ITMB
COC252
F011321

EHR System Modernization in the Republic of Moldova

by

Calin Corcimaru

Supervisor: Dr. Georgina Cosma

Department of Computer Science
Loughborough University

May 2025

Abstract

Abstract to be added

Acknowledgements

Acknowledgements to be added

Contents

Chapter 1

Introduction

1.1 Background

The Republic of Moldova is a small country in Eastern Europe that borders Romania and Ukraine, with a current population of 2.4 million people (National Bureau of Statistics of the Republic of Moldova, 2024). Since its independence in 1991, Moldova has faced a number of challenges, including political instability, corruption, and economic difficulties which have left Moldova as one of the poorest countries in Europe (BBC, 2024).

Despite these challenges, Moldova has made significant progress in its digital transformation efforts, with the government launching a number of initiatives to modernize its public services and improve the quality of life for its citizens (E-Governance Agency, n.d.[a]). An example is the Citizen's Government Portal (MCabinet), which allows citizens to access personal information such as 'valid identity documents, social contributions and benefits, own properties, information about the family doctor and the health institution where the person is registered, tax payments and other information about the citizen-government relationship' (E-Governance Agency, n.d.[b]).

To continue supporting the existing transformation initiatives, Moldova's Cabinet of Ministers has recently approved the 'Digital Transformation Strategy of the Republic of Moldova for 2023-2030', which aims to transform the country into a digital society by 2030, with the ultimate goal of having 'all public services available in a digitalized format' (United Nations Development Programme, 2023).

1.2 Problem Statement

The healthcare sector in Moldova has also seen some transformations, with the introduction of a new electronic health record system (EHR) in 15 hospitals across the country in 2017, called 'Sistemul informațional automatizat „Asistența Medicală Spitalicească" (SIA AMS)' (Ciurcă, 2021). While the system has been successful in helping doctors access patient information more efficiently such as medical history, examinations, test results, and prescriptions, the system hasn't been updated since its inception in 2017 and there are still challenges that need to be addressed in 2024.

The main challenge with the current system lies in the user experience (UX) – SIA AMS feels old and isn't user-friendly, with a clunky interface that is difficult to navigate, not adhering to modern accesibility standards and only accessible via Internet Explorer or legacy version of Microsoft Edge, with no support for other browsers or devices (Ciurcă, 2021).

Another big challenge with the system is its lack of interoperability within public and private medical institutions due to a lack of a nationally-wide integrated system – each hospital and clinic have their own, siloed, information system that contains the patient information, with no communication being made between systems in different hospitals (Ciurcă, 2021).

Finally, due to the current economic situation in Moldova, the government has not allocated any funds to upgrade the current or develop new systems, and the hospitals and clinics that use the system do not have the resources to update it themselves.

1.3 The Client

The client, "Nicolae Testemiteanu" State University of Medicine and Pharmacy in Moldova (USMF), is a public university in Chisinau, Moldova, that offers a range of medical programs, including medicine, dentistry and pharmacy (USMF, 2023). Many of the faculty at USMF are also practicing doctors at hospitals and clinics across Moldova, and have first-hand experience with the current IT systems used in both public and private medical institutions. The USMF faculty members that the student will be interacting with during the project are part of an innovation team that researches potential opportunities to improve the healthcare sector in Moldova through the use of technology. As such, the client has expressed a need for a prototype that can act as a proof of concept for a modern system that could either replace or augment the current system in Moldova.

1.4 Project Objectives

This project aims to initially conduct some research on the current situation of the IT systems used in the healthcare sector in Moldova by interviewing several stakeholders from various healthcare-related institutions. Afterwards, the project will conduct a literature review on the most appropriate technologies and methodologies for developing a modernized EHR system, and an analysis of existing EHR systems to identify their existing functionality. Finally, based on the information gathered, the project will focus on designing and developing a working prototype, based on the requirements gathered and the feasibility of the chosen solution for the Moldovan healthcare sector. The student's hope is that the solution can then be used as a proof of concept to secure funding for a full-scale implementation of the new system in Moldova by the relevant authorities, such as the Ministry of Health.

