN Position pst ON w.position_id = pst.position_id
    JOIN Person ps ON w.person_id = ps.person_id
    LEFT JOIN GroupZonePersonPivot gzp ON gzp.person_id = ps.person_id
    JOIN PostalCode pc ON ps.postal_code_id = pc.postal_code_id

```

```

        JOIN City c ON pc.city_id = c.city_id
        JOIN Region r ON c.region_id = r.region_id
        JOIN Province p ON p.province_code = r.province_code
        WHERE w.health_worker_id = '{$id}'
        GROUP BY w.health_worker_id");

    return response()->json((count($result) > 0 ? $result[0] : null),
        count($result) > 0 ? 200 : 404
    );
}

```

### 5.2.3 Update

```

<?php
public function update(Request $request, $id){
    $personFieldsToUpdate = collect();
    $workerFieldsToUpdate = collect();
    if ($request->filled('password')) {
        $personFieldsToUpdate->put('password = ?', $request->password);
    }
    if ($request->filled('first_name')) {
        $personFieldsToUpdate->put('name = ?', $request->name);
    }
    if ($request->filled('last_name')) {
        $personFieldsToUpdate->put('last_name = ?', $request->name);
    }
    if ($request->filled('address')) {
        $personFieldsToUpdate->put('address = ?', $request->address);
    }
    if ($request->filled('postal_code_id')) {
        $personFieldsToUpdate->put('postal_code_id = ?',
            $request->postal_code_id);
    }
    if ($request->filled('citizenship')) {
        $personFieldsToUpdate->put('email = ?', $request->email);
    }
    if ($request->filled('email')) {
        $personFieldsToUpdate->put('phone = ?', $request->phone);
    }
    if ($request->filled('phone')) {
        $personFieldsToUpdate->put('phone = ?', $request->phone);
    }
    if ($request->filled('dob')) {
        $personFieldsToUpdate->put('dob = ?', $request->dob);
    }
    if ($request->filled('region_id')) {
        $personFieldsToUpdate->put('region_id = ?', $request->region_id);
    }
    if ($request->filled('position')) {
        $workerFieldsToUpdate->put('position = ?', $request->position);
    }
}

```

```

        if ($request->filled('schedule')) {
            $workerFieldsToUpdate->put('schedule = ?', $request->dob);
        }
        $personId = DB::select("SELECT person_id FROM PublicHealthWorker WHERE
            health_worker_id = '{$id}'")[0]->person_id ?? null;
        $this->syncGroupIds($personId, $request->group_zones);
        $this->doUpdate('Person', 'person_id', $id, $personFieldsToUpdate);
        $this->doUpdate('PublicHealthWorker', 'health_worker_id', $id,
            $workerFieldsToUpdate);
        $fieldsUpdated = $personFieldsToUpdate->count() +
            $workerFieldsToUpdate->count();
        return response()->json(['message' => $fieldsUpdated . " field(s) updated
            successfully!"], 200);
    }

```

#### 5.2.4 Delete

```

<?php
public function delete($id){
    $status = DB::delete("DELETE FROM PublicHealthWorker WHERE health_worker_id =
        ?", [$id]);
    return response()->json(['status' => "Deleted successfully!"], 200);
}

```

### 5.3 Facility CRUD

#### 5.3.1 Create

```

<?php
public function create(CreateFacilityRequest $request){
    $parameters = collect($request->only([
        'name',
        'phone',
        'address',
        'postal_code',
        'postal_code_id',
        'type',
        'website',
        'method',
        'drive_thru'
    ]));
    $id = $this->doInsertAndGetId('PublicHealthCenter', $parameters);
    return response()->json(['health_center_id' => $id], $id ?
        Response::HTTP_CREATED : Response::HTTP_BAD_REQUEST);
}

```

### 5.3.2 Read

```
<?php
public function readAll(Request $request){
    return response()->json(DB::select("SELECT `health_center_id`, `name`,
        `phone`, `address`, `type`,`method`,`drive_thru` FROM
        PublicHealthCenter"));
}
public function readAll(Request $request){
    return response()->json(DB::select("SELECT `health_center_id`, `name`, `phone`,
        `address`, `type`,`method`,`drive_thru` FROM PublicHealthCenter"));
}
```

### 5.3.3 Update

```
<?php
public function create(CreateFacilityRequest $request){
    $parameters = collect($request->only([
        'name',
        'phone',
        'address',
        'postal_code',
        'postal_code_id',
        'type',
        'website',
        'method',
        'drive_thru'
    ]));
    $id = $this->doInsertAndGetId('PublicHealthCenter', $parameters);
    return response()->json(['health_center_id' => $id], $id ?
        Response::HTTP_CREATED : Response::HTTP_BAD_REQUEST);
}
```

### 5.3.4 Delete

```
<?php
public function create(CreateFacilityRequest $request){
    $parameters = collect($request->only([
        'name',
        'phone',
        'address',
        'postal_code',
        'postal_code_id',
        'type',
        'website',
        'method',
        'drive_thru'
    ]));
    $id = $this->doInsertAndGetId('PublicHealthCenter', $parameters);
    return response()->json(['health_center_id' => $id], $id ?
        Response::HTTP_CREATED : Response::HTTP_BAD_REQUEST);
}
```

```

    ));
    $id = $this->doInsertAndGetId('PublicHealthCenter', $parameters);
    return response()->json(['health_center_id' => $id], $id ?
        Response::HTTP_CREATED : Response::HTTP_BAD_REQUEST);
}

```

## 5.4 Region CRUD

### 5.4.1 Create

Our Webapp already has all the regions we don't create or delete any and we have an autocomplete function that renders the region based on postal code.

```

<?php
public function autocomplete(Request $request){
    return response()->json(DB::select("
        SELECT pc.postal_code_id, c.city_id, c.city, r.region_id, r.region_name,
        p.province_code, p.province
        FROM PostalCode pc
        JOIN City c ON pc.city_id = c.city_id
        JOIN Region r ON c.region_id = r.region_id
        JOIN Province p ON p.province_code = r.province_code
        WHERE pc.`postal_code_id` like '{$request->input('postal_code')}%')");
}

```

### 5.4.2 Read

