Costa Rica Institute of Technology

Academical Area of Computer Engineering

I Semester 2020

Codename: **CoTEC-2020**

Technical Documentation

pandemica

Schlafenhase Development Team:

- **Jose D. Acuña** - *Web Functionality Manager & Design Assistant* - JoDaniel1412
- **Kevin Cordero** - *Lead Developer on Back-end and Connections* - Skryfall
- **Alejandro Ibarra** - *Lead Designer & Mobile App Developer* Assistant Project Manager- AlejandroIbarraC
- **Jose D. Sánchez** – *Database Administration & Definition* - JoseDavidSS
- **Jesús Yamir Sandoval** - *Project Manager & Web Developer* - shuzz22260427

Technical Advisor:

- **Marco Rivera** - *Teacher*

Course:

- **Databases** - CE 3101

Languages used:

- C#
- JavaScript / TypeScript
- Swift
- Kotlin
- MEAN stack tools (Node, Angular, Express)

A. Description of the data structure used (tables).

SQL Tables: Tables are database objects that contain all the data in a database, the information is logically organized in rows and columns. Each row represents a unique record, and each column represents a field in the record.

For the project in question, the tables were used as the main structure to organize the server information, which provides said information, previously stored in the tables, directly to the web page.

Lists: These are used in the API, Web and mobile apps to parse incoming data from the database. The main function of this data structure was to transport the information (which was organized in tables) to the front end. These types of lists follow the standard array concepts in C-like languages, which are pretty well known in the programming scope.

B. Description of implemented Triggers/Stored Procedures/Functions

1. Triggers.

- *STATE_ALTERATION_TRIGGER*: This trigger is in charge preventing the deletion or modification of the three predetermined states: Active, Recovered and Dead.
- *STATE_DELETION_FROM_PATIENT_TRIGGER*: This trigger is in charge of preventing the deletion of a patient state, so we can have a journal with all patient's states.
- *CAPACITY_CHECK_TRIGGER*: This trigger is in charge of preventing a new patient addition to a hospital if either, the hospital is at maximum capacity, the hospital's ICU unit is at maximum capacity or the patient was inserted the wrong way.

2. Stored Procedures

- *GetCountryMeasures*: This stored procedure returns all the country measures in the database.
- *InsertCountryMeasures*: This stored procedure inserts a country measure in the database.
- *UpdateCountryMeasures*: This stored procedure updates a country measure in the database.

- *GetPatientsFromHospital*: This stored procedure returns all the patients from a hospital.
- *GetContactsFromHospital*: This stored procedure returns all the contacts of a patient.
- *DeleteContact*: This stored procedure deletes a contact from a patient, and if the contact is not associated to another patient, it gets deleted from the database.
- *InsertContact*: This stored procedure inserts a contact into the database and makes a connection between the patient and the inserted person, if the person already exists, the insertion is not made and only the connection is made.
- *UpdateContact*: This stored procedure updates a contact from the database.
- *GetStatesFromPatient*: This stored procedure returns all the states from a patient.
- *DeleteStateFromPatient*: This stored procedure deletes a state from a patient. It's usually blocked by a trigger.
- *InsertStateForPatient*: This stored procedure makes an association between a state and a patient.
- *UpdateStateForPatient*: This stored procedure makes an update in a patient state.
- *GetMedicationsFromPatient*: This stored procedure returns all the medications from a patient.
- *DeleteMedicationsFromPatient*: This stored procedure deletes a medication from a patient.
- *InsertMedicationsForPatient*: This stored procedure makes an association between a medication and a patient.
- *UpdateMedicationsForPatient*: This stored procedure makes an update in a patient medication.
- *GetPathologiesFromPatient*: This stored procedure returns all the pathologies from a patient.
- *DeletePathologiesFromPatient*: This stored procedure deletes a pathology from a patient.
- *InsertPathologiesForPatient*: This stored procedure makes an association between a pathology and a patient.