As such, the objectives of the project are as follows:

1. Identify 2 to 4 stakeholders from various perspectives, such as healthcare institutions in Moldova and patients, that can provide insights into the current IT systems used in the healthcare sector in Moldova.
2. Conduct interviews with the identified stakeholders to gather information on the current IT systems used in the healthcare sector in Moldova.
3. Carry out a literature review to research the most appropriate technologies (frontend, backend and database) and project management methodologies for developing a modernized EHR system.
4. Explore at least 2 existing EHR systems and identify their strengths and weaknesses.
5. Design and develop a working web or mobile app for an EHR system based on the requirements gathered and the feasibility of the chosen solution for the Moldovan healthcare sector.
6. Offer the client the prototype to be used as a proof of concept to secure funding for a full-scale implementation of the new system in Moldova.

Chapter 2

Research and Requirements

To gain a more complete understanding of the current situation in Moldova, the existing problems and possible needs of the people involved, it is important to start with an analysis to identify the possible key stakeholders for this project. As previously mentioned in the literature review, a diverse group of stakeholders is essential to ensure that the currrent situation is reviewed from multiple perspectives.

Afterwards, the next step is to utilise the chosen stakeholders to gather as much information as possible from various perspective to ensure that the project is aligned with the needs of both patients and healthcare professionals in Moldova and solves the existing problems.

2.1 Stakeholder Analysis

The student has identified the following stakeholders for the project:

- A doctor working at the Republican Hospital - the main stakeholder, who will be providing guidance and feedback throughout the project on the perspective of the end users, the doctors and nurses who will be using the EHR system.
- The Vice-Director of CNAM (National Health Insurance Company) - the person who will be providing feedback from the perspective of the health insurance company.
- Senior IT staff member at CNAM - a person who also has previously worked within the Ministry of Health, and who will be providing feedback on the technical aspects of the integration between the EHR system and the health insurance system.
- Department head at the Republican Hospital - a person who will be providing feedback

on the administrative aspects of the EHR system.

- Patients - both of public and private healthcare institutions, who will be providing feedback on the usability and accessibility of an EHR system prototype.

These stakeholders have been identified so that they can provide a bigger picture on the needs and requirements of the project, and to ensure that the project is aligned with the needs of the healthcare industry in Moldova from both the provider and patient perspective.

The stakeholders have also been placed the stakeholder influence-interest grid, to help the student understand the level of influence and interest that each stakeholder has in the project:

- The doctor working at the Republican Hospital - low influence, high interest
- The Vice-Director of CNAM - high influence, high interest
- Senior IT staff member at CNAM - high influence, low interest
- Department head at the Republican Hospital - high influence, low interest
- Patient - low influence, high interest

2.1.1 Current situation analysis

Following the stakeholder analysis, the student has conducted several exploratory interviews with the chosen people to provide insights into the current issues with the IT systems used in the healthcare sector. After the conclusion of the interviews, several main themes for problems and potential solutions have been identified:

1. Current system is outdated, not user friendly and only accessible via Internet Explorer or legacy version of Microsoft Edge. A potential solution is to develop a new, modernized version of the existing system (thus retaining the core functionality) that is accessible via modern browsers and devices and can be augmented with additional features if necessary.
2. Lack of interoperability between medical institutions due to a lack of a nationally-wide integrated system. A potential solution is to create a new systems where patients can upload their own medical records (such as lab tests, previous medical history, etc) and share them with any medical practitioner, regardless of the institution they work at.
3. Some systems are not digitalized at all, and still rely on paper-based records or very rudimentary data structures, such as the transplant registry. A potential solution is

a the creation of a digitalized system, as is in the case of the transplant registry, that can be accessed by any medical practitioner in Moldova.

After analysing the current issues and potential solutions, the student has determined that the solutions for issues #1 and #3 are too complex, as they require an overhaul of the system and integration with other existing systems. As such, the student has decided to focus on issue #2, as it is the most feasible and can be implemented within the timeframe of the project.

After the decision has been made, some changes to the stakeholders have been made. The Department head at the Republican Hospital and Senior IT staff member at CNAM have been removed from the list of stakeholders, as their feedback is no longer relevant to the project. Instead, the focus was shifted onto having more patients as stakeholders, as they will now be the main users of the system.

The new stakeholders are as follows:

- Doctor working at the Republican Hospital
- Doctor who previously worked at the Republican Hospital
- The Vice-Director of CNAM
- 2 Patients that are using both public and private healthcare institutions
- Doctor working a private healthcare institution

After the new stakeholders were identified, additional interviews were conducted to focus on the requirements for the chosen solution. The requirements have been gathered and documented in the next section.