```

<?php
public function readAll(Request $request){
    return response()->json(DB::select("
        SELECT
            r.region_id,
            r.region_name,
            a.alert_id,
            a.alert_info,
            GROUP_CONCAT(c.city) as 'cities',
            GROUP_CONCAT(pc.postal_code_id) as 'postal_codes'
        FROM Region r
        JOIN Alert a ON r.alert_id = a.alert_id
        JOIN City c ON c.region_id = r.region_id
        JOIN PostalCode pc ON pc.city_id = c.city_id
        GROUP BY r.region_id"));
}

public function readOne($id)
{
    $result = DB::select("SELECT * FROM Region WHERE $id = region_id");

    return response()->json((count($result) > 0 ? $result[0] : null),

```

```

        count($result) > 0 ? 200 : 404);
    }

```

### 5.4.3 Update

```

<?php
public function update(Request $request, $id){
    $newAlertId = $request->input('alert_id');
    $currentAlertId = DB::select("SELECT alert_id FROM Region WHERE region_id =
        $id")[0]->alert_id ?? null;
    try {
        if (abs($newAlertId - $currentAlertId) > 1 && $newAlertId != 0 ||
            !$currentAlertId) {
            return response()->json(['message' => "wowow cant jump from more
                than 1 alert my guy!"], 400);
        } else {
            $this->doUpdate('Region', 'region_id', $id, collect(['alert_id = ?'
                => $newAlertId]));
            $fieldsUpdated = 1;
            return response()->json(['message' => $fieldsUpdated . " field(s)
                updated successfully!"], 200);
        }
    } catch (QueryException $e) {
        if ($e->getStatusCode() == 23000) {
            return response()->json(['message' => "alert does not exist!"],
                400);
        }
    }
}

```

## 5.5 Group-Zone CRUD

### 5.5.1 Create

```

<?php
public function create(Request $request){
    $parameters = collect($request->only([
        'name',
        'activity',
    ]));
    $id = $this->doInsertAndGetId('GroupZone', $parameters);

    return response()->json(['group_zone_id' => $id], $id ?
        Response::HTTP_CREATED : Response::HTTP_BAD_REQUEST);
}

```

### 5.5.2 Read

```
<?php
public function readAll(Request $request){
    return response()->json(DB::select("SELECT * FROM GroupZone"));
}

public function readOne($id){
    $result = DB::select("SELECT *
        FROM GroupZone WHERE group_id = '{$id}'");
    return response()->json((count($result) > 0 ? $result[0] : null),
        count($result) > 0 ? 200 : 404);
}
```

### 5.5.3 Update

```
<?php
public function update(Request $request, $id){
    $groupZoneFieldsToUpdate = collect();

    if ($request->filled('activity')) {
        $groupZoneFieldsToUpdate->put('activity = ?', $request->activity);
    }
    if ($request->filled('name')) {
        $groupZoneFieldsToUpdate->put('name = ?', $request->name);
    }
    $this->doUpdate('GroupZone', 'group_id', $id, $groupZoneFieldsToUpdate);
    $fieldsUpdated = $groupZoneFieldsToUpdate->count();
    return response()->json(['message' => $fieldsUpdated . " field(s) updated
        successfully!"], 200);
}
```

## 5.6 Recommendation CRUD

### 5.6.1 Create

```
<?php
public function create(Request $request){
    $recommendation = $request->input('recommendation');
    DB::insert("INSERT INTO Recommendation (recommendation) VALUES (?)",
        [$recommendation]);
}
```

### 5.6.2 Read



```

<?php
public function readAll(Request $request){
    return response()->json(DB::select("
        SELECT
            r.region_id,
            r.region_name,
            a.alert_id,
            a.alert_info,
            GROUP_CONCAT(c.city) as 'cities',
            GROUP_CONCAT(pc.postal_code_id) as 'postal_codes'
        FROM Region r
        JOIN Alert a ON r.alert_id = a.alert_id
        JOIN City c ON c.region_id = r.region_id
        JOIN PostalCode pc ON pc.city_id = c.city_id
        GROUP BY r.region_id"));
}

public function readOne($id){
    $result = DB::select("SELECT * FROM Region WHERE $id = region_id");

    return response()->json((count($result) > 0 ? $result[0] : null),
        count($result) > 0 ? 200 : 404);
}

```

### 5.6.3 Update

```

<?php
public function update(Request $request, $id){
    $field = $request->input('recommendation');
    DB::update("UPDATE recommendation SET recommendation = (?) WHERE
        recommendation_id = $id", [$field]);
}

```

```

<?php
public function update(Request $request, $id){
    $newAlertId = $request->input('alert_id');
    $currentAlertId = DB::select("SELECT alert_id FROM Region WHERE region_id =
        $id")[0]->alert_id ?? null;
    try {
        if (abs($newAlertId - $currentAlertId) > 1 && $newAlertId != 0 ||
            !$currentAlertId) {
            return response()->json(['message' => "wowow cant jump from more
                than 1 alert my guy!"], 400);
        } else {
            $this->doUpdate('Region', 'region_id', $id, collect(['alert_id = ?'
                => $newAlertId]));
            $fieldsUpdated = 1;
        }
    }
}

```

```

        return response()->json(['message' => $fieldsUpdated . " field(s)
            updated successfully!"], 200);
    }
} catch (QueryException $e) {
    if ($e->getCode() == 23000) {
        return response()->json(['message' => "alert does not exist!"],
            400);
    }
}
}
}

```

## 5.7 Trigger for Alerts

Once a new alert is set for a region, a trigger must send message(s) to all the people that are registered in that region informing them of the new alert state, and also inform them whether the alert is more strict or less strict than the previous alert

### 5.7.1 MySQL

```

CREATE TRIGGER alertTrigger
AFTER UPDATE ON Region
FOR EACH ROW
BEGIN
    if (NEW.alert_id = 4) THEN
    INSERT INTO messages(`message`, `region_id`, `msg_date`, `person_id`, `alert_id`)
    SELECT
        CONCAT('Bonsoir ',ps.first_name, ', email: ', ps.email, ' this is an automated
            message generated at ',NOW(), ' to warn you that the alert level in your region:
            ', r.region_name, ' went from ', a.alert_info, ' to ', b.alert_info,
            '.This is RED alert, the maximum alert a region can have. Please follow the link
            and read about guidelines, visit www.coviddb.com/guidelines.' ) AS 'message',
        OLD.region_id AS 'region_id',
        NOW() AS 'msg_date',
        ps.person_id AS 'person_id',
        NEW.alert_id AS 'alert_id'
    FROM
        Person ps
        JOIN PostalCode pc ON pc.postal_code_id = ps.postal_code_id
        JOIN City c ON c.city_id = pc.city_id
        JOIN Region r ON r.region_id = OLD.region_id
        JOIN Alert a ON a.alert_id = OLD.alert_id
        JOIN Alert b ON b.alert_id = NEW.alert_id;
    END IF;
    if (NEW.alert_id < 4) THEN
    INSERT INTO messages(`message`, `region_id`, `msg_date`, `person_id`, `alert_id`)
    SELECT
        CONCAT('Bonsoir ',ps.first_name, ', email: ', ps.email, ' this is an automated
            message generated at ',NOW(), ' to warn you that the alert level in your region:
            ', r.region_name, ' went from ', a.alert_info, ' to ', b.alert_info) AS
            'message',

```

