**HIS Graduation Project Book**

***Prepared by: Ibrahim Khalaff***

***Mohamed Tarek***

***Omar ElGamily***

***Mazen ElSharkawy***

***Ahmed Ibrahim***

***Hasan Mohamed***

**Abstract**

Healthcare is an extremely important field in our everyday lives and with technology evolving ever-so-quickly on a daily basis, so does healthcare. In ICT, there is a branch specifically created to develop and evolve healthcare technologies called Healthcare informatics. Healthcare informatics is the intersection between information science, computer science and healthcare. It is the field of study that handles the management and use of patient health care information. With the development and growth of this field, hospitals began to abandon paper-based work in favor of more efficient computerized applications that help them acquire, store, retrieve and use information that helps optimize the hospital’s operation and benefit. One such example is the Hospital Information System (HIS), a system focusing mainly on the administrational needs of hospitals to help hospitals keep a digital track of all its daily operations. In our graduation project, we will implement a Hospital Information System and in this report, we will discuss with great detail what an HIS is and our plan to successfully implement one.

**Introduction**

Before we delve into the HIS, we need to get acquainted with the concept of a computer information system (CIS). A computer information system is a formal, organizational system designed to collect, process, store and distribute information. It is primarily composed of five components: task, people, computers, structure/role and technology. There are five types of information systems:

1. Executive Support System (ESS): this type of IS was designed to help senior management support the business and make strategic decisions. Gathering, analyzing and summarizing the key internal and external information used in the everyday business.
2. Management Information systems (MIS): it is mostly concerned with internal sources of information. These systems usually take data from TPS and summarize it into management reports.
3. Transaction Processing Systems (TPS): TPS are designed to process repetitive transactions efficiently and accurately. A business may have more than one TPS.
4. Decision Support System (DSS): DSS is an information system intended to help users reach a decision when a decision-making situation arises. It comprises tools and techniques to help collect relevant information to analyze the choices and alternatives we have to make a well-informed decision to help with the situation at hand.
5. Knowledge Management Systems (KMS): KMS exist to help businesses create and share various information. They are typically used in industries where employees create new knowledge and expertise.
6. Office Automation Systems (OAS): systems that try to improve the efficiency of employees who need to process data and information.

Naturally, Information Systems face multiple technical problems and such problems include:

* Lack of immediate retrievals: The information is very difficult to retrieve and to find particular information like- E.g. - To find out about the patient’s history, the user has to go through various registers. This results in inconvenience and waste of time.
* Lack of immediate information storage: The information generated by various transactions takes time and efforts to be stored at right place.
* Lack of prompt updating: Various changes to information like patient details or immunization details of child are difficult to make as paper work is involved.
* Error prone manual calculation: Manual calculations are error prone and take a lot of time this may result in incorrect information. For example, calculation of patient’s bill based on various treatments.
* Preparation of accurate and prompt reports: This becomes a difficult task as information is difficult to collect from various registers.

Now that we are acquainted with the concept of the information system, we can now delve into the HIS. An HIS is a comprehensive, integrated computer information system designed to manage all the aspects of a hospital’s operation, such as medical, administrative, financial and legal issues and the corresponding processing of services. It is also known as Hospital Management Software/system (HMS). Multiple studies and researches were conducted in different countries on HIS and its applications all over the world. Each of these studies has helped us gain more information and perspective on how to develop our own version of the HIS. An HIS consists of multiple information systems with each one tending to a certain field or aspect specific to the hospital as we discussed earlier in its definition. Each one of those systems falls under one or more of the different categories of a general information system as well. HISs are in high demand due to increasing population needs and to aid doctors and hospital staff with timely service and precision. Progressing through this report, we will discuss and learn more about every system from which the HIS is comprised and all the benefits that the HIS provides to the hospital in terms of managing all its data.

**The Aims and goals of HIS**

The HIS’s main objective is to gather, store and retrieve data to optimize the hospital’s day-to-day operations. The HIS covers all the hospital’s needs in terms of business and medical care aspects to achieve the best possible support of patient care and outcome and administration by presenting data where needed and acquiring data where generated by electronic data processing. The main demands of an HIS are correct data storage, where you store the acquired data in its appropriate system, reliable usage, keeping the data as up-to-date as possible in order to ensure more efficient practice, fast to reach data, ensuring that the data be retrieved as quickly as possible, secure to keep data on storage, to ensure that no data leaks outside of the hospital at any time and finally, lower cost on usage, so that it optimizes its operations with the least possible cost it may consume whether financially or technically. It provides a common source of information about a patient’s health history, for example, who is the m.d. following their case, their basic personal information and so on. Furthermore, it has to keep the data in secure places and control who can reach the data under which circumstances. The main goals of HIS are planned approach towards working, accuracy, reliability, no redundancy, immediate retrieval of information, immediate storage of information and easiness of operation. The HIS also reduces the possibility of error that hand-written tracking of information may have.