2.2 Requirements

The student has gathered the following requirements for the project:

2.2.1 Non-functional requirements

1. The system must be accessible on all modern desktop and mobile based browsers.
2. The system must be accessible from any location by using an internet connection.
3. When shared with the doctor via a link, the system must load within 3 seconds when accessed via a desktop browser.
4. The system must provide a secure login mechanism for patients via MFA or e-signature.

5. The system should allow patients to upload their own medical records in a variety of formats (PDF, DOC, etc).
6. When shared with the doctor via a link, the system should secure the data with a unique token or PIN that expires after the specified time frame.
7. The system could be accessible on all modern mobile devices via a mobile application.

2.2.2 Patient history requirements - doctor perspective

1. The system must provide an overview of the patient history through 3 main sections: personal information, lab tests and doctor consultations.
2. The system must allow the doctor to only view the patient's history, not edit it.
3. The system must allow the doctor to view blood tests in a graphical format.
4. The system must allow the doctor to view blood tests in a numerical, tabular format.
5. The system must allow the doctor to view the patient's history in a chronological order.
6. The system must display the doctor consultation and every lab test,except for blood tests, in a free text or document format.
7. For blood test results, the system should display the normal range values for each test.

2.2.3 Patient personal data requirements

1. The system must allow the patient to include their personal information, such as name, date of birth, address, allergies, etc.
2. The system must allow the patient to include their insurance information.
3. The system must allow the patient to enter vitals information, such as height, weight, blood pressure, etc.
4. When multiple vital entries are made, the system should display a historical graph of the patient's vitals.

2.2.4 Patient medication

1. The system must allow the patient to enter their current medication.
2. The system must allow the patient to enter their past medication.
3. The system must allow the patient to enter the dosage and frequency of the medication.

4. The system must allow the patient to enter the reason for taking the medication.
5. The system must allow the patient to enter the start and end date of the medication.
6. The system must allow the patient to enter the doctor who prescribed the medication.
7. When adding medication, the system should have 2 options: add a simplified version of the medication or add a detailed version of the medication.
8. When choosing the simplified version, the system should allow the patient to just add the name, dosage and duration of the medication.
9. When choosing the detailed version, the system should allow the patient to add the name, dosage, duration, reason for taking the medication, doctor who prescribed the medication, and any additional notes.
10. After entering the medication, the system could allow the patient to track the medication intake.
11. The system could send reminders to the patient to take the medication by using push notifications.

2.2.5 Patient shareable link

1. The system must allow the patient to generate a shareable link to their medical records.
2. When creating the shareable link, the system must allow the patient to set an expiration date for the link.
3. When creating the shareable link, the system should allow the patient to set an access PIN or token for the link.
4. When create the shareable link, the system should allow the patient to select which records to share with the doctor.

Chapter 3

Literature Review

This chapter will provide a review of the existing literature, which will be used guide the student in their planning and development efforts of the project.

As such, it will be covering the following areas:

- Software development methodologies
- Requirement gathering
- System design
- Tech stack
- Machine Learning models
- Gap Analysis

3.1 Software development methodologies

3.1.1 Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a process used to guide the development of software applications or systems (Ruparelia, 2010). The SDLC consists of multiple phases, each with its own set of activities and deliverables. Yas, Alazzawi, and Rahmatullah (2023) outline the phases of the SDLC as following:

1. Requirement gathering and analysis phase - This phase involves gathering and analyzing the requirements of the software to be developed. These requirements are

gathered from the project stakeholders and saved in a specific document. Based on the requirements gathered, a development plan is created and a feasibility study is conducted.

2. Design phase - This phase involves representing the previously gathered requirements in a project design written in a more technical manner, that will later guide the developers to create and implement the software.
3. Implementation phase - This phase involves the actual development of the software. Additionally, some smaller unit tests may occur during this phase as parts of the software are developed.
4. Testing phase - This phase focuses on testing the software to ensure that it meets the requirements and is free of bugs. This phase may involve multiple types of testing, such as unit testing, integration testing, and system testing. Luo (2001) describes the different types of tests as following:
    - Unit testing - This type of test is done on the lowest level of the software, testing individual units or components of the software.
    - Integration testing - This type of test is performed on two or more units combined together, usually focusing on the interfaces between these components.
    - System testing - This type of test focuses on the 'end-to-end quality of the entire system', testing it as a whole based on the system requirement specification.
5. Maintenance phase - This phase involves the deployment and maintenance of the software. Additionally, this phase may include user acceptance testing, where the software is handed over to the end-users to ensure that it meets their needs (Luo, 2001).