```

    OLD.region_id AS 'region_id',
    NOW() AS 'msg_date',
    ps.person_id AS 'person_id',
    NEW.alert_id AS 'alert_id'
FROM
    Person ps
    JOIN PostalCode pc ON pc.postal_code_id = ps.postal_code_id
    JOIN City c ON c.city_id = pc.city_id
    JOIN Region r ON r.region_id = OLD.region_id
    JOIN Alert a ON a.alert_id = OLD.alert_id
    JOIN Alert b ON b.alert_id = NEW.alert_id;
END IF;
END;

```

## 5.8 Trigger for Diagnostic Result

Upon a person's test result is determined, a trigger must send a message to the person notifying him/her of the test result. If the test is positive, an additional message is sent to the person with information about the public health recommendations, and a reminder message is sent to the person to fill out the follow-up-form using the username (medicare card number) and the password (DOB of the form ddmmyyyy).

### 5.8.1 MySQL

```

CREATE TRIGGER diagTrigger
AFTER INSERT ON Diagnostic
FOR EACH ROW
BEGIN
    IF (NEW.result = 1) THEN
        INSERT INTO messages(`message`, `region_id`, `msg_date`, `person_id`, `alert_id`)
        SELECT
            CONCAT('Bonsoir ', ps.first_name, '. You have received a POSITIVE result for your
                COVID diagnostic.'),
            null AS 'region_id',
            NEW.diagnostic_date AS 'msg_date',
            ps.person_id AS 'person_id',
            null AS 'alert_id'
        FROM
            Person ps
            JOIN patient pc ON pc.person_id = ps.person_id AND NEW.patient_id = pc.patient_id;
        INSERT INTO messages(`message`, `region_id`, `msg_date`, `person_id`, `alert_id`)
        SELECT
            CONCAT('Bonsoir ', ps.first_name, '. You have recently received a POSITIVE
                diagnostic for COVID-19. Here are the public health recommendations that should
                be followed during
                14 consecutive day after your diagnostic www.coviddb.com/recommendations. Reminder:
                Do not forget to fill up the follow-up-form on www.coviddb.com using your
                medicare <', ps.medicare, '> as username and your date of birth <', ps.dob, '> as
                password.'),
            null AS 'region_id',
            NEW.diagnostic_date AS 'msg_date',

```

```

    ps.person_id AS 'person_id',
    null AS 'alert_id'
FROM
    Person ps
    JOIN patient pc ON pc.person_id = ps.person_id AND NEW.patient_id = pc.patient_id;
END IF;
IF (NEW.result = 0) THEN
INSERT INTO messages(`message`, `region_id`, `msg_date`, `person_id`, `alert_id`)
SELECT
    CONCAT('Bonsoir ', ps.first_name, '. You have received a NEGATIVE result for your
        COVID diagnostic.'),
    null AS 'region_id',
    NEW.diagnostic_date AS 'msg_date',
    ps.person_id AS 'person_id',
    null AS 'alert_id'
FROM
    Person ps
    JOIN patient pc ON pc.person_id = ps.person_id AND NEW.patient_id = pc.patient_id;
END IF;
END;

```

## 6 Scripts & Queries

### 6.1 Form Followup

Fill-Up a follow up form for a person who is tested positive, for fourteen consecutive days following the test results. The form must include the date, the time, the temperature, the status of all the main symptoms and other symptoms as specified for the COVID-19 as well as other non-specified symptom(s) if exists. Every person can fill up his/her form using the username and password. The username is the medicare card number of the person and the password is the date of birth of the person of the form (ddmmyyyy).

#### 6.1.1 MySQL

```

INSERT INTO FollowUpForm(`filled_by`, `patient_id`, `form_id`, `created_at`,
    `temperature`, `other_symptoms`)
VALUES (5,2,1,'2021-04-11 02:26:25', `36`,`strong_appetite`);
INSERT INTO FollowUpFormSymptomPivot (`form_id`, `symptom_id`)
SELECT '1', symptom_id
FROM Symptoms
WHERE symptom IN ('fever','loss_of_appetite');

```

#### 6.1.2 PHP

The implementation in the back-end is written as:

```

<?php
# Q9

```

```

public function create(Request $request){
    $parameters = collect($request->only([
        'filled_by',
        'patient_id',
        'form_id',
        'created_at',
    ]));
    DB::insert("INSERT INTO FollowUpForm (filled_by, patient_id, form_id,
        created_at) VALUES (?, ?, ?, ?)", [$parameters]);
}

```

### 6.1.3 Representative tuples

filled_by	patient_id	form_id	created_at
1	1	17	2021-04-16 10:25:27
1	1	25	2021-04-17 10:30:00
1	1	39	2021-04-18 21:07:18
5	1	42	2021-04-19 10:25:27
1	1	58	2021-04-20 04:00:00

Table 17: Follow-up form representative tuples

### 6.1.4 Screenshot in WebApp

The screenshot shows a web application interface for COVID-DB. On the left is a sidebar menu with options: Admin, Patients, Workers, Facilities, Regions, GroupZones, Recommendations, Symptoms, User, and Diagnosis & Follow-Up. The main content area is titled 'COVID-DB' and contains a 'FOLLOW-UP DATE' form. The form has a date input field showing '17-04-2021' with a calendar dropdown open, displaying the month of April 2021 with the 17th selected. Below the date field are five checkboxes for symptoms: 'vomiting' (checked), 'headache' (unchecked), 'muscle\_pain' (unchecked), 'diarrhea' (checked), and 'sore\_throat' (unchecked). At the bottom of the form is a blue 'Submit Form' button.

Figure 3: Filling up a form

## 6.2 Symptoms Progress

Provide a detail progress of all the symptoms for a given person who tested positive on a specific date.

### 6.2.1 MySQL

```
SELECT (`form_id`,`patient_id`,`created_at`,`symptom`)
FROM followupformsymptompivot
NATURAL JOIN followupform
NATURAL JOIN symptom
WHERE patient_id = patient_id AND created_at > start_date
```

### 6.2.2 PHP

The implementation in the back-end is written as:

```
<?php
# Q9
public function readAll(Request $request){
return response()->json(DB::select("SELECT
    (`form_id`,`patient_id`,`created_at`,`symptom`)
FROM followupformsymptompivot
NATURAL JOIN followupform
NATURAL JOIN symptom
WHERE patient_id = $request->patient_id AND created_at = $request->start_date"));
}
```

### 6.2.3 Representative tuples

An example GET request would be :

127.0.0.1:8000/api/form?patient\_id=1&start\_date='2021-04-17'

Which would return something like :

form_id	patient_id	created_at	symptom
4	1	2021-04-17 10:25:27	headache
5	1	2021-04-18 10:25:27	throat ache
6	1	2021-04-18 21:07:18	cough
7	1	2021-04-18 10:30:00	cough
8	1	2021-04-19 11:00:00	headache

Table 18: Symptom progress representative tuples

## 6.3 Messages during a given time

Display all messages generated by the system within a specific period of time.

### 6.3.1 MySQL