**Why do we need HIS?**

Hospital management has greatly changed over the last decades. Business expertise, modern technologies, connected devices, mobile apps, and knowledge of healthcare are key elements for the implementation of hospital management system project. The number of healthcare providers has increased and the patients have a wide choice of medical specialists. The interactions between the hospital and the patient can be simplified for the convenience of both sides. Each institution has the opportunity to create the efficient, clear and fast delivering healthcare model. There is always the wide choice of features that can be included in the system. Moreover, the most important thing they are created to streamline various procedures that meet the needs of all the users. The hospital management system feature list is concentrated on providing the smooth experience of patients, staff and hospital authorities. It might seem that their expectations differ, they still are covered by components of the hospital information system. Quality and security still remain the main criteria of the medical industry. It is also known for the constant and rapid changes to improve the efficiency of medical services and satisfaction of the patients. As long as each stage implementation needs to be accurate and explicit, the clinic management system provides certain automation of many vital daily processes. The hospital system software covers the services that unify and simplify the work of healthcare professionals as well as their interactions with patients. It is also known for the constant and rapid changes to improve the efficiency of medical services and satisfaction of the patients.

**The Benefits of HIS**

* Easier access to data for doctors to generate different records, including diversification based on demographic, gender, age and so on. It is most helpful in ambulatory (out-patient) point. Hence improving continuity of care.
* Improved quality and efficiency of patient care.
* Helps hospital authorities develop comprehensive and well-informed healthcare policies.
* Efficient and accurate administration of finance, diet of patient, engineering and distribution of medical aid. Helps view a broad picture of hospital growth.
* Improved tracking and monitoring of drug usage and effectiveness. Thus, it leads to the reduction of adverse drug interactions while encouraging more efficient pharmaceutical utilization and administration.
* Dramatically improving the quality of documentation.
* Majorly increasing information integrity.
* Reduction of transcription errors and duplication of information entries.
* More convenient to use and effectively eliminating errors from paper-based recording.
* Development of a common clinical database.

**Who and what Benefits from HIS**

The information gathered by the HIS can benefit multiple directions in the hospital between facilities and staff, such as physicians, nurses, wards, administration, clinics and so on. It is also gathered from directions outside of the hospital, mainly patients to help track their information with optimized efficiency and treat their condition.

**Physicians**

Physicians are the ones who benefit the most from HISs, as they are the ones who use most of the information provided by the HIS to help attend to their patients’ treatment.

* Introduces the concept of Computerized Physician Order Entry (CPOE), which is the process of electronic entry of medical practitioner instructions for the treatment of patients under their care.
* Improves accuracy, legibility of and access to the required patient medications. Therefore, better tracking of their conditions improvement.
* Improving the efficiency and effectiveness of clinicians through the provision of key patient information at time of ordering, in addition to checking schedule conflicts as well as orders and online access to best practice information.
* Logging of all orders to improve care.
* Medication error rates are dramatically reduced.

**Nurses and Allied Health Professionals**

Nurses and allied health professionals such as nutritionists, personal trainers, psychologists, speech and language pathologists, massage therapists and paramedics, are also vital users of the HIS since they also attend to patients on either a daily or frequent basis.

The HIS allows and provides the following benefits:

* Immediate access to orders and results, therefore speeding up the response to the patients’ needs.
* Immediate access to patients’ demographics, personal information, medication and test results, in order to check and change their frequency according to the patient’s condition.
* Improved access to information online, in order to conduct searches and further improve a patient’s treatment plan.
* Decreasing the need for papers, errors and increasing patient safety.

**Ward and Registration Clerks**

When patients check in to a hospital, they must first give their personal information to clerks in order to enter them into the hospital and check if there are any services available for the patients. The HIS provides a single point of contact for patient registration information and reduces effort duplication.

**Clinical Benefits**

The most important facilities that benefit from the HIS in a hospital are clinics. Since the physicians need to retrieve the data for their patients on a near daily basis.

The HIS provides the following benefits for clinics:

* A common source for a person’s health history and therefore providing better insight into their treatment.
* Improving the ability of healthcare professionals to coordinate care by providing a person’s health information and visit history at the time and place of need.
* Allow care providers access to the patient’s health history and results between facilities.
* Strengthen communication between healthcare providers.
* Improved access to information online, in order to conduct searches and further improve a patient’s treatment plan.
* Decreasing the need for papers, errors and increasing patient safety.

**Administrative Benefits**

* Strengthen communication between healthcare providers.
* Improved access to information online, in order to conduct searches and further improve a patient’s treatment plan.
* Decreasing the need for papers, errors and increasing patient safety.
* Decreasing the need for patient re-registration across multiple sites, therefore reducing redundant effort.

**Types of HIS**

In Malaysia, there was a study conducted to actually improve and update the concept of HIS in order to optimize their operation and perhaps even implement more features that may help hospitals operate in better capacity. In this study, the HIS has been divided into three types with different scales:

1. Basic HIS (BHIS): this system only implements two of the systems we discussed earlier, mainly Clinical Information System and Patient Management System, which can implement the Patient Registration Scheduling Module but can also be integrated within the CIS.

2. Intermediate HIS (IHIS): this version integrates the BHIS and adds the Laboratory Information System (LIS) and the Pharmacy Information System to its implementation.

3. Total HIS (THIS): This version integrates the IHIS and the rest of the systems discussed through this paper, but this is the costliest out of all 3 types in terms of implementation and finance.

