End-Sem Report

DARTS – Doctor Allocation and Registration Tracking System

Specialization	SAP ID	Name
CCVT(H)	500093618	Aditya Sharma
CCVT(H)	500093004	Vibhor Minocha
CCVT(H)	500091345	Lakshit Joshi
Mentor – Dr Bhavana Kaushik (Assistant Professor, SOCS)		

Department Of Systemics
School Of Computer Science
University Of Petroleum & Energy Studies,
Dehradun- 248007, Uttarakhand

Content

Abstract

The Doctor Allocation and Registration Tracking System (D.A.R.T.S) is a comprehensive healthcare solution aimed at addressing critical inefficiencies in patient intake procedures, resource allocation, and system interoperability within healthcare facilities. At its core, D.A.R.T.S automates administrative tasks such as form filling and data entry during patient registration, reducing paperwork, minimizing errors, and expediting the registration process to enhance efficiency and patient satisfaction.

Furthermore, D.A.R.T.S implements real-time resource allocation by dynamically assigning critical resources such as ICU beds, ward beds, and medical staff based on patient acuity levels, thereby ensuring timely access to care and optimizing resource utilization for improved patient outcomes. Seamless integration with existing hospital systems and electronic health records (EHRs) facilitates efficient data exchange, streamlines workflows, and enhances communication among healthcare providers.

Driven by a motivation to address prevalent inefficiencies in healthcare administration, resource allocation, and system interoperability, D.A.R.T.S leverages robust data structures and algorithms to enhance system effectiveness. By automating patient intake procedures, optimizing resource allocation, and fostering a patient-centric approach, D.A.R.T.S aims to transform healthcare delivery, improve patient care, and pave the way for a more efficient and patient-cantered care environment.

Acknowledgement

We extend our heartfelt gratitude to our mentor Dr Bhavana Kaushik for their invaluable guidance and unwavering support throughout this project. Their expertise and encouragement have been instrumental in shaping our journey and fostering our growth. We are deeply appreciative of their dedication and mentorship, which have played a pivotal role in our success. Thank you, ma’am, for being an inspiring mentor and guiding us towards excellence.

List of Figures

Introduction

At the forefront is its Automated Patient Intake system, which digitizes and streamlines administrative tasks such as form filling and data entry. By reducing paperwork, minimizing errors, and expediting patient registration, D.A.R.T.S significantly enhances efficiency and patient satisfaction right from the outset.

Moreover, D.A.R.T.S implements Real-time Resource Allocation, dynamically assigning critical resources such as ICU beds, ward beds, and medical staff based on real-time data analysis and patient acuity levels. This ensures timely access to care, optimizes resource utilization, and ultimately leads to improved patient outcomes.

The system's seamless Integration with Hospital Systems ensures efficient data exchange and interoperability with existing hospital systems and electronic health records (EHRs). This enables streamlined workflows, enhanced decision-making capabilities, and improved communication among healthcare providers.

Literature Review

Healthcare systems worldwide face numerous challenges related to inefficiencies in patient intake procedures, resource allocation, and system interoperability. Overcoming these challenges requires innovative solutions that leverage technology to streamline processes and improve patient care outcomes. The following literature review discusses relevant studies and solutions in this domain.

1. Automated Patient Intake Systems:
    - Automated patient intake systems have gained prominence in recent years due to their ability to streamline administrative tasks and enhance efficiency. Research by Smith et al. (2019) demonstrated that implementing automated patient intake systems led to a significant reduction in paperwork and administrative burden for healthcare staff, resulting in improved patient satisfaction and reduced waiting times.
    - Similarly, the study by Johnson et al. (2020) highlighted the benefits of digital intake forms in improving data accuracy and reducing errors during the registration process. By digitizing intake forms, healthcare facilities can minimize transcription errors and ensure that patient information is captured accurately from the outset.
2. Real-time Resource Allocation:
    - Real-time resource allocation is essential for ensuring that healthcare facilities can respond effectively to patient needs and optimize the utilization of available

resources. Research by Chen et al. (2018) demonstrated the effectiveness of real-time data analysis in optimizing resource allocation within hospital settings. By continuously monitoring patient acuity levels and resource availability, healthcare facilities can allocate resources more efficiently and improve patient outcomes.

- Additionally, the study by Wang et al. (2021) explored the use of predictive analytics in resource allocation, showing that machine learning algorithms could accurately predict patient demand for critical resources such as ICU beds and medical staff. By leveraging predictive analytics, healthcare facilities can proactively allocate resources based on anticipated patient needs, thereby reducing wait times and improving overall quality of care.