- *UpdatePathologiesForPatient*: This stored procedure makes an update in a patient pathology.
- *DeleteContactsOlderThan14Days*: This stored procedure deletes a contact from a patient if it's older than 14 days.
- *CheckContacts*: This stored procedure runs the DeleteContactsOlderThan14Days stored procedure every 12 hours.
- *spCasesByCountry*: This stored procedure adds most of the country selected information from the database to a table pre-created named TEMPORARY_DATA, which will be updated every time this stored procedure is called. Finally, it returns the TEMPORARY_DATA table for the API to grab its information and show it in the front end's Home view.
- *spCasesByRegion*: This stored procedure selects all regions from the defined country. The selected table will return all regions and all cases (recovered, active, dead) of each region.
- *spAccumulatedCasesByCountry*: This stored procedure selects a country and returns a table with active cases and its cumulative value from the last 30 days.
- *spMeasurementsState*: This stored procedure selects a country and returns a table with sanitary measurements from that country, shows if the measurement is active or inactive and finally shows the date when the measurement will end.
- *spActiveDailyCases*: This stored procedure selects a country and returns a table with the active cases of that country by date from the last 30 days from the actual date.
- *spRecoveredDailyCases*: This stored procedure selects a country and returns a table with the recovered cases of that country by date of the last 30 days from the current date.
- *spDeathsDailyCases*: This stored procedure selects a country and returns a table with the deaths of that country by date of the last 30 days from the current date.

3. Functions (Front End)

- *selectChange*: This function was created with the purpose of obtaining the information about the options chosen in the select and in the Angular date picker. It works with an observed variable that notifies the TypeScript logic file of the actions that were carried out in the HTML, as the function encapsulates and stores the values in an observable data type to use as information for the server.

- *pop-up*: This function is responsible for opening local pop-up windows when the user clicks on add or edit. When doing this, an observable variable of the Boolean type is activated which indicates to the pop-up window that it should be opened or closed, depending on what is needed by the user, in addition to the type of pop-up, so it can adjust the corresponding form correctly.

C. Architecture, Class and Database Diagrams

Due to the differences in dimensions and structure of each of these diagrams, they are located in the sub-folder called "Diagrams". These include the database diagrams, located in the sub-folder "PandemicaDB" for the SQL Server database and a file called "Pandemica Health Center - ER Diagram" for the SQLite database.

Also, a general class and architecture diagram is included. Please refer to the Diagram Annotations PDF file for more information.

D. Problems found.

1.1) Problems with spreadsheet analysis: One of the first problems encountered was with sending the spreadsheet file through the network using HTTP requests. At first, an attempt was made to send the file as "Content-Type: 'application / vnd.ms-excel'".

1.2) Solution: After many failed tests, it was concluded that the correct way to send should be to embed the request with "Accept: 'multipart / form-data'"to avoid errors, this being the solution to the problem.

1.3) References:

- **郝海**东冠状病六四事件法轮功, C., & Asbury, M. (2020). What does enctype='multipart/form-data' mean?. Retrieved 3 July 2020, from https://stackoverflow.com/questions/4526273/what-does-enctype-multipart-form-data-mean

2.1) Problems with *NgFor: This was one of the least expected problems and was found when testing the CPU usage of the project.

The problem was that the *NgFor command (which is used to create a loop in HTML using the Angular web framework) consumed a lot of the resources on the PC (about 6GB of RAM and 187% of processor power)

2.2) Solution: It should be noted that this is not a problem caused by the group as such, but one due to the poor optimization of this tool (in this case the *NgFor by Angular and Google), so a total solution is basically impossible to achieve by part of the users unrelated to its maintenance and development. However, it is possible to mitigate if the resources that are made available when executing the problem is strictly regulated.

2.3) References:

- Improving Angular *ngFor Performance Through trackBy. (2020). Retrieved 3 July 2020, from https://medium.com/better-programming/improving-angular-ngfor-performance-through-trackby-ae4cf943b878

E. Known bugs.

The main (and only) problem that was encountered during the development of the project was at the time of making the subsection that had as its task the mobile application. The axis of the problem is that the architecture of the project is designed to move immense amounts of information (this because it is dealing with incoming data of all people and countries in the database at the same time).