3.1.2 SDLC Models

The literature describes several SDLC models that have been used in the development of software applications. Ruparelia (2010) and Yas, Alazzawi, and Rahmatullah (2023) outline the most common SDLC models:

- Waterfall Model
- V Model
- Spiral Model
- Iterative Model

- Agile model

Due to the nature of the project being different to traditional software development projects, and the student's familiarity with only Waterfall and Agile models, the next sections will only focus on these two models.

Waterfall Model

The Waterfall Model is probably the most well-known SDLC model. Waterfall is a linear model, where the development process is divided into distinct, sequential phases that follow the SDLC. As such, each phase must be completed before the next phase can begin.

The Waterfall Model's strengths lie in its simplicty of use, ease of understanding and providing a structured approach to a project (Alshamrani and Bahattab, 2015). An additional stregth of the Waterfall model that the authors note is its extensive documentation and planning, which is done in the early stages of a project, but also maintened throughout the project's lifecycle. These two factors also help minimize the overhead that comes with planning and management of a project, which in the case of Waterfall is done in the early stages of the project.

However, the Waterfall model is not perfect. One of its main weaknesses, mentioned by Alshamrani and Bahattab (2015), is its lack of flexibility in regards to change of requirements. As such, once the project leaves the requirements analysis or design phase, it may be difficult to make any changes to the project deliverable. Thus, this model is not suitable for projects where the requirements are not well understood or are likely to change. Finally, the deliverable is only available at the end of the project, so the end-users are unable to see the final product until the end of the project, nor can they provide any feedback during its development (Alshamrani and Bahattab, 2015).

Agile Model

Another well-known SDLC model is the Agile model. Taking its roots from the Agile Manifesto, it describes a different way of developing software from the Waterfall model, with a focus on:

> *Individuals and interactions over processes and tools,*
> *Working software over comprehensive documentation,*
> *Customer collaboration over contract negotiation,*
> *Responding to change over following a plan* (Cunningham, 2001).

The Agile model focuses on the ideas that requirements are not always well-known or cannot be predicted, accepting that change is inevitable and emphasis should be put on being

able to accommodate any changes that may arise (Sunner, 2016). Similarly, as the author mentions, the focus of this methodology is on continuous delivery of software and value to the customer. As such, it is integral for an Agile project to have a close customer involvement in the development process, to ensure that constant feedback is received.

Agile does come with its own drawbacks. One of the main issues with Agile is its lack of documentation and formal planning, especially in the early stages of the project (Ruparelia, 2010). Consequently, the Agile methodology may not be suitable for large scale projects, where extensive documentation and planning are required (Sunner, 2016; Yas, Alazzawi, and Rahmatullah, 2023). Similarly, the Agile model may not be suitable for projects where the requirements are well understood, unlikely to change or where there may be strict regulations that guide how the project should be developed. Finally, Yas, Alazzawi, and Rahmatullah (2023) also mention that unfamiliarity with Agile frameworks could also be an impeding factor in the success of Agile projects, as the staff may not be familiar with Agile and require extensive training beforehand.

Agile has multiple frameworks that can be used to guide the development process, such as Scrum, Kanban, Lean, and Extreme Programming (XP). Alqudah and Razali (2018) mention that Scrum is the most widely used Agile framework in software development. As such, the next sections will focus on Scrum and Kanban, due to their popularity and student's familiarity with these frameworks.

Scrum is an Agile framework, with its core idea being the division of the project into smaller, manageable parts called sprints. Each sprint usually lasts between 2-4 weeks and each sprint focuses on delivering value to the customer through working software features and close communication with client stakeholders (Alqudah and Razali, 2018; Sunner, 2016). Scrum employs a set of artifacts and ceremonies, that are used to guide the development process, such as the sprint and product backlog, daily scrums, backlog grooming/prioritization, and many more. Scrum also has a set of team roles, such as the Scrum Master, Product Owner, and the Development Team, that are responsible for the development process (Alqudah and Razali, 2018).

On the other hand, Kanban is not as prescriptive as Scrum. Kanban focuses on visualizing the workflow of the project through a visual board with columns, cards and swimlanes. Kanban's main goal is to limit the work in progress through column WIP limits and a pull system to maximize the flow of work through the system (Sunner, 2016). Kanban does not have specific roles or artifacts, but instead focuses on the continuous delivery of value to the customer through smaller batching and daily prioritizations (Alqudah and Razali, 2018).