**Implementation and Design**

Naturally, as we begin to implement our project, the first thing we must think about is how to implement our database and we have opted to use SQL for several reasons. As we know, of course, SQL is a domain-specific language used in programming and designed to manage data in relational databases management systems (RDBMS). It is particularly useful in handling structured i.e. data that incorporates relations among entities and relations. An HIS will most definitely receive so many bits of information on a daily basis and will need a place to safely store this information and would act as a reference for the doctors should any patients come and revisit the hospital, which should immensely help with their treatment planning and projection of what they need and could use. One of the most important procedural extensions of SQL is MySQL, which is a major evolution on SQL since its inception in 1974 as MySQL has been introduced in 1995. MySQL can be written in c and c++ programming languages. MySQL has several advantages, including:

1. It’s easy to use: easy to install, setting up an implementation is a relatively easy task due a multitude of third-party tools, it’s an easy database to work with and finally, so long as you understand the language well, you shouldn’t run into too many problems.

2. Support is readily available whenever necessary: this is due in large part to the popularity the MySQL language has garnered since its inception among the developers and enthusiasts’ communities to which one could easily turn for help and therefore MySQL has no shortage of experts.

3. It’s Open-source: it is an open-source database option as its code is still available for free online.

4. It’s incredibly inexpensive: it’s significantly less expensive than multiple database options on the market.

5. It’s an industry standard: it’s compatible with virtually every operating system and therefore has been cemented as an industry standard.

**Further benefits for database design using MySQL**

1. Data Security: MySQL is globally renowned for being the most secure and reliable database management system used in popular web applications, such as WordPress, Facebook and Twitter to name a few. The security and support for transactional processing that accompany the most recent version of MySQL can greatly benefit any business, especially in healthcare as we’re about to find out in our project.

2. High Performance: MySQL features a distinct storage-engine framework that facilitates system administrators to configure the MySQL database server for a flawless performance. MySQL is designed to meet even the most demanding applications while ensuring optimum speed, full-text indexes and unique memory caches for enhanced performance. We could use that with our HIS considering how many patients go in and out of the hospital on a daily basis for example or the staff’s registrations as well.

3. Complete Workflow Control: MySQL is a comprehensive solution with self-management features that automate everything from space expansion and configuration to data design and database administration.

**Improving MySQL Performance**

1. Performance Tuning: A database load balancing software is designed to deliver the agility and scalability needed to expand capabilities and meet both unplanned and anticipated performance demands.

2. Security Audits: You can easily prevent performance issues and increase uptime with a robust load balancing software that ensures automatic failover and security updates.

3. Optimizing Queries: SQL server load balancing software is a one-stop solution for maintaining uptime, data consistency, increasing performance, reducing service costs, and ensuring continuous availability for an enhanced customer experience. It does everything from running health checks and maintaining high performance to lowering the query wait time and enabling even distribution of load across multiple servers.

**Electronic Health Records (EHR)**

An EHR is a digital record that can provide comprehensive health information about your patients. EHR systems are built to share information with other health care providers and organizations – such as laboratories, specialists, medical imaging facilities, pharmacies, emergency facilities, and school and workplace clinics – so they contain information from all clinicians involved in a patient’s care. EHRs are real-time, patient-centered records that make information available instantly and securely to authorized users. While an EHR does contain the medical and treatment histories of patients, an EHR system is built to go beyond standard clinical data collected in a provider’s office and can be inclusive of a broader view of a patient’s care.

**Types of Information in an EHR:**

1. Administrative and billing data

2. Patient demographics

3. Progress notes

4. Vital signs

5. Medical histories

6. Diagnoses

7. Medications

8. Immunization dates

9. Allergies

10. Radiology Images

11. Lab and test results

**Functions of EHR:**

1. Contain a patient’s medical history, diagnoses, medications, treatment plans, immunization dates, allergies, radiology images, and laboratory and test results

2. Allow access to evidence-based tools that providers can use to make decisions about a patient’s care  
3. Automate and streamline provider workflow

**Benefits of EHRs**

1. Improved Patient Care:

Snapshot of Improved Health Care Quality and Convenience for Providers:

* Quick access to patient records from inpatient and remote locations for more coordinated, efficient care
* Enhanced decision support, clinical alerts, reminders, and medical information
* Performance-improving tools, real-time quality reporting
* Legible, complete documentation that facilitates accurate coding and billing
* Interfaces with labs, registries, and other EHRs
* Safer, more reliable prescribing

Snapshot of Improved Health Care Quality and Convenience for Patients:

* Reduced need to fill out the same forms at each office visit
* Reliable point-of-care information and reminders notifying providers of important health interventions
* Convenience of e-prescriptions electronically sent to pharmacy
* Patient portals with online interaction for providers
* Electronic referrals allowing easier access to follow-up care with specialists

2. Increase patient participation in their care:

Electronic health records (EHRs) can help providers:

* **Ensure high-quality care.** With EHRs, providers can give patients full and accurate information about all of their medical evaluations. Providers can also offer follow-up information after an office visit or a hospital stay, such as self-care instructions, reminders for other follow-up care, and links to web resources.
* **Create an avenue for communication with their patients.** With EHRs, providers can manage appointment schedules electronically and exchange e-mail with their patients. Quick and easy communication between patients and providers may help providers identify symptoms earlier. And it can position providers to be more proactive by reaching out to patients. Providers can also provide information to their patients through patient portals tied into their EHR system.