For simple insertions, updates and deletions there are no issues, since the server can easily do that and keep the database updated the changes by means of the corresponding HTTP requests. However, because the app is supposed to keep information as up-to-date as possible, every time a change is made to the database on another client, the app's local database has to be refreshed.

This needs to happen in order to prevent conflicts between two separate database instances. Synchronization can only be done one way. To prevent these potential issues, the only logical course of action is to fetch the whole database at once, but when trying to move that much information on a simple local consumer-grade computer and refreshing many times, a bottleneck is generated in the network infrastructure, which brings down the project's API services layer.

During the project's development, we came up with various solutions to this problem: (implementing triggers, mobilizing information in parts, etc.), but they all drag the same problem: The database would no longer be updated in real time and would become a static database. This doesn't make sense due to the nature of the data we are dealing with. Patient data are people who are currently checked in a Health Center, their information is vital and cannot be outdated.

After conducting tests with each of these proposed solutions, we concluded that the concept of the requested mobile app's functionality completely breaks the paradigm on which the need to create it was founded, since although they correct the problem, they break the essence of what its goal should be in the first place, which was that all the information in the database was always updated in a mobile manner.

It is called as a "bug" by the team for technical reasons, but it would be more correct to point it out as a design flaw in the architecture as a whole, since as the desired functionality is currently established, it becomes impossible to comply with the real use-case scenarios requested for the mobile application.

For this reason, we decided to include both an online and offline mode for managing the database. Both solutions are fully implemented, but only for demonstrative purposes. We do not endorse nor recommend the technical specification as requested.

F. Conclusions and Recommendations

- Conclusions.

-- Angular is one of the most powerful and useful tools for web development.

-- Aesthetics and ease of use is of utmost importance for creating a good website.

-- Diagrams are of vital importance when creating a database. An error in this process can have terrible consequences in the project's development.

-- Organization and care is extremely important when creating a database since the slightest error can cause great long-term damage to its implementation or use.

-- Triggers must be used with great care, as they can cause problems if they are not handled correctly

-- Stored Procedures are extremely useful when performing various methods that can be reused multiple times

- Recommendations.

-- Read the Angular documentation before even think of starting a web project.

-- Make a good conceptual and ER diagram before starting work on the database.

-- Try to implement Stored Procedures to achieve an efficient use of code.

-- Make a good task plan and a correct division of all team member's work.

-- Maintain good database administration by documenting all code blocks, tables, Stored Procedures, Triggers and more.

G. Blog and bibliography

The following section aims to describe in a general way the tasks carried out by different members of the Schlafenhase main development team during the various periods of time involved in the project. If you want a more specific view of the tasks performed (time, manager, etc.), please review the attached work plan, located in "Docs/Technical" folder.

01-05-20 to 4-06-20:

The first days of the project were used to carry out the work plan and division of tasks. For this purpose, the Trello platform was used, where tasks were assigned to each member of the group, each task had a deadline that had to be fulfilled without delays. In order to maintain a good work organization, the project was divided into 3 major phases. In each phase, there were a series of tasks to be carried out for each member, thus maximizing the efficiency of the work performed.

The tool used for audio meetings was Discord, as everyone worked from hoe due to the ongoing COVID-19 pandemic (kinda ironic, actually). The group met in the channel called "Régimen Corderista - Virtual Sessions", with a call time of 1:23h and the presence of all the members of the group

Sites used this day:

Pandemica Task Plan. (2020). Retrieved 3 July 2020, from https://trello.com/b/25jVTfzM/pandemica-task-plan

05-06-20 to 10-06-20:

The main tasks that were carried out in the days after planning were those of design. This task was carried out by J.A Ibarra, who successfully created view models to visualize how the front end wanted to be shown to users. To carry out this task as Lead Designer, Alejandro made use of Figma, in which he has vast experience. This tool allowed him to create models for all the views and distribute them among the members in charge of making them so that they could replicate them in the HTML and CSS code.

Another important task was the creation of the various initial diagrams to assemble the database (both relational and conceptual). This task was divided equally among all the members of the group, this in order to obtain the greatest participation as a team in knowing how it would work and how the data would be stored.