Finally, there is also a possibility of combining these two frameworks into one, called Scrum-

ban. Nowadays, the use of Scrum and Kanban together is becoming quite popular, with many teams employing both frameworks in their projects. As Alqudah and Razali (2018) mention, it can be quite beneficial as it allows the team to 'adopt the appropriate practices of both methods based on different situations to meet their needs'. The authors emphasize that the team members need to understand which elements of either framework bring value to the project and adapt them accordingly.

3.1.3 A hybrid approach

For many years, Waterfall (or the traditional approach) has been the most widely used model in software development projects, with Agile approaches gaining a lot of popularity in the recent years (Gemino, Reich, and Serrador, 2021). However, the authors note that a hybrid approach has also been emerging, being described as an approach that 'combines methodologies and practices from more than one project management approach'. Results from surveys cited by articles show that the hybrid approach has been used by over 50% of respondents (Gemino, Reich, and Serrador, 2021; Prenner, Unger-Windeler, and Schneider, 2020). The most common combinations that form this hybrid approach are 'Scrum, Iterative Development, Kanban, Waterfall and DevOps', with the approach that combines Waterfall and Scrum being the most popular (Prenner, Unger-Windeler, and Schneider, 2020).

An interesting finding is that even pure Agile approaches have been found to rarely exist, with hybrid Agile approaches being in reality most Agile implementations (Gemino, Reich, and Serrador, 2021). This may be due to multiple reasons, some of which may include regulations or safety standards in areas such as healthcare or defense systems documentation requirements or even time constraints. Thus, only the development part is usually done in an 'Agile way' - with the rest of the project using the traditional approach as a 'backbone' (Prenner, Unger-Windeler, and Schneider, 2020).

So how would a hybrid approach work versus Waterfall or Agile? Prenner, Unger-Windeler, and Schneider (2020) explain that the hybrid approach uses Waterfall as its main methodology - still retaining the 6 general phases that were mentioned above. The authors note that the approach begins with the classic requirements analysis phase, where the general projects requirements are analysed on a high level, which can include system specifications, risks, resourcing and project scope/budget. This is followed by a design phase, where initial designs and diagrams are created and decisions regarding technology and tools are made. Next, the previously gathered requirements are broken down in smaller pieces (like User Stories or Epics) and transferred to Agile-specific artifacts such as the Product Backlog, that will be used in the next phase. Following this, the development phase starts, usually following the Scrum framework that was described above. Finally, the project ends with a

testing phase, where system testing is done to ensure all components work well, followed by a operations or maintenance phase, where the software is deployed and maintenance support is offered.

Gemino, Reich, and Serrador (2021) note that projects using either Agile, traditional or hybrid approach show similar levels of success in terms of budget, time and quality. However, the authors have found that agile and hybrid approaches perform much better on the customer satisfaction metric than the traditional counterpart.

3.2 Requirements gathering

As noted in the previous section, the requirements gathering phase is the first step in the software development process, whether it is done in a traditional, Waterfall, approach or in an Agile approach. As described by Young (2002), a requirement is a 'necessary attribute in a system... that identifies a capability, characteristic, or quality factor of a system in order for it to have value and utility to a user'. Multiple studies mention how proper requirement gathering and analysis play a pivotal role in the project quality and success, with a majority of project failures being attributed to poor requirements gathering (Hickey and A.M. Davis, 2003; Sharma and Pandey, 2013; Alan Davis et al., 2006a).

3.2.1 Requirement types

Laplante (2019, p. 4) classifies requirements into 3 levels of abstraction: User requirements, System requirements and Design specifications. User requirements are the highest level of abstraction, are written in natural language and describe what services that system is expected to provide to the user. System requirements are more detailed, structured and precise, sometimes referred as functional specifications. Finally, design specifications are used by developers to implement the system, and is derived from analysis of system and design documentation. There 3 levels of abstraction are then later used during testing, each level corresponding to the respective testing level: User requirements are used in acceptance testing, System requirements are used in system and integration testing and Design specifications are used in unit testing (Laplante, 2019, p. 4).

The author also mentions that requirements can be classified into 2 categories: functional and non-functional requirements. Functional requirements describe the system's behavior, such as what the system should do and how it will react to different inputs, while non-functional requirements describe the system's quality attributes, such as performance, security, reliability, etc. (Laplante, 2019, p. 6).