3. Improved Care Condition:

**How EHRs Can Improve Care Coordination**

Electronic health record (EHR) systems can decrease the fragmentation of care by improving care coordination. EHRs have the potential to integrate and organize patient health information and facilitate its instant distribution among all authorized providers involved in a patient's care. For example, EHR alerts can be used to notify providers when a patient has been in the hospital, allowing them to proactively follow up with the patient.

**With EHRs, every provider can have the same accurate and up-to-date information about a patient. This is especially important with patients who are:**

* Seeing multiple specialists
* Receiving treatment in emergency settings
* Making transitions between care settings

Better availability of patient information can reduce medical errors and unnecessary tests.

Better availability of information can also reduce the chance that one specialist will not know about an unrelated (but relevant) condition being managed by another specialist.

**Better care coordination can lead to better quality of care and improved patient outcomes.**

4. Improved Diagnostics & Patient Outcomes:

EHRs can reduce errors, improve patient safety, and support better patient outcomes

How? EHRs don't just contain or transmit information; they "compute" it. That means that EHRs manipulate the information in ways that make a difference for patients. For example:

* A qualified EHR not only keeps a record of a patient's medications or allergies, it also automatically checks for problems whenever a new medication is prescribed and alerts the clinician to potential conflicts.
* Information gathered by a primary care provider and recorded in an EHR tells a clinician in the emergency department about a patient's life-threatening allergy, and emergency staff can adjust care appropriately, even if the patient is unconscious.
* EHRs can expose potential safety problems when they occur, helping providers avoid more serious consequences for patients and leading to better patient outcomes.
* EHRs can help providers quickly and systematically identify and correct operational problems. In a paper-based setting, identifying such problems is much more difficult, and correcting them can take years.

Risk Management and Liability Prevention: Study Findings

EHRs May Improve Risk Management by:

* Providing clinical alerts and reminders
* Improving aggregation, analysis, and communication of patient information
* Making it easier to consider all aspects of a patient’s condition
* Supporting diagnostic and therapeutic decision making
* Gathering all relevant information (lab results, etc.) in one place
* Support for therapeutic decisions
* Enabling evidence-based decisions at point of care
* Preventing adverse events
* Providing built-in safeguards against prescribing treatments that would result in adverse events
* Enhancing research and monitoring for improvements in clinical quality

Certified EHRs May Help Providers Prevent Liability Actions By:

* Demonstrating adherence to the best evidence-based practices
* Producing complete, legible records readily available for the defense (reconstructing what actually happened during the point of care)
* Disclosing evidence that suggest informed consent

EHRs can improve public health outcomes

EHRs can also have beneficial effects on the health of groups of patients.

Providers who have electronic health information about the entire population of patients they serve can look more meaningfully at the needs of patients who:

* Suffer from a specific condition
* Are eligible for specific preventive measures
* Are currently taking specific medications

This EHR function helps providers identify and work with patients to manage specific risk factors or combinations of risk factors to improve patient outcomes.

For example, providers might wish to identify:

* How many patients with hypertension have their blood pressure under control.
* How many patients with diabetes have their blood sugar measurements in the target range and have had appropriate screening tests.

This EHR function also can detect patterns of potentially related adverse events and enable at-risk patients to be notified quickly.

5. Medical Practice Efficiencies & Cost Savings:

Many health care providers have found that electronic health records (EHRs) help improve medical practice management by increasing practice efficiencies and cost savings. EHRs benefits medical practices in a variety of ways, including:

* Reduced transcription costs
* Reduced chart pull, storage, and re-filing costs
* Improved documentation and automated coding capabilities
* Reduced medical errors through better access to patient data and error prevention alerts
* Improved patient health/quality of care through better disease management and patient education.

**EHR vs EMR: What is the difference?**

An EMR is a fully integrated module that attends to medical and clinical patient records. It supports all medical professionals in various departments and provides them with all relevant information such as medical examinations, diagnoses, test results and so on. It provides access to critical and complete patient data to ensure high quality, cost effective and efficient patient care. It provides information to the larger and more information filled EHR to help optimize its efficiency as well in its functions. It has multiple advantages including:

* Track data over time
* Easily identify which patients are due for preventive screenings or checkups
* Check how their patients are doing on certain parameters—such as blood pressure readings or vaccinations
* Monitor and improve overall quality of care within the practice

However, the EHR does all the EMR functions and more and so we could say that an EMR could be integrated as a part of the extensive content of the EHR which has a broader and stronger image of the hospital’s patients and track records.  EHRs focus on the total health of the patient—going beyond standard clinical data collected in the provider’s office and inclusive of a broader view on a patient’s care. EHRs are designed to reach out *beyond* the health organization that originally collects and compiles the information. They are built to share information with other health care providers, such as laboratories and specialists, so they contain information from *all the clinicians involved in the patient’s care*. The National Alliance for Health Information Technology stated that EHR data “can be created, managed, and consulted by authorized clinicians and staff across more than one healthcare organization.” The information moves with the patient—to the specialist, the hospital, the nursing home, the next state or even across the country. In comparing the differences between record types, HIMSS Analytics stated that, “The EHR represents the ability to easily share medical information among stakeholders and to have a patient’s information follow him or her through the various modalities of care engaged by that individual.” EHRs are designed to be accessed by all people involved in the patients care—*including the patients themselves*. Indeed, that is an explicit expectation in the Stage 1 definition of “**meaningful use**” of EHRs.