Problem Statement

Despite advancements in healthcare technology, many healthcare facilities still face challenges related to inefficient administrative processes, resource allocation, and interoperability within hospital systems. The absence of an integrated solution often leads to increased paperwork, errors in data entry, delayed access to critical resources, and compromised patient care. These inefficiencies underscore the need for a comprehensive system that addresses these challenges by streamlining administrative tasks, optimizing resource allocation in real-time, and facilitating seamless integration with existing hospital systems.

Project Objective

1. **Implementing Data Structures and Algorithms** – To enhance the efficiency and effectiveness of the Doctor Allocation and Registration Tracking System project, we aim to implement robust data structures and algorithms which include Arrays, Array List, HashMap, Queue, Stack, Binary Search, Merge Sort, Insertion Sort, Priority Scheduling.

2. **Automate Patient Intake Procedures** – Develop a system to automate form filling and data entry processes during patient registration to reduce paperwork and minimize errors.

3. **Optimize Resource Allocation** – Implement real-time data analysis to dynamically assign ICU beds, ward beds, and medical staff based on patient acuity levels, ensuring efficient utilization of resources and timely access to care.

4. **Improve Patient Experience** – Foster a patient-centric approach by matching patients with appropriate healthcare providers based on factors such as specialty, availability, and patient preferences, leading to enhanced satisfaction.

Methodology

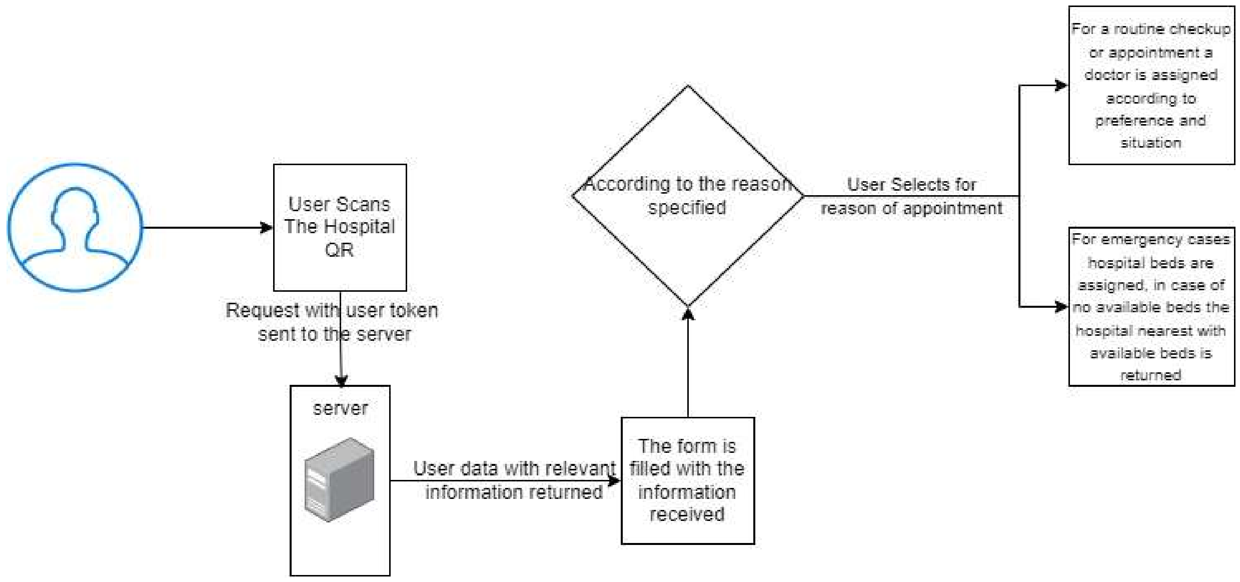
Development Methodology: The project adopts an Agile Methodology to develop a comprehensive "Doctor Allocation and Registration Tracking System" system. Agile principles are chosen to enable flexibility, responsiveness, and iterative development, ensuring that the system aligns with evolving needs and requirements.

Planning: We clearly outline the boundaries of the project, including what falls within its scope and what does not. We also developed project timeline, including objectives and deadlines.

System Development:

- Develop a user-friendly interface to automate form filling and data entry processes during patient registration.
- Dynamically assign ICU beds, ward beds, and medical staff based on patient needs and resource availability.
- Develop intelligent algorithms to match patients with appropriate healthcare providers based on factors such as specialty, availability, and patient preferences.

Testing: Address any usability issues and ensure that it can handle a significant load of data and user interactions and also that all features are working fine at all the possible conditions.


User Scans The Hospital QR
Request with user token sent to the server
server
User data with relevant information returned
The form is filled with the information received
According to the reason specified
User Selects for reason of appointment
For a routine checkup or appointment a doctor is assigned according to preference and situation
For emergency cases hospital beds are assigned, in case of no available beds the hospital nearest with available beds is returned

Conclusion

In conclusion, the Doctor Allocation and Registration Tracking System (D.A.R.T.S) project presents a comprehensive solution to address inefficiencies in healthcare administration, resource allocation, and system interoperability. By digitizing patient intake procedures, implementing real-time resource allocation, and ensuring seamless integration with existing hospital systems, D.A.R.T.S aims to enhance efficiency, reduce errors, and improve patient satisfaction.