Along with the aforementioned details, the realization of the various views in the HTML code began, as well as the creation of the tables in which the information was to be shown to the users of the page.

Sites used these days:

Pandemica – Web App. (2020). Retrieved 4 July 2020, from https://www.figma.com/file/eL6Pf6gfbMHfsF4fOaBrif/Web-App?node-id=0%3A1https://angular.io/cli/build

How to build a dynamic table component in Angular 9?. (2020). Retrieved 4 July 2020, from https://medium.com/@the_harsha_chinni/how-to-build-a-dynamic-table-component-in-angular-9-9aacc3d81b08

*ngFor, a., Jain, P., Jain, P., & Templier, T. (2020). access key and value of object using *ngFor. Retrieved 4 July 2020, from https://stackoverflow.com/questions/35534959/access-key-and-value-of-object-using-ngfor

Team, A. (2020). Angular Material. Retrieved 4 July 2020, from https://material.angular.io/components/dialog

12-06-20 to 17-06-20:

The main tasks carried out these days were the complete implementation of the log-in system for the authentication of users and creating the main view "Home View". This was carried out by the group member J.D Acuña. Along with this, also by Jose Daniel, the investigation to carry out the task specified in the subsection for spreadsheet analysis began. He started an exhaustive search and started the first implementations of how to load a .xlsx file from the web page and that it had a successful parsing.

Another main task performed on this date was by the Back-end team, which was made up of K.S Cordero and J.D Sánchez. This team had the task of implementing the API and database, so in this period of time they were focused only on that task, which they carried out successfully due to good teamwork and experience gained in previous projects. Thanks to this, the database was quickly implemented, and tests were carried out within the established period. However, future tests will continue to be carried out to corroborate possible incidents

The last two tasks of this period were the implementation of the mobile application, in charge of J.A Ibarra and the completion of the "Admin View" view, which was in charge of J.Y Sandoval. Both tasks were successful and as planned.

Sites used these days:

Shpilt, M. (2020). Find, Fix, and Avoid Performance Problems in C# .NET: 10 Best Practices - Michael's Coding Spot. Retrieved 4 July 2020, from https://michaelscodingspot.com/performance-problems-in-csharp-dotnet/

Reading Excel files from C#. (2020). Retrieved 4 July 2020, from https://stackoverflow.com/questions/15828/reading-excel-files-from-c-sharp

Calbimonte, D. (2020). Are SQL Server database triggers evil?. Retrieved 4 July 2020, from https://www.sqlshack.com/are-sql-server-database-triggers-evil/

Angular 8 - User Registration and Login Example & Tutorial | Jason Watmore's Blog. (2020). Retrieved 4 July 2020, from https://jasonwatmore.com/post/2019/06/10/angular-8-user-registration-and-login-example-tutorial

18-06-20 to 25-06-20:

The last week of time stipulated according to the work plan was used mostly as testing and adjustment time, mainly in the database (due to the implementation of new SPs and triggers and the elimination of others). full implementation of excel within views, obtaining a successful result as expected

The other set of tasks was carried out by J.A Ibarra and J.Y Sandoval, who were in charge of the main parts of the external documentation (diagrams, manuals, etc.). For this task tools such as "Microsoft Word" for drafting and "Canva" for stylization of the documents were used.

Sites used these days:

Slack. (2020). Retrieved 4 July 2020, from https://app.slack.com/client/T016QC238PN/learning-slack

Problems With Crystal Report on Windows 10 Computers - SAP Q&A. (2020). Retrieved 4 July 2020, from https://answers.sap.com/questions/750296/problems-with-crystal-report-on-windows-10-compute.html

(2020). Retrieved 4 July 2020, from https://www.canva.com/es_419/

NOTE: Project Weekly Summaries were made every week during development. These are attached in the project's structure, inside the folder "Docs/Weekly Summaries". Also, a spreadsheet offline copy of the finalized Trello task plan that can be opened using Microsoft Excel is found on the root of "Docs/Technical" folder.