And that makes all the difference. Because when information is shared in a secure way, it becomes more powerful. Health care is a team effort, and shared information supports that effort. After all, much of the value derived from the health care delivery system results from the effective communication of information from one party to another and, ultimately, the ability of multiple parties to engage in interactive communication of information.

**EHR Code**

**EHRDb.js**

This Class is implemented to store all the EHRs recorded by the doctors for the patients in their own database.

import Promise from "bluebird";

import { MongoClient } from "mongodb";

import config from "../config";

let production = false; //process.env.NODE\_ENV === 'production';

const key = config.database;

const dbName = "HIS";

let db = null;

let connect = (cb) => {

console.log("Connecting to db");

MongoClient.connect(

key,

{

useNewUrlParser: true,

},

(err, database) => {

if (err) return console.log(err);

console.log("db connected");

db = database.db(

process.env.NODE\_ENV !== "production" ? "blloc-staging" : "blloc"

);

cb && cb();

}

);

};

export default class Database {

static databaseObj(cb) {

if (!db) return connect(() => cb(db));

return cb(db);

}

static async openDb() {

return MongoClient.connect(key, { useNewUrlParser: true }).then(

(dbc) => (db = dbc.db(dbName))

);

}

static async read({ tableName, query }) {

if (!db) await this.openDb();

return db.collection(tableName).find(query).toArray();

}

static async write({ tableName, object }) {

if (!db) await this.openDb();

console.log("DB: " + db.dbName);

console.log("Object to write: " + object);

return await db.collection(tableName).insertOne(object);

}

}

export class AsyncDatabase {

static async update({ collection, query, update }) {

return new Promise(async (resolve, reject) => {

let updated = null;

if (!db) await AsyncDatabase.connect();

updated = db

.collection(collection)

.update(query, update, { upsert: false });

if (updated) resolve(updated);

else reject(updated);

});

}

static async findOneAndUpdate({ collection, query, update }) {

return new Promise(async (resolve, reject) => {

if (!db) await AsyncDatabase.connect();

db.collection(collection).findOneAndUpdate(

query,

update,

{ upsert: false, returnOriginal: false },

(err, result) => {

if (err) reject(err);

resolve(result.value);

}

);

});

}

static async connect() {

return new Promise((resolve, reject) => {

connect(resolve);

});

}

static async writeToDb({ data, table }) {

return new Promise(async (resolve, reject) => {

if (!db) await AsyncDatabase.connect();

db.collection(table).insertMany(data, (err, res) => {

if (err) reject(err);

else resolve("done");

});

});

}

static async findMany({ query, table }) {

return new Promise(async (resolve, reject) => {

if (!db) await Database.openDb();

db.collection(table)

.find(query)

.toArray((err, record) => {

if (err || !record) return reject("not found");

else resolve(record);

});

});

}

static async findDb({ query, table }) {

return new Promise(async (resolve, reject) => {

if (!db) await Database.openDb();

db.collection(table).findOne(query, (err, record) => {

if (err || !record) return reject("not found");

else resolve(record);

});

});

}

}

**Clinicsdb.js**

This database is created specifically for the clinics’ EHRs and other data concerning its primary doctors and patients.

import { MongoClient } from "mongodb";

import config from "../config";

const key = config.database;

const dbName = "HIS";

var db = null;

const createConnection = async () => {

return new Promise((resolve, reject) => {

MongoClient.connect(

key,

{

useNewUrlParser: true,

},

(err, database) => {

if (err) reject(err);

db = database.db(dbName);

db.once("open", () => console.log("connected to db"));

resolve();

}

);

});

};

export const read = async (table, query) => {

if (!db) await createConnection();

return await db.collection(table).find(query).toArray();

};

/\*\*

\* return value => Promise

\*/

export const insert = (table, data) => {

return new Promise(async (resolve, reject) => {

if (!db) await createConnection();

db.collection(table).insertOne(data, (err, res) => {

if (err) reject("error occured durin insertion");

resolve("object inserted");

});

});

};

**EHR.js**

This class implements and creates the EHR’s format and all the data it contains before inserting it into the EHR database.

import Router from "express";

import Database from "../db/EHRdb";

import { verifyTokenMiddleware } from "../middlewares/token";

const router = Router();

let tableName = "EHR";

const getAllEHR = async (req, res) => {

//get \_id from sql and sub in query

let query = {};

let EHR = await Database.read({

tableName,

query,

});

res.status(200).send(EHR);

};

const getEHRByID = async (req, res) => {

let { ID } = req.body;

let query = {

"Recipient.NathionalID": { $regex: `.\*${ID}.\*` },

};

let EHR = await Database.read({

tableName,

query,

});

res.status(200).send(EHR);