```
SELECT * FROM Messages
WHERE msg_date > start_date
AND msg_date < end_date
```

### 6.3.2 PHP

The implementation in the back-end is written as:

```
<?php
# Q10
public function readAll(Request $request){
    return response()->json(DB::select("SELECT * FROM messages WHERE msg_date >
    $request->start_date AND msg_date < $request->end_date"));
}
```

### 6.3.3 Representative tuples

An example GET request would be :

127.0.0.1:8000/api/messages?start\_date='2021-04-13 23:59:59'&end\_date=2021-04-17 23:59:59'

Which would return something like :

msg_id	region_id	msg_date	alert_id	message	person_id
45	NULL	2021-04-14 15:56:50	NULL	Bonsoir Eliott You have recieved a POSITIVE result for you COVID diagnostic. Please click here to view the public health recommendations that should be followed & click here to fill your followup form for the 14 next days	6
46	NULL	2021-04-15 15:56:50	NULL	Bonsoir Angela. You have recieved a Negative result for you COVID diagnostic.	1
47	21	2021-04-15 19:06:50	4	Bonsoir Tyrell. This is an automated message generated at 19:06:50 to warn you that the alert level in you region 5 went from 3 to 4. Click here to read about the new guidelines	8
48	NULL	2021-04-15 21:48:23	NULL	Bonsoir Tyrell. You have recieved a NEGATIVE result for your COVID diagnostic	8
49	5	2021-04-17 18:56:50	2	Bonsoir Eliott. This is an automated message generated at 18:56:50 to warn you that the alert level in you region 5 went from 1 to 2	6

Table 19: Messages generated in a given time representative tuples

### 6.3.4 Screenshot in WebApp

COVID-DB Logout

List of All Messages

MESSAGE	MSG_DATE
Bonsoir Marc, (e-mail: brigitte92@lambert.com) This is an automated message generated at 2021-04-17 10:13:09 to warn you that the alert level in your region, Montreal, went from Level 3: Orange to Level 2: Yellow. Stay Safe!	2021-04-17 10:13:09
Bonsoir Patricia, (e-mail: zacharie56@gmail.com) This is an automated message generated at 2021-04-17 10:13:09 to warn you that the alert level in your region, Montreal, went from Level 3: Orange to Level 2: Yellow. Stay Safe!	2021-04-17 10:13:09
Bonsoir Marc, (e-mail: brigitte92@lambert.com) This is an automated message generated at 2021-04-17 10:18:12 to warn you that the alert level in your region, Montreal, went from Level 2: Yellow to Level 3: Orange. Stay Safe!	2021-04-17 10:18:12
Bonsoir Benjamin, (e-mail: zacharie56@gmail.com) This is an automated message generated at 2021-04-17 10:18:12 to warn you that the alert level in your region, Montreal, went from Level 2: Yellow to Level 3: Orange. Stay Safe!	2021-04-17 10:18:12

Figure 4: Display all messages generate during in a given time

## 6.4 People in an address

Give a list of all the People in a specific address. For every person, provide the first name, last name, DOB, medicare card number, telephone number, citizenship, email address, full mother's name and full father's name.

### 6.4.1 MySQL

```
SELECT ps.first_name, ps.last_name, ps.dob, ps.medicare, ps.phone, ps.citizenship,
       ps.email,
       group_concat(ps2.first_name , ' ', ps2.last_name) as parent_full_name FROM person ps
JOIN carer c ON c.child_id = ps.person_id
JOIN person ps2 ON c.parent_id = ps2.person_id
WHERE ps.address = '3239 Isaac Lake Apt. 198'
GROUP BY medicare;
```

### 6.4.2 PHP

```
<?php
public function readAll(Request $request)
{
    $criteria = collect();
    if ($request->filled('address')) {
        $criteria->put("address = {$request->address}");
    }

    $stringQuery = $criteria->count() > 0 ? 'WHERE ' . $criteria->join(',') ;
    return response()->json(DB::select("SELECT p.patient_id, ps.*
```



```

FROM Patient p
JOIN Person ps ON p.person_id = ps.person_id $stringQuery"));
}

```

### 6.4.3 Representative tuples

Which would return something like :

first	last	dob	medicare	phone	citizenship	parent full
Christine	Tessier	1988-08-15	YNPI68246266	18135950684	US	Nathalie Roy, Jules Langlois
Julie	Titty	1969-04-20	ADFY12346214	13953393929	CA	Marcel Bo, Dom Nadeau
Edith	Tardif	1990-06-29	GCPI16679278	(730) 404-1566	MX	Guillaume

Table 20: People living in the same address representative tuples.

### 6.4.4 Screenshot in WebApp

The screenshot shows a web application interface for COVID-DB. At the top, there's a 'Logout' link. Below it, the title 'List of All Patients' is displayed. A search filter is labeled 'FILTER BY ADDRESS:' with a text input containing 'chantal' and a 'Search' button. To the right of the search bar is a 'Create a new Patient' button. Below the search bar, a table displays the results. The table has columns: FIRST\_NAME, LAST\_NAME, ADDRESS, MEDICARE, and DOB. One row is visible: Patrick, Fillion, 538 Chantal Field, UEDV28796760, and 1995-05-09.

Figure 5: Patients in an address

## 6.5 Details of all facilities

Give a detail list of all facilities (Address, Number of health workers, accept walk-in clients or by appointment only, Drive thru capability, etc.).

### 6.5.1 MySQL

```

SELECT
  phc.`health_center_id`,
  phc.`name`,
  phc.`phone`,
  phc.`address`,
  phc.`type`,
  phc.`method`,
  COUNT(w.health_worker_id) as 'worker_amount',
  if (phc.`drive_thru`, 'Yes', 'No')
FROM PublicHealthCenter phc
JOIN PublicHealthWorker w ON w.health_center_id = phc.health_center_id
GROUP BY phc.health_center_id

```

### 6.5.2 PHP