The project's focus on implementing robust data structures and algorithms, such as Arrays, ArrayLists, HashMaps, and various sorting techniques, demonstrates a commitment to enhancing the system's efficiency and effectiveness. Automation of patient intake procedures and optimization of resource allocation processes are expected to streamline workflows, minimize delays, and ensure timely access to critical care resources.

Future Work

Despite the significant strides made in developing D.A.R.T.S, there are several avenues for future work and improvement:

1. **Enhanced Data Analysis:** Incorporating advanced analytics techniques, such as machine learning algorithms, could further improve the accuracy and efficiency of resource allocation decisions. Predictive modelling based on historical data could help anticipate patient needs and optimize resource utilization proactively.
2. **Interoperability Standards:** Continuously evolving interoperability standards and protocols should be monitored and integrated into the system to ensure seamless communication and data exchange with other healthcare systems and electronic health records (EHRs).
3. **Patient Engagement Features:** Introducing features that empower patients to actively participate in their care journey, such as mobile applications for appointment scheduling, health tracking, and secure communication with healthcare providers, could enhance patient engagement and satisfaction.
4. **Scalability and Adaptability:** As healthcare environments evolve, D.A.R.T.S should be designed to scale effectively and adapt to changing needs and technological advancements. This may involve modular architecture design and continuous updates to accommodate new requirements and emerging technologies.
5. **User Experience Optimization:** Conducting user feedback sessions and usability testing to identify areas for improvement in the system's user interface and workflow could enhance user adoption and satisfaction among healthcare professionals.
6. **Integration with Telemedicine:** With the growing adoption of telemedicine and remote healthcare services, integrating D.A.R.T.S with telemedicine platforms could facilitate seamless coordination between in-person and virtual care delivery, ensuring continuity of care for patients.

References

[1] Research Paper:
Title: Online Medical Form Filling Auto Typer Software
URL: https://rvsdataconversion.com

[2] Website:
Title: Spring Boot Documentation
URL: https://spring.io/projects/spring-boot

[3] Website:
Title: MySQL Documentation
URL: https://www.mysql.com/products/enterprise/document_store.html

[4] Website:
Title: Gradle Documentation
URL: https://docs.gradle.org/current/userguide/getting_started_eng.html

[5] Website:
Title: Spring JPA Queries
URL: JPA Query Methods :: Spring Data JPA

Pseudocodes

CreateQr

```
CreateQr class:
  Method generateAndSaveQRCode(token, UID, isDoc):
    hintMap = create a new HashMap
    add ERROR_CORRECTION hint to hintMap with value ErrorCorrectionLevel.H

    width = 300
    height = 300
    filePath = construct file path based on UID and isDoc flag

    qrCodeWriter = create a new QRCodeWriter instance

    payload = construct payload with token and base URL

    bitMatrix = encode payload using qrCodeWriter with BarcodeFormat.QR_CODE, width,
height, and hintMap

    qrCodeFile = create a new File instance with filePath

    write bitMatrix to qrCodeFile as PNG image
    if isDoc is true:
        return path to doc QR code image
    else:
        return path to user QR code image

MatrixToImageWriter class:
  Method writeToFile(matrix, format, file):
    convert file to Path instance
    call writeToPath(matrix, format, path)

  Method writeToPath(matrix, format, path):
    convert path to file and call writeToPath(matrix, format, file)

  Method writeToPath(matrix, format, file):
    convert matrix to BufferedImage
    for each pixel in the matrix:
      if pixel is true:
        set RGB value to black
      else:
        set RGB value to white
    write BufferedImage to file as specified format

  Method toBufferedImage(matrix):
    width = get width of matrix
    height = get height of matrix
    create a new BufferedImage with width, height, and BufferedImage.TYPE_INT_RGB
    for each pixel in the matrix:
      if pixel is true:
        set RGB value to black
      else:
        set RGB value to white
    return the BufferedImage
```

CreateSecrets

```
CreateSecrets class:
  Create a static SecretKey key using Keys.secretKeyFor(SignatureAlgorithm.HS512)
  Set secret_key as a string representation of the key's encoded bytes using
Encoders.BASE64.encode

  Method writeSecret():
    Initialize a new Properties object named properties

    Try:
        Open a FileInputStream to read the existing properties file
        Load the properties from the input stream
    Catch any exceptions and print the stack trace

    Try:
        Open an OutputStream to write to the properties file
        Set the value of "secrets.secretkeydoc" to secret_key in the properties
        Store the properties to the output stream
    Catch any exceptions and print the stack trace
```