};

const getEHRByPhone = async (req, res) => {

let { Phone } = req.body;

let query = {

"Recipient.Demographic.Phone": { $regex: `.\*${Phone}.\*` },

};

let EHR = await Database.read({

tableName,

query,

});

res.status(200).send(EHR);

};

const addRecipient = async (req, res) => {

let {

NathionalID,

FirstName,

LastName,

Title,

Address,

Phone,

DOB,

Sex,

EmergencyContact,

} = req.body;

let object = {

Recipient: {

NathionalID,

Name: {

FirstName,

LastName,

Title,

},

Demographic: {

Address,

Phone,

},

DOB: new Date(DOB),

Sex,

EmergencyContact,

},

Encounter: [],

};

let writeStatus = await Database.write({ tableName, object });

if (writeStatus.result.ok === 1) res.status(200).send({ inserted: true });

else res.status(400).send({ inserted: false });

};

router.get("/getallEHR", getAllEHR);

router.post("/getEHRbyid", getEHRByID);

router.post("/getEHRbyphone", getEHRByPhone);

router.post("/addrecipient", addRecipient);

export default router;

**Clinics.js**

This class enters the operating times of the clinics and further information about each clinic before storing it in the clinics database.

import Router from "express";

import { insert } from "../db/clinicsdb";

import {

clinicValidation,

timingValidation,

} from "../middlewares/dataValidation";

const router = Router();

const insertClinic = async (req, res) => {

let data = req.body;

insert("Clinic", data)

.then(() => {

res.status(200).send();

})

.catch((err) => {

console.log(err);

res.status(400).send();

});

};

const insertTiming = async (req, res) => {

let data = req.body;

insert("Timing", data)

.then(() => {

res.status(200).send();

})

.catch((err) => {

console.log(err);

res.status(400).send();

});

};

router.post("/insert-clinic", clinicValidation, insertClinic);

router.post("/insert-timing", timingValidation, insertTiming);

export default router;

**DataValidation.js**

This class is implemented to ensure that the data entered as clinic information is valid whether it’s timing or other pieces of information

import \_ from "lodash";

export const clinicValidation = (req, res, next) => {

let data = req.body;

let errors = {};

if (!data["name"]) errors["name missing"] = 1;

if (!data["address"]) errors["address missing"] = 1;

if (!data["open"]) errors["open time missing"] = 1;

if (!data["close"]) errors["close time missing"] = 1;

if (typeof data["open"] != "object" || typeof data["close"] != "object")

if (!errors["open time missing"] && !errors["close time missing"])

errors["open and close should be Date format"] = 1;

if (typeof data["name"] != "string" || typeof data["address"] != "string")

if (!errors["name missing"] && !errors["address missing"])

errors["name and address should be strings"] = 1;

if (\_.keys(errors) == 0) next();

else {

let err = "Data invalid: " + \_.keys(errors).toString();

res.status(400).send(err);

}

};

export const timingValidation = (req, res, next) => {

let data = req.body;

let errors = {};

if (!data["doctor"]) errors["doctor \_id missing"] = 1;

if (!data["clinic"]) errors["clinic \_id missing"] = 1;

if (!data["start"]) errors["start time missing"] = 1;

if (!data["end"]) errors["end time missing"] = 1;

if (typeof data["start"] != "object" || typeof data["end"] != "object")

if (!errors["start time missing"] && !errors["end time missing"])

errors["start and end should be 'Date' format"] = 1;

if (typeof data["doctor"] != "string" || typeof data["clinic"] != "string")

if (!errors["doctor \_id missing"] && !errors["clinic \_id missing"])

errors["doctor and clinic should be \_id strings"] = 1;

if (\_.keys(errors) == 0) next();

else {

let err = "Data invalid: " + \_.keys(errors).toString();

res.status(400).send(err);

}

};

**Token.js**

The token class ensures that the token returned from storing the EHR on the system is verified and valid

const jwt = require("jsonwebtoken");

export function verifyTokenMiddleware(req, res, next) {

// Get auth header value

const token = req.headers["authorization"];

// Check if bearer is undefined

if (typeof token !== "undefined") {

// Set the token

req.token = token;

// Next middleware

verifyTokenValid(req.token, (valid) => {

if (!valid.success)

res.status(403).send({

success: false,

token: false,

err: "token not valid",

});

else next();

});

} else {

// Forbidden

res.sendStatus(403);

}

}

function verifyTokenValid(token, cb) {

jwt.verify(token, SECRET\_KEY, (err, data) => {

if (err) cb({ success: false });

else cb({ success: true });

});

}

**Mongo Database**

In our micro-services, we have also used Mongo Database technology, which is a NoSQL database. NoSQL stands for “Not Only SQL” is an alternative to the relational database. In the relational database model, tables are used to store data. But the NoSQL has a variety of data models, like key-value pairs and graph formats. One of the most flexible ways of these models is the key-value pair. MongoDB is one popular NoSQL database that stores data in BSON. BSON is binary encoding JSON which stores data in key-value pairs.

**Main Features:**

1. Ad-hoc queries: MongoDB supports field, range query, and regular-expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions. Queries can also be configured to return a random sample of results of a given size.