When writing the requirements in a document, it is important to ensure that everything is written in a clear and concise manner to avoid any ambiguity. As such, Laplante (2019, p. 112) recommends using a standard format for writing all requirements, using simple language in a consistent manner, avoiding the use of technical language unless necessary, avoiding vague or speculative terms such as 'generally' or 'sometimes' and instead using precise terms or even ranges/values, and avoiding the use of conjunctions such as 'and', 'or', 'but' in a single requirement to not create multiple requirements in a single statement.

Additionally, the author points to "ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering" (2018), which provides a list of characteristics of individual requirements: necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, correct, and conforming.

The same standard also provides some examples of requirements attributes within a requirement document, such as identification, owner, priority, risk, rationale, difficulty, and type (functional/non-functional).

Finally, prioritizing requirements is crucial, especially in projects with numerous requirements. Several techniques can help determine which requirements to address first. Simple methods include categorizing requirements as low, medium, or high priority, or assigning a numerical value within a specific range (Khan et al., 2015). In addition to the previously mentioned techniques, the author mentions another approach - the MoSCoW method, which classifies requirements into four categories:

- Must have - must be implemented in the software before being released
- Should have - important but not necessary for the software to be released
- Could have - desirable but not necessary for the software to be released
- Won't have - requirements that are not included in the current release

Requirements in Agile

Laplante (2019, p. 191) provides an explanation of requirements within Agile projects. The author mentions that in Agile projects, the most basic unit of requirements are usually written in the form of User Stories, which are short, simple descriptions of a feature desired by the customer. User Stories are written in a specific format, such as 'As a [user], I want to [action] so that [benefit]'. Other components of a User Story include a title, acceptance test/criteria (a list of conditions that must be met for the story to be considered complete), priority, story points (estimated time to implement the user story), and description.

User stories are part of the Product and Sprint backlog - the former one containing the list

of requirements for the whole project and the latter containing the list of requirements for the current sprint/iteration.

3.2.2 Stakeholders

Laplante (2019, p. 34) explains that stakeholders represent the 'set of individuals who have some interest (a stake) in the success (or failure) of the system in question'. The author continues that there may be many types of stakeholders and stresses the importance of accurately and completely identifying all possible stakeholders for a system. Identification of all possible stakeholders needs to be done in the early stages of the project, as leaving out key stakeholder could lead to missing out on important requirements or constraints later on the project.

After identifying the stakeholders, it is important to do a stakeholder analysis by understanding the stakeholders' interests, needs, expectations, and influence on the project and then mapping them on a stakeholder matrix. One such matrix is the Influence/Interest grid, which classifies stakeholders based on their power and interest in the project (M., 2024). The matrix divides stakeholders into 4 categories:

1. Low influence, low interest - try to increase interest
2. Low influence, high interest - keep informed
3. High influence, low interest - engage and consult on interest area
4. High influence, high interest - key players, involve in decision making and engage regularly

3.2.3 Requirement gathering techniques

There are multiple requirement gathering techniques, the most popular ones being interviews, collaborative meetings, prototyping, modeling, brainstorming, storyboards, document analysis and ethnography (observing users interacting with an existing system) (Hickey and A.M. Davis, 2003; Young, 2002; Sharma and Pandey, 2013; Tiwari, Rathore, and Gupta, 2012). One of the studies interviewed several individuals with multiple years of experience in the field of requirement gathering and analysis. As a result, the authors found that the most used requirement gathering techniques were collaborative meetings, interviews, ethnography and modeling (Hickey and A.M. Davis, 2003). Another study by Saeeda et al. (2020) proposes a framework for improved requirement gathering in Scrum with a real-life case study of an IT project in Norway. The authors introduce the framework as being a series of 3 sections: pre-elicitation phase, where a high-level overview and breakdown of the requirements

is done during an interview with the stakeholders, a mid-elicitation phase, where the a deep dive into the requirements is done through the usage of mind maps and a post-elicitation phase, where the output of the previous phase is used to feed into a Scrum Product Backlog and later a Sprint Backlog.

3.2.4 Interview technique considerations

Multiple research papers point to interviews being recognised as the most commonly used technique for requirement gathering (Alan Davis et al., 2006b; Donati et al., 2017; Bano et al., 2019). Interviews can be divided into 3 different types: structured, semi-structured and unstructured, with the first type being more effective at gathering information from stakeholders versus the other 2 types (Alan Davis et al., 2006b; Wahbeh, Sarnikar, and El-Gayar, 2020).