```
<?php
public function readAll(Request $request)
{
    return response()->json(DB::select("
        SELECT
            phc.`health_center_id`,
            phc.`name`,
            phc.`phone`,
            phc.`address`,
            phc.`type`,
            phc.`method`,
            COUNT(w.health_worker_id) as 'worker_amount',
            if (phc.`drive_thru`, 'Yes','No') as 'drive_thru'
        FROM PublicHealthCenter phc
        JOIN PublicHealthWorker w ON w.health_center_id = phc.health_center_id
        GROUP BY phc.health_center_id"));
}
```

### 6.5.3 Screenshot in webapp

COVID-DB Logout

List of All Facilities Create a new Facility

NAME	TYPE	WORKER_AMOUNT	ADDRESS	PHONE	METHOD	DRIVE_THRU
Girard, Jacques and Desrosiers	SPECIAL INSTALLMENT	1	3235 Olivier Alley	+1-606-508-0641	walk-in	Yes
Gaudreault, Desjardins and Gingras	SPECIAL INSTALLMENT	4	1754 Vaillancourt Squares Suite 930	625.237.8755	both	No
Guay Ltd	CLINIC	1	448 Girard Lane Suite 134	+1.903.423.4085	walk-in	Yes
Tessier Group	SPECIAL INSTALLMENT	1	54371 Maryse Crossroad Suite 460	+1-690-805-9089	walk-in	Yes

Figure 6: Displaying details of all facilities

## 6.6 Details of all Regions

Give a detail list of all Regions (Name, list of all cities within the region, and a list of all the postal codes within every city).

### 6.6.1 MySQL

```
SELECT
    r.region_id,
    r.region_name,
    a.alert_id,
    a.alert_info,
    count(p.person_id) as 'people_amount',
```

```

        GROUP_CONCAT(c.city) as 'cities',
        GROUP_CONCAT(pc.postal_code_id) as 'postal_codes'
FROM Region r
JOIN Alert a ON r.alert_id = a.alert_id
JOIN City c ON c.region_id = r.region_id
JOIN PostalCode pc ON pc.city_id = c.city_id
LEFT JOIN Person p ON p.postal_code_id = pc.postal_code_id
GROUP BY r.region_id ORDER BY count(p.person_id) DESC

```

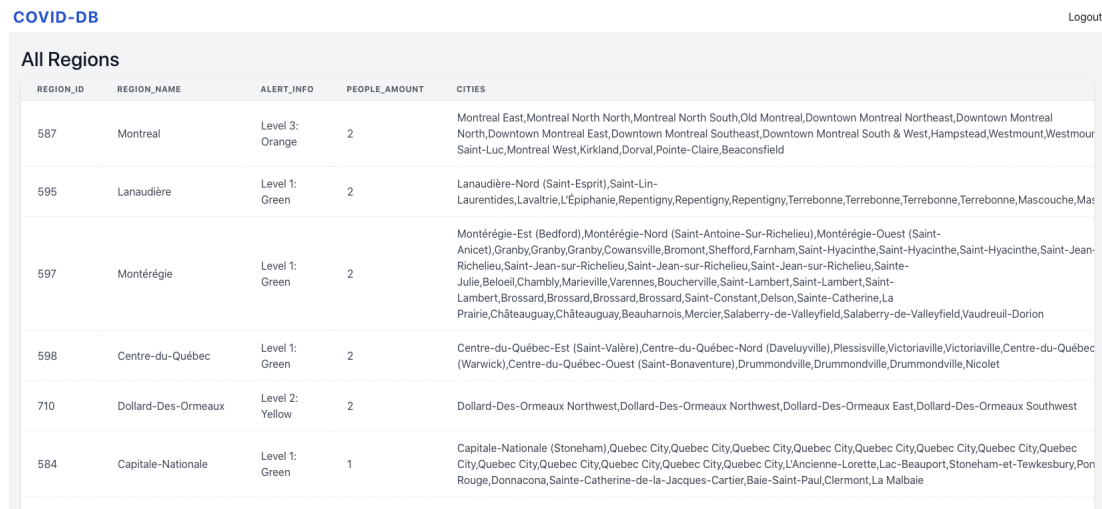
## 6.6.2 PHP

```

<?php
public function readAll(Request $request)
{
    return response()->json(DB::select("
        SELECT
            r.region_id,
            r.region_name,
            a.alert_id,
            a.alert_info,
            count(p.person_id) as 'people_amount',
            GROUP_CONCAT(c.city) as 'cities',
            GROUP_CONCAT(pc.postal_code_id) as 'postal_codes'
        FROM Region r
        JOIN Alert a ON r.alert_id = a.alert_id
        JOIN City c ON c.region_id = r.region_id
        JOIN PostalCode pc ON pc.city_id = c.city_id
        LEFT JOIN Person p ON p.postal_code_id = pc.postal_code_id
        GROUP BY r.region_id ORDER BY count(p.person_id) DESC"));
}

```

### 6.6.3 Screenshot in Webapp



REGION_ID	REGION_NAME	ALERT_INFO	PEOPLE_AMOUNT	CITIES
587	Montreal	Level 3: Orange	2	Montreal East, Montreal North North, Montreal North South, Old Montreal, Downtown Montreal Northeast, Downtown Montreal North, Downtown Montreal East, Downtown Montreal Southeast, Downtown Montreal South & West, Hampstead, Westmount, Westmount Saint-Luc, Montreal West, Kirkland, Dorval, Pointe-Claire, Beaconsfield
595	Lanaudière	Level 1: Green	2	Lanaudière-Nord (Saint-Esprit), Saint-Lin-Laurentides, Lavalltrie, L'Épiphanie, Repentigny, Repentigny, Repentigny, Terrebonne, Terrebonne, Terrebonne, Terrebonne, Mascouche, Mat
597	Montréal	Level 1: Green	2	Montréal-Est (Bedford), Montréal-Nord (Saint-Antoine-Sur-Richelieu), Montréal-Ouest (Saint-Anicet), Granby, Granby, Granby, Cowansville, Bromont, Shefford, Farnham, Saint-Hyacinthe, Saint-Hyacinthe, Saint-Hyacinthe, Saint-Jean-Richelieu, Saint-Jean-sur-Richelieu, Saint-Jean-sur-Richelieu, Saint-Jean-sur-Richelieu, Sainte-Julie, Beloeil, Chambly, Marieville, Varennes, Boucherville, Saint-Lambert, Saint-Lambert, Saint-Lambert, Brossard, Brossard, Brossard, Brossard, Saint-Constant, Delson, Sainte-Catherine, La Prairie, Châteauguay, Châteauguay, Beauharnois, Mercier, Salaberry-de-Valleyfield, Salaberry-de-Valleyfield, Vaudreuil-Dorion
598	Centre-du-Québec	Level 1: Green	2	Centre-du-Québec-Est (Saint-Valère), Centre-du-Québec-Nord (Daveluyville), Plessisville, Victoriaville, Victoriaville, Centre-du-Québec (Warwick), Centre-du-Québec-Ouest (Saint-Bonaventure), Drummondville, Drummondville, Drummondville, Nicolet
710	Dollard-Des-Ormeaux	Level 2: Yellow	2	Dollard-Des-Ormeaux Northwest, Dollard-Des-Ormeaux Northwest, Dollard-Des-Ormeaux East, Dollard-Des-Ormeaux Southwest
584	Capitale-Nationale	Level 1: Green	1	Capitale-Nationale (Stoneham), Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, Quebec City, L'Ancienne-Lorette, Lac-Beauport, Stoneham-et-Tewkesbury, Pont Rouge, Donnacona, Sainte-Catherine-de-la-Jacques-Cartier, Baie-Saint-Paul, Clermont, La Malbaie

Figure 7: Displaying details of all regions

## 6.7 Details of people's test result in a date ordered by result

Give a List of all people who got the result of the test on a specific date (Date of the result of the test) grouped by the test result value (First who got positive result, then who got negative result). The list must include First Name, Last Name, DOB, Phone Number and email address of the diagnosed person.

### 6.7.1 MySQL