2. Indexing: Fields in a MongoDB document can be indexed with primary and secondary indices.

3. Replication: MongoDB provides high availability with replica sets. A replica set consists of two or more copies of the data. Each replica-set member may act in the role of primary or secondary replica at any time. All writes and reads are done on the primary replica by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can optionally serve read operations, but that data is only eventually consistent by default.

4. Load Balancing: MongoDB scales horizontally using sharding. The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more replicas.). Alternatively, the shard key can be hashed to map to a shard – enabling an even data distribution. MongoDB can run over multiple servers, balancing the load or duplicating data to keep the system up and running in case of hardware failure.

5. File storage: MongoDB can be used as a file system, called GridFS, with load balancing and data replication features over multiple machines for storing files. This function, called grid file system, is included with MongoDB drivers. MongoDB exposes functions for file manipulation and content to developers. GridFS can be accessed using mongofiles utility or plugins for Nginx and lighttpd. GridFS divides a file into parts, or chunks, and stores each of those chunks as a separate document.

**Advantages of MongoDB**

1. We do not need to design a schema for the DB, which saves us a lot of time.

2. No complex joins and no relationships among data.

3. Easy to scale.

4. Easy to set up and install.

5. Document query language it supports is simpler compared to SQL

6. It is very easy to store arrays and objects since MongoDB uses JSON format to store data.

7. Free to use with absolutely no cost to it.

8. Higher performance compared to any relational database.

9. No need to map app objects to database objects.

10. Uses internal memory for storage which enables faster access to the data.

**Implementation of the Micro-services**

**Authentication System**

This system receives the email and password from a user, encrypts them, stores them in its database and returns an encrypted token that allows them to access their respective systems as a whole. the system could have one or more admins and assigns each other system its admins as well. If the user tries to access a system that they’re unauthorized to access, then the system denies their request and prevents them from entering said system.

**The System’s Code**

**User.js**

This is the class in which we define how the user enters their information so that it could be encrypted and stored in the system’s database.

import bcrypt from 'bcrypt'

import Router from 'express'

import Database from '../db'

import tokenMiddleWare from '../middleware/token'

import checkUser from '../middlewares/checkUser'

const router = Router()

router.post('/', tokenMiddleWare, checkUser, createUser)

async function createUser(req, res, next) {

let { email, password, extraData } = req.body

if (!email || !password) {

return next({

error: {

input: 'One of the fields are empty',

code: 409

}

})

}

if (password.length < 4) {

return next({

error: {

password: 'Password is too short',

code: 409

}

})

}

email = String(email).toLowerCase()

const emailPattern = /^[a-z0-9.-\_]+@[a-z]+\.[a-z]{3}$/

const isValid = emailPattern.test(email)

if (!isValid)

return next({

error: {

email: 'Email is not in a valid format',

code: 409

}

})

const saltRounds = 10

const hash = await bcrypt.hash(password, saltRounds).then(hash => hash)

const user = {

email,

password: hash,

...extraData

}

const doc = Database.write({ tableName: 'User', object: user })

res.status(201).send(doc)

}

export default router

**Index.js**

This class finds and handles all sorts of errors that the system may face.

import bodyParser from 'body-parser';

import express from 'express';

import helmet from 'helmet';

import xss from 'xss-clean';

import tokenMiddleWare from './middlewares/token';

import tokens from './routes/tokens';

import user from './routes/user';

global.dev\_ENV = process.env.NODE\_ENV !== 'production'

var app = express()

// app.use(

// express.static(path.join(\_\_dirname, '../client/build'), { index: false })

// )

app.use(xss())

app.use(bodyParser.json())

app.use(bodyParser.urlencoded({ extended: false }))

app.use(helmet())

app.use(tokenMiddleWare)

app.use('/tokens', tokens)

app.use('/user', user)

// app.get('/\*', function (req, res) {

// res.sendFile(path.join(\_\_dirname, '../client/build', 'index.html'));

// });

// 404 handling

// =============================================================================

app.use(function (req, res, next) {

res.status(404).send({ error: 'not found' })

})

// Error handling

// =============================================================================

app.use(function (err, \_req, res, \_next) {

if (err.name === 'UnauthorizedError') {

res.send({ error: 'invalid token...' })

} else {

if (err.status) res.status(err.status)

else res.status(err.error.code)

if (!err.error || !err.error.code) {

return res.send('something bad happened')

}

res.send({ error: err.error })

}

})

var dev\_ENV = process.env.NODE\_ENV !== 'production'

var listener = app.listen(

dev\_ENV ? 3001 : process.env.PORT || 3001,

function () {

console.log(

(dev\_ENV ? 'Dev' : 'Prod') +

'Mode Listening on port ' +

listener.address().port

) //Listening on port 8888

}

)

**Db.js**

This class handles the system’s database storage, where all the information about the users is encrypted and stored.

import Promise from "bluebird";

import { promises as fs } from "fs";

import { MongoClient } from "mongodb";

import config from "./config";

let production = false; //process.env.NODE\_ENV === 'production';

const key = config.database;

// const dbName = production ? "blloc" : "blloc-staging";

const dbName = "HIS";

let db = null;

let connect = (cb) => {

console.log("Connecting to db");