Two articles by Mohedas et al. (2022) and Loweth et al. (2021) suggest a list of recommended practices for conducting interviews, such as:

1. Encouraging deep thinking through situational thinking or providing rationales.
2. Avoiding ambiguity by asking clarifying questions.
3. Being flexible by probing into relevant topics that have arisen during the discussion and not being rigidly attached to the interview script.
4. Verifying that the conclusion/summary aligns with the customer's vision.
5. Using projective techniques, such as analogies, scenarios, stories, and role-playing.
6. Having the stakeholder teach the analyst about a specific topic or a complex concept.
7. Stating the goals of the interview at the beginning and offering stakeholders time at the end to add any missing information.

On the other hand, two studies have been found that provide some insights into the potential mistakes that student analysts may do during requirement gathering sessions. One such study done by Donati et al. (2017) mentions the following mistakes:

1. Wrong opening - need to understand the context of the problem first, so it is important to understand the system as-is and how it would change with the new system before talking about the system itself.
2. Not leveraging ambiguity - ambiguity can reveal gaps in the knowledge between the analyst and the customer and be used to elicit important system-related knowledge.

3. Implicit goals - it is important to either explicitly ask/suggest clear goals or at least ask for more clarification.
4. Implicit stakeholders - not taking into account 3rd party stakeholders.
5. Non-functional requirements not considered
6. Interviews sounding like interrogations - to combat this, the authors recommend using techniques such as imagining, scenarios and teaching to let the customer explain their vision.
7. Problems in phrasing questions - some questions can be direct, so they need some background information to be understood.
8. Wrong closing - the authors stress the importance of a interview summary at the end of the session, to receive feedback from the customer if necessary.

A similar study by Bano et al. (2019) suggests the following mistakes that can be made during interviews:

1. Question formulation - asking vague, technical, irrelevant or long questions.
2. Question omission - not asking about stakeholders or existing business processes or not doing follow-up questions.
3. Order of interview - incorrect opening (no introductions, no description of the current situation) or ending (no summary)
4. Communication skills - too much technical jargon, not listening to the customer, interview sounding too strict or passiveness of the analyst.
5. Customer interaction - not building rapport with the customer
6. Planning - lack of planning in terms of sequence of questions to ask

3.3 Tech stack

3.3.1 Database

There are several types of databases, such as: relational (SQL), NoSQL databases, graph databases or object-oriented databases (Oracle, 2020). Due to the prototype nature of the project, it is not expected that the data used in the system will be complex, so the next subsections will focus on the two most common types of databases - relational and NoSQL databases.

Relational databases

Relational databases store structured data in tables, linked through keys to create relationships like one-to-one and one-to-many (Anderson and Nicholson, 2022). They use SQL (Structured Query Language) to create powerful queries and schemas to help organise and manage data efficiently. Anderson and Nicholson (2022) highlight that relational databases, by following ACID (Atomicity, Consistency, Isolation, Durability) principles, ensure high data integrity for industries like finance and healthcare. Relational databases are widely used and have a large community, making it easier to find support and resources. However, the rigid schema limits adaptability to rapid data changes, and scaling horizontally in distributed systems can be challenging. Examples include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

NoSQL databases

NoSQL databases manage unstructured or semi-structured data without rigid schemas or relationships (Anderson and Nicholson, 2022). As the authors describe, NoSQL databases, such as key-value, document, column-family, and graph databases, excel in flexibility and scalability. Guided by the CAP (Consistency, Availability, Partition Tolerance) theorem, NoSQL databases often prioritize performance over strict consistency, making them suitable for large, varied datasets of unstructured data but less ideal for complex transactions. Although growing in popularity, NoSQL systems lack SQL's mature standardization and support. Examples include MongoDB, Cassandra, Couchbase, and Redis.

3.3.2 Backend framework

Choosing the right backend framework is crucial for the development of the project deliverable. The backend framework is responsible for handling the business logic of the application, such as processing requests, interacting with the database, and returning responses to the client. A good framework can also come with the added benefit of included features such as security and authentication, database support through an ORM (Object-Relational Mapping) and community support through existing documentation, library of extensions and a user base that can provide support if necessary.

Based on recent a recent survey by Vailshery (2024), the most popular backend frameworks are Express, Flask, Spring Boot, Django and Laravel. This data is also supported by the Stack Overflow Developer Survey 2024, which lists the most popular programming languages as JavaScript (Express), Python (Flask, Django), Java (Spring Boot) and PHP (Laravel) (Stack Overflow", 2024).