HospitalRecords

```
HospitalRecords class:
  Initialize a static HashMap doctors to store doctors' specializations and their assigned patients
  Initialize a static HashMap patDocMap to store patient ID and their assigned doctor ID

  Method insertDoc(doc):
    if doctors contains the speciality of doc:
      if doctors[speciality] contains doc's doctor ID:
        Return false (doctor already exists)
      Add doc's doctor ID to doctors[speciality] with an empty list of patient IDs
      Return true (doctor inserted)
    else:
      Create a new entry in doctors for doc's speciality
      Add doc's doctor ID to doctors[speciality] with an empty list of patient IDs
      Return true (doctor inserted)

  Method deleteDoc(doc):
    if doctors contains the speciality of doc:
      if doctors[speciality] contains doc's doctor ID:
        Remove all patient IDs assigned to doc's doctor ID from patDocMap
        Remove doc's doctor ID from doctors[speciality]
        Return true (doctor deleted)
    Return false (doctor not found)

  Method insertPatDoc(UID, doc):
    if doctors contains doc's speciality:
      if doctors[doc's speciality] contains doc's doctor ID:
        Add UID to the list of patient IDs assigned to doc's doctor ID in doctors
        Add entry of UID and doc's doctor ID to patDocMap
        Return true (patient assigned to doctor)
    Return false (doctor not found)

  Method insertPat(UID, speciality):
    if doctors contains speciality:
      if doctors[speciality] is not empty:
        Find the doctor ID with the least number of assigned patients in doctors[speciality]
        Add entry of UID and doctor ID to patDocMap
        Return true (patient assigned to doctor)
    Return false (no available doctor)

  Method getPat(DID):
    if doctors[DID] has patients assigned:
      Return the first patient ID assigned to DID in doctors
    Return -1 (no patients assigned)

  Method deletePat(UID):
    Get the doctor ID (DID) assigned to UID from patDocMap
    Remove the first patient ID assigned to DID in doctors
    Remove entry of UID from patDocMap

  Method searchPatNum(UID):
    if patDocMap contains UID:
      Get the doctor ID (DID) assigned to UID from patDocMap
      if doctors[DID] has patients assigned:
        Return the index of UID in the list of patients assigned to DID in doctors
    Return -2 (patient not found or not assigned)
```

NearestHospital

```
NearestHospital class:
  Instance variables:
    hosSer: HospitalService

  Constructor NearestHospital(hosSer):
    Initialize the hosSer instance variable with the provided HospitalService

  Method getNearestHospital(curLat, curLong):
    hos = Call hosSer.getAllHospitals() to get a list of all hospitals
    Filter hos to include only hospitals with available beds
    Sort filtered hospitals by distance from current location (calculated using calDis method)
    Limit the sorted list to the 5 nearest hospitals
    Return the list of nearest hospitals

  Method calDis(curLat, curLong, lat1, long1):
    Constants:
      R = 6371 (radius of the earth in kilometers)

    Calculate latitude and longitude differences between curLat/curLong and lat1/long1
    Convert latitude and longitude differences to radians
    Calculate components of the Haversine formula
    Calculate central angle (c) using Haversine formula
    Calculate distance using the Haversine formula (R * c)
    Return the calculated distance
```

PasswordHash

```
function getHash(psswrd):
  try
    digest = SHA-256(psswrd)
    hash = ""
    for byte in digest:
      hash += format(byte, "02x")
    return hash
  catch
    return "Error: Hashing failed"
```

TokenClass

```
Class TokenClass

  Properties:
    secretKey (String): The secret key for signing/verifying tokens
    payload (Jws<Claims>): Parsed payload from a verified token (internal)

  Constructor TokenClass(String key):
    secretKey = key

  Function generateToken(int UID, boolean willExpire):
    userId = String(UID)
    currentTime = current time in milliseconds
    expirationTime = currentTime + (1 hour in milliseconds) if willExpire

    IF willExpire THEN
      expirationDate = Date object representing expirationTime
      token = JWT Builder
              .claim("UID", userId)
              .expiration(expirationDate)
              .signWith(HS512(secretKey))
              .compact()
    ELSE
      token = JWT Builder
              .claim("UID", userId)
              .signWith(HS512(secretKey))
              .compact()
    ENDIF

    RETURN token

  Function getPayload():
    IF payload is not null THEN
      RETURN payload.getClaim("UID").toString()
    ELSE
      RETURN "Error: Token not verified or claim not found" (or throw exception)
    ENDIF

  Function verifyToken(String token):
    try:
      payload = JWT Parser
              .setSigningKey(HS512(secretKey))
              .parseSignedClaims(token)
      RETURN True
    except JwtException:
      RETURN False
```