MongoClient.connect(

key,

{

useNewUrlParser: true,

},

(err, database) => {

if (err) return console.log(err);

console.log("db connected");

db = database.db(

process.env.NODE\_ENV !== "production" ? "blloc-staging" : "blloc"

);

cb && cb();

}

);

};

export default class Database {

static databaseObj(cb) {

if (!db) return connect(() => cb(db));

return cb(db);

}

static async openDb() {

return MongoClient.connect(key, { useNewUrlParser: true }).then(

(dbc) => (db = dbc.db(dbName))

);

}

static async read({ tableName, query }) {

if (!db) await this.openDb();

return db.collection(tableName).find(query).toArray();

}

static async write({ tableName, object }) {

if (!db) await this.openDb();

console.log("DB: " + db.dbName);

console.log("Object to write: " + object);

return db.collection(tableName).insertOne(object, function (err) {

if (err) {

console.log("ops !! an error inserting");

console.log(err);

} else console.log("1 document inserted");

});

}

static async getOrders() {

return this.read({ tableName: "Orders", query: {} });

}

static async loadOffline() {

return fs.readFile("data.json");

}

}

export class AsyncDatabase {

static async update({ collection, query, update }) {

return new Promise(async (resolve, reject) => {

let updated = null;

if (!db) await AsyncDatabase.connect();

updated = db

.collection(collection)

.update(query, update, { upsert: false });

if (updated) resolve(updated);

else reject(updated);

});

}

static async findOneAndUpdate({ collection, query, update }) {

return new Promise(async (resolve, reject) => {

if (!db) await AsyncDatabase.connect();

db.collection(collection).findOneAndUpdate(

query,

update,

{ upsert: false, returnOriginal: false },

(err, result) => {

if (err) reject(err);

resolve(result.value);

}

);

});

}

static async connect() {

return new Promise((resolve, reject) => {

connect(resolve);

});

}

static async writeToDb({ data, table }) {

return new Promise(async (resolve, reject) => {

if (!db) await AsyncDatabase.connect();

db.collection(table).insertMany(data, (err, res) => {

if (err) reject(err);

else resolve("done");

});

});

}

static async findMany({ query, table }) {

return new Promise(async (resolve, reject) => {

if (!db) await Database.openDb();

db.collection(table)

.find(query)

.toArray((err, record) => {

if (err || !record) return reject("not found");

else resolve(record);

});

});

}

static async findDb({ query, table }) {

return new Promise(async (resolve, reject) => {

if (!db) await Database.openDb();

db.collection(table).findOne(query, (err, record) => {

if (err || !record) return reject("not found");

else resolve(record);

});

});

}

}

**Config.js**

This class is simply the database’s configuration and the technology used in constructing it, which as we discussed before is MongoDB.

export default {

database : "mongodb+srv://root:graduation2020@his.8pqkg.mongodb.net/HIS?retryWrites=true&w=majority",

}

**Middleware classes**

Naturally, this system uses middleware in order to check the users and return the encrypted token after their information is stored on the database.

**CheckUser.js**

This class checks the user and which systems are they allowed to access. If they try to access a system they are unauthorized to access, then the system denies them access and returns ‘Access Denied.’

const userResponses = {

AccessDenied: {

message: 'accessDenied',

code: 403

},

TokenCreated: {

data: {},

token: ''

},

User: {

\_id: '',

username: '',

email: '',

extraData: {},

token: ''

},

UsersList: [] // Array<User>

}

export default (req, res, next) => {

if (req.User !== null) {

next()

} else {

const AccessDenied = userResponses.AccessDenied

next({

error: {

accessDenied: AccessDenied.message,

code: AccessDenied.code

}

})

}

}

**Tokens.js**

This class returns the token that checks whether the information entered by the user is of the right format and length right before its storage.

import bcrypt from 'bcrypt'

import { Router } from 'express'

import jwt from 'jsonwebtoken'

import Database from '../db'

import keys from '../keys'

import checkUser from '../middlewares/checkUser'

import token from '../middlewares/token'

const router = Router()

router.post('/', loginUser)

router.get('/', token, checkUser, (req, res) => {

res.status(200).send(req.User)

})

async function loginUser(req, res, next) {

var { email, password } = req.body

if (!email || !password) {

next({

error: {

input: 'One or more fields are empty',

code: 409

}

})

}

if (password.length < 4) {

next({

error: {

password: 'Password is too short',

code: 409

}

})

}

email = String(email).toLowerCase()

const emailPattern = /^[a-z0-9.-\_]+@[a-z]+\.[a-z]{3}$/

const isValid = emailPattern.test(email)

if (!isValid)

next({

error: {

email: 'Email is not in a valid format',

code: 409

}

})

const user = await Database.read({

tableName: 'User',

query: { email }

})

if (!user || user.length === 0) {

return next({

error: {

user: 'Wrong Email or Password',

code: 409

}

})

}

bcrypt.compare(password, user[0].password).then(function (isMatching) {

if (isMatching) {

const payload = {

id: user[0].\_id,

email: user[0].email,

data: user[0].extraData

}

const token = jwt.sign(payload, keys.privateKey, { expiresIn: "24h" })

res.status(201).send({

payload,

token

})

} else {

next({

error: {

user: 'Wrong Email or Password',

code: 409

}

})

}

})

}

export default router