```
SELECT ps.first_name, ps.last_name, ps.dob, ps.phone, ps.email, diagnostic_date,
       result
FROM diagnostic
NATURAL JOIN patient pt
NATURAL JOIN person ps
WHERE diagnostic_date > '2020-07-01 02:26:24'
ORDER BY result DESC;
```

## 6.8 List of all workers in a Facility

Give a list of all workers in a specific facility.

### 6.8.1 MySQL

```
SELECT ps.first_name, ps.last_name, health_worker_id FROM publichealthworker
NATURAL JOIN Person ps
WHERE health_center_id = '3'
```

## 6.9 Workers with COVID at a facility on date and their reach

Give a list of all public health workers who tested positive on a specific date in a specific facility. In addition, provide a list of the employees who worked with the infected health workers for the period of fourteen days prior to the test. Use the date of the test taken by the infected health worker.

### 6.9.1 MySQL

```
#6.9(1)
#ALL WORKERS WITH COVID ON THIS DATE AT THIS FACILITY
SELECT * FROM diagnostic
NATURAL JOIN patient pt
NATURAL JOIN person ps
JOIN publichealthworker pw ON pw.person_id = ps.person_id
WHERE result = '1' AND diagnostic_date > 0 AND pw.health_center_id = 2;

#6.9(2)
#RETURN WORKER THAT WORKS AT SAME FACILITY
SELECT * FROM publichealthworker
WHERE health_center_id = 2
```

## 6.10 Report for every region

For all regions, provide a report that include the region name, the number of people who have positive results, the number of people who have negative results and a history of alerts within a specific period-of-time.

### 6.10.1 MySQL

```
SELECT
  r.region_id,
  r.region_name,
  a.alert_id,
  a.alert_info,
  count(p.person_id) as 'people_amount',
  sum(if(aux.came_positive, 1, 0)) as amount_positive,
  sum(if(aux.came_positive, 0, 1)) as amount_negative,
  GROUP_CONCAT(c.city) as 'cities',
  GROUP_CONCAT(pc.postal_code_id) as 'postal_codes'
FROM Region r
JOIN Alert a ON r.alert_id = a.alert_id
JOIN City c ON c.region_id = r.region_id
JOIN PostalCode pc ON pc.city_id = c.city_id
LEFT JOIN Person p ON p.postal_code_id = pc.postal_code_id
LEFT JOIN (
  SELECT
    pt.person_id as 'person_id',
    MAX(d.diagnostic_date) 'last_diagnostic_date',
    MAX(d.result) as 'came_positive'
```

```
FROM Patient pt
  JOIN Diagnostic d ON pt.patient_id = d.patient_id
  GROUP BY pt.person_id
  ORDER BY MAX(d.diagnostic_date)
) as aux ON aux.person_id = p.person_id
GROUP BY r.region_id ORDER BY count(p.person_id) DESC;
```

## 7 User Interface

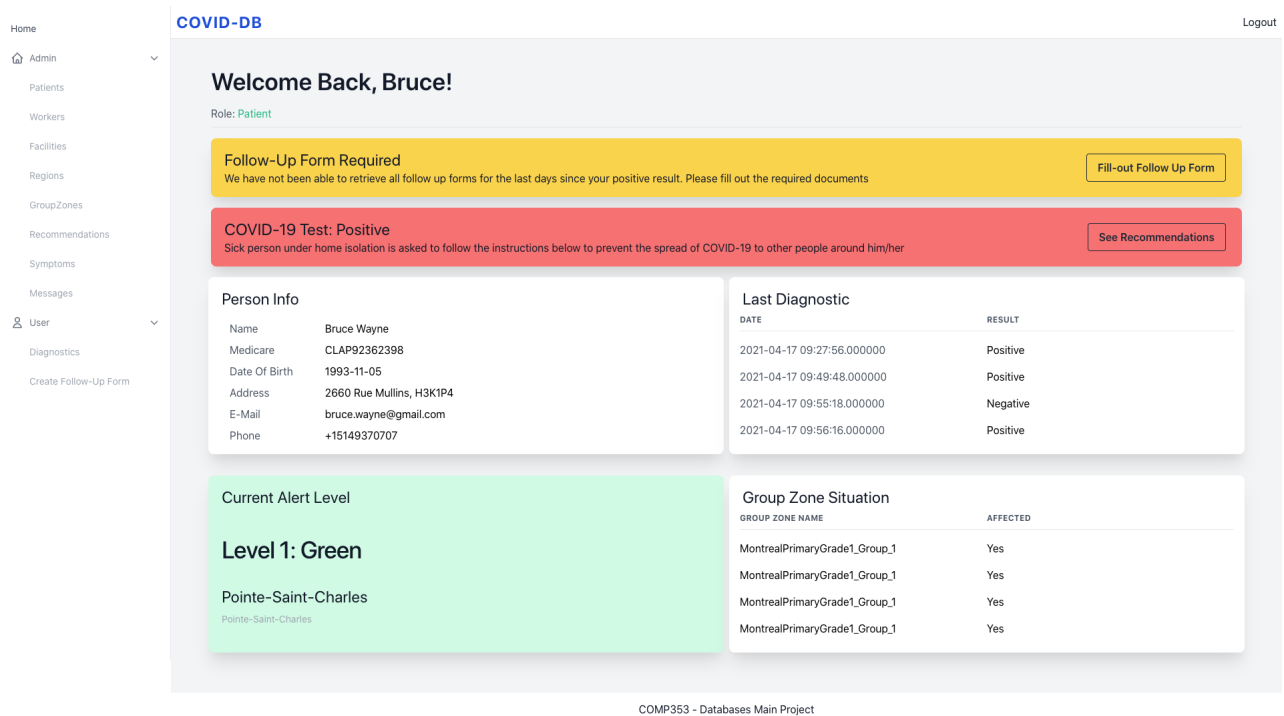


Figure 8: Landing Page of the User

### 7.1 Linked Resources

**GitHub URL:** <https://github.com/wdlnph/coviddb>

**Live Site URL:** <https://ydc353.encs.concordia.ca/coviddb/portal>

### 7.2 Backend Component: Laravel

The COVID-DB Application was built by our team using the Laravel PHP Framework, built by Taylor Otwell in June 2011. Not only does it help with proper session handling, password encryption and security layers for our service, but it also provides an simple organized MVC directory layout where we can properly separate our back-end component from the front end one. This allowed for the team to be highly effective during implementation, as the Controllers would be able to be implemented concurrently to the UI Layouts to be built

To accommodate with the requirements provided to us, some compromises were made to omit the use of some features in Laravel.

#### Fluent Language

Rather than using the fluent language, we have opted to write direct SQL into our service, allowing for the marker to properly verify that the queries have been implemented

#### Migrations

A new file create\_tables.sql was created in /database/sql folder to provide all the Relations that were created throughout this project, rather than the turnkey migration solution commonly performed in Laravel.

#### Seeding

Factory files were also omitted. Instead, we have provided the files `insert_statements.sql`, `insert_regions.sql` and `triggers.sql` to ensure the database is populated in the way it should match results found in this report.

### 7.3 Frontend Component: ReactJS

For a highly intuitive interface, we have opted to make use of the ReactJS framework for our front end. React allows us to easily re-use components throughout multiple entities, and provide a simple way to validate forms and communicate with the backend component. The axios package was used to communicate with the API, and the Formik plugin is used for validation with Yup.

In addition, we have decided to use Tailwind CSS to design our components, all of which is compiled using the Webpack wrapper provided by Laravel Mix.

### 7.4 Authentication

The user is able to log in using his medicare card and password, which is its date of birth, in YYYYMMDD format. A register functionality would also be provided, as well as a forgot password and remember me checkbox. These, however, do not enter into the scope of this project.

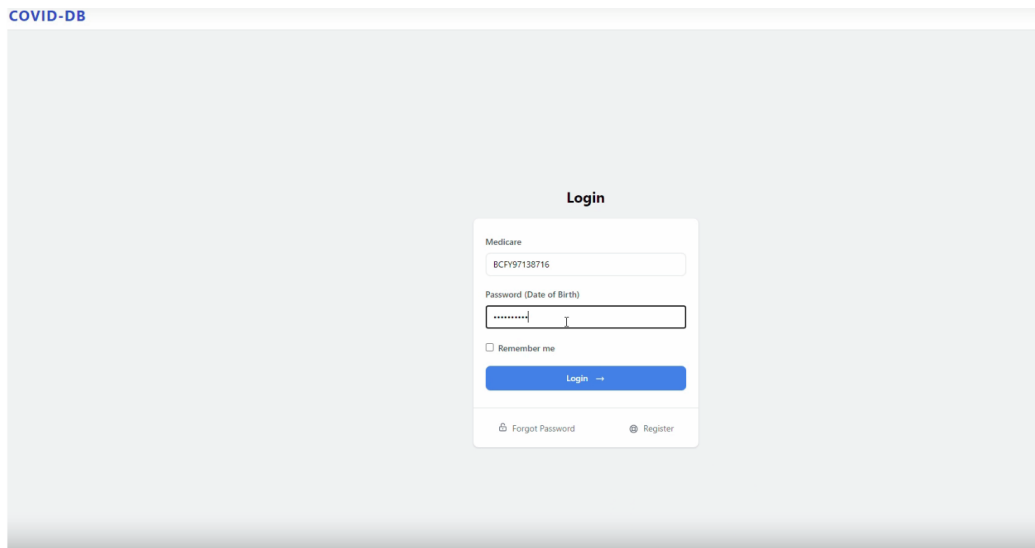


Figure 9: Home Page allows you to login with your medicare and dob

### 7.5 CRUD Table

A table was designed to display information coming from any 'readAll' functions. These rows can be clicked to further go into the Edit Entity page in order to update the selected row. Depending on the page, some filters may be provided at the top of the page.



**COVID-DB** Logout

**List of All Patients** Create a new Patient

FILTER BY ADDRESS: Search

FIRST_NAME	LAST_NAME	ADDRESS	MEDICARE	DOB
Patrick	Fillion	538 Chantal Field	UEDV28796760	1995-05-09
Antoine	Michaud	4530 Camille Fork	KGZF49486401	1991-04-16
Olivia	Ouellet	9629 Mathilde Overpass Suite 843	DQPP59430483	2002-12-02
Zoé	Rousseau	46052 Gilbert View Suite 888	YBKM00781930	2014-07-24
Michelle	Bergeron	523 Rolande Locks Apt. 463	WEYC17226416	2010-10-10
Marc	Gauthier	756 Tessier Prairie Apt. 347	EEKJ96363064	1994-09-29
Guillaume	Savard	207 Théophile Skyway Apt. 187	EQMK62883826	1978-04-02
Élise	Gravel	255 Christophe Bypass	FTMH74095551	1974-09-19
Tristan	Leduc	8050 Daniel Causeway	GCPI16679278	1977-05-08
Édith	Tardif	1037 Gauthier Gateway Apt. 688	RQVV09187089	1990-06-29
Bruce	Wayne	2660 Rue Mullins	CLAP92362396	1993-11-05

COMP353 - Databases Main Project

Figure 10: List of all workers in a table

## 7.6 Create/Update Entity

Each entity provides its own schema that can be validated using the combination of the Yup and Formik plugins. Validation prevents for erroneous data from reaching the database, as it may not result into the right constraints being respected. This only serves as a first layer protection, since the Database and the API itself provide their own validation rules as well.

The form has been made to accommodate both the creation and update of a record. a delete option is also provided at the bottom in order to delete the entity if needed.

**COVID-DB** Logout

**(#9) Dominic Nadeau** [< Back to Workers](#)

**FIRST NAME**  
  
First Name Required

**LAST NAME**  
  
Last Name Required

**MEDICARE ID**  
  
Medicare ID Required

**DATE OF BIRTH**

**ADDRESS**

**POSTAL CODE**

**CITY**

**PROVINCE**

**COUNTRY**

**EMAIL ADDRESS**

**PHONE NUMBER**

**GROUP ZONES**

**POSITION**

**FACILITY NAME**

**SCHEDULE INPUT**

	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
OPEN	--:-- --	--:-- --	--:-- --	--:-- --	--:-- --	--:-- --	--:-- --
CLOSE	--:-- --	--:-- --	--:-- --	--:-- --	--:-- --	--:-- --	--:-- --

COMP353 - Databases Main Project

Figure 11: This edit page provides feedback on what has not been filled out by the user

## 7.7 Toast notification

The Toast plugin allows us to provide to the user any information that is pertaining to him if they come to be.



Figure 12: A toast notification

## 7.8 Region Update

We have made the assumption that a Region cannot be deleted. Therefore, this leaves only the ability for us to update the alert level. We have therefore taken the liberty to provide a more in depth view that explains some restrictions related to the alert in place. This information can be toggled, however the information is hard coded into the application rather than in the database.

COVID-DB

Counter input

Level 3: Moderate Alert

**Level 1**  
Vigilance  
Basic Measures

- No restrictions
- Avoid unnecessary social contact

**Level 2**  
Early Warning  
Strengthened Basic Measures

- Maximum 250 people in public spaces
- No curfew

**Level 3**  
Moderate Alert  
Intermediate Measures

- Maximum 25 people in public spaces
- Curfew from 9:30 p.m to 5 a.m.
- Travel prohibited to Yellow (Level-2) zones

**Level 4**  
Maximum Alert  
Maximum Measures

- Access public spaces prohibited
- Curfew from 9:30 p.m to 5 a.m.
- Travel prohibited to Yellow (Level-2) zones
- Hybrid teaching

Submit Delete

COMP353 - Databases Main Project

Figure 13: Edit Region Form

## 8 Bonus: Additional feature

### 8.1 Twilio SMS API Integration

Using the Twilio Messaging API, We can set up a cronjob which fetches any messages that have yet to be sent

#### 8.1.1 MySQL Query

```
SELECT
    m.msg_id,
    m.message,
    p.phone
FROM Messages m
JOIN Person p ON p.person_id = m.person_id
WHERE is_sent = 0
```

#### 8.1.2 PHP Command

```
<?php
public function handle()
{
    // Fetch all unsent notifications
    $results = DB::select("SELECT
        m.msg_id,
        m.message,
        p.phone
```

```

FROM Messages m
JOIN Person p ON p.person_id = m.person_id
                WHERE is_sent = 0");

$updateMsgs = collect();
foreach ($results as $result) {
    // Only send to the configured phones
    if (in_array($result->phone, explode(',', env('TWILIO_ALLOWED_PHONES')))) {
        // Send it to the user here
        $this->line("Sending to {$result->phone}");
        $client = new Client(
            env('TWILIO_ACCOUNT_SID'),
            env('TWILIO_AUTH_TOKEN')
        );
        $client->messages->create($result->phone, [
            "body" => $result->message,
            "from" => env("TWILIO_PHONE_NUMBER"),
        ]);
    }
    $updateMsgs->add($result->msg_id);
}
$stringUpdate = $updateMsgs->join(',');
DB::update("UPDATE Messages SET is_sent=1 WHERE msg_id IN ($stringUpdate)");

$this->line("Successfully Sent " . $updateMsgs->count() . " messages.");
return 0;
}

```

### 8.1.3 Twilio Configuration

A phone number was purchased from the Twilio Console, allowing for the delivery of messages to the Canadian carrier network

### 8.1.4 Cron Execution

The cron runs every minute to verify any unsent messages created from the MySQL trigger, and proceeds to update the messages 'is\_sent' value, and sending to the appropriate avenues

## COMP 353 COVID DB Test Number

Configure Calls Log Messages Log Events Log Regulatory Information

### Properties

FRIENDLY NAME

COMP 353 COVID DB Test Number

PHONE NUMBER

+1 438 834 1890

PHONE NUMBER SID

PN469eb95d15287e9f71e2ceb366bdce09

TYPE

Local

Figure 14: Phone Configuration in the Twilio Console

```
# Development Sandbox Credentials
TWILIO_ACCOUNT_SID="ACcbb870841a96a600ed20c6c862864add"
TWILIO_AUTH_TOKEN="XXXXXXXXXXXXXXXXXXXX"
# Phone number to text or call from
TWILIO_PHONE_NUMBER="+14388341890"
# Allowed Phones to text (avoid spam on unwarranted generated phones)
TWILIO_ALLOWED_PHONES="+15144213247"
```

Figure 15: Our webapp sends text messages using the Twilio SMS API

```
→ coviddb git:(feature/twilio) x php artisan alerts:send
Sending to +15144213247
Successfully Sent 1 messages.
→ coviddb git:(feature/twilio) x
```

Figure 16: The command executes to process any unsent messages in the database

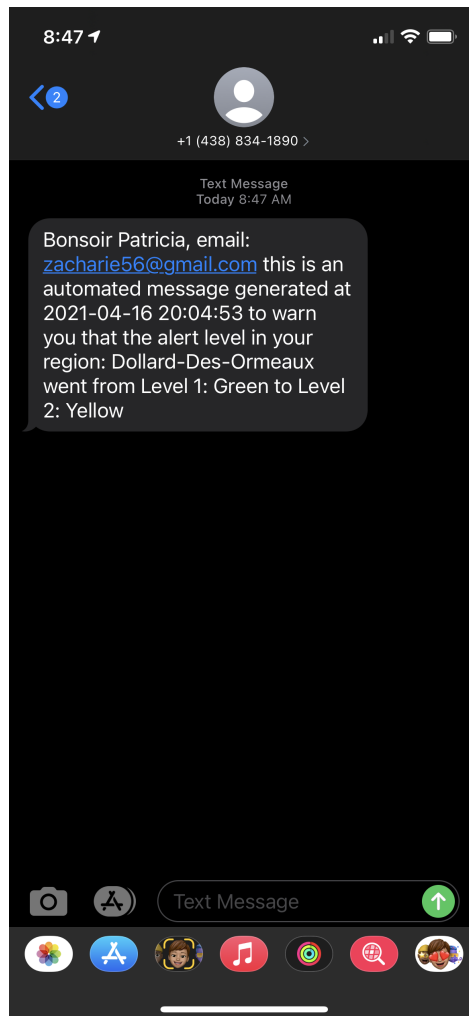


Figure 17: Our webapp sends text messages using the Twilio SMS API

## 9 Contribution & Separation of work

### **Antoine Assal - 40022745**

- E/R Diagram
- Relational Schema
- Normalization
- SQL Queries
- PHP controllers
- Report

### **Benjamin Lofo Follo - 40008156**

- Server Installation
- User Interface Components
- PHP API Integration
- Twilio Integration
- Report section 8

### **Rami Bou Abboud - 40043011**

- E/R Diagram
- Relational Schema
- SQL Table creation
- Constraints and triggers
- PHP API Integration

### **Ribelle El Ayoubi - 29276726**

- User Interface Components
- Forms for creation/edits
- User Interface Validation
- Report section 7
- References

## 10 References

- [Laravel](#)
- [ReactJS](#)
- [TailwindCSS](#)
- [Tailwind CSS components](#)
- [Normalization](#)
- [SQL queries](#)
- [Twilio SMS API](#)