Due to their high popularity among developers, the decision was made to compare the above frameworks, as the student will be able to much easier find support and online resources if necessary. Utilising the information from Broadcom (2024), Laravel Holdings Inc. (2024), OpenJS Foundation (n.d.), and Django (2024), the table below will compare the most popular backend frameworks based on the following criteria: programming language used, learning curve, community support, security features, database features, and project size suitability.

Framework	**Django**	**Laravel**	**Spring Boot**	**Express**
Language	Python	PHP	Java	JavaScript
Learning curve	Medium	Low	High	Low
Community Support	High	High	High	High
Security features	High	High	High	Medium
Database features	Medium	Medium	High	Medium
Project size suitability	Small to medium	Small to medium	Medium to large	Small to medium

Table 3.1: Comparison of backend frameworks

3.3.3 Frontend framework

Similar to the backend frameworks, choosing a suitable frontend framework is equally important. The frontend framework is responsible for the user interface of the application, such as displaying data, handling user interactions, and making requests to the backend. A good frontend framework can also come with the added benefit of included features such as routing, state management, and component libraries.

Based on the same survey by Vailshery (2024), the most popular frontend frameworks are React, Angular, Vue.js, and Svelte. This data is also supported by the Stack Overflow Developer Survey 2024, where the above frameworks rank among the highest for developers wishing to continue using them (Stack Overflow", 2024).

As such, the table below will utilise the information from Meta Platforms (n.d.), Google (2024), You (2024), and Svelte (n.d.) to compare the most popular frontend frameworks based on the following criteria: learning curve, community and documentation, ecosystem and tooling support, performance, state management, and project size suitability.

Framework	React	Angular	Vue.js	Svelte
Learning curve	Low	High	Medium	Low
Community and documentation	High	High	High	Medium
Ecosystem and tooling support	High	High	Medium	Low
Performance	High	Medium	Medium	High
State management	High	High	Medium	Low
Project size suitability	Small to large	Medium to large	Small to medium	Small to medium

Table 3.2: Comparison of frontend frameworks

3.4 Machine Learning models

3.5 Gap Analysis

3.5.1 Medvalet

A similar solution to the one proposed in this project is Medvalet - a mobile app developed in Romania that allows patients to upload their history in the form of PDF or scanned documents to the application (Medvalet, 2024).

The application offers several features such as:

- Upload of documents - patients can upload their medical history in the form of PDF or scanned documents
- Categorization - the application allows the user to choose the type of document they are uploading, such as prescriptions, lab results, medical imaging, etc.
- Patient vitals - the application allows the user to input their vitals, such as blood pressure, heart rate, weight, etc. The vitals are shown in a graphical format, allowing the user to track their progress over time.
- Patient information - the application allows the user to input their personal information, such as name, age, height, weight, etc.
- Doctor accounts - the application allows doctors to create accounts and access the patient's medical history directly through the app by scanning a QR code generated by the patient or by searching for the patient's name.

While the application offers a robust set of features, there are several areas where the application could be improved:

- Doctors need to create an account to access the patient's medical history - this could be a barrier for doctors who are not familiar with technology or do not have the time

to create an account.

- Doctors can always access the patient's medical history via the app - this could be a privacy concern for patients, as doctors could access the patient's medical history without their consent.
- All history is stored as documents - this could be cumbersome for doctors to navigate through, especially if the patient has a long medical history.
- No extraction of data or values from the documents - the application does not extract any data or values from the documents, such as lab results which could be useful for doctors to quickly access the information they need.
- The app is only available on mobile - this could be a barrier for doctors who prefer to access the app on a desktop thanks to the larger screen size.

3.5.2 Andaman7

Another similar solution is Andaman7, a mobile app developed by a Belgian-American eHealth company with the goal to improve communication between doctors and patients (Software, 2022). The application offers several features such as:

- Multiple sections - the application contains multiple sections where patients can input their personal information, medical history, allergies, vaccinations, medication, personal logbook, etc.
- Automatic collection of health data - patients using this app can access their health information from over 300 hospitals and clinics in the US and Europe.
- Collection of multiple data sources - the app collects a health data from a variety of sources: doctors, hospitals, labs, smart devices and even manual input.
- No access to patient data - the app does not have access to the patient's data, as the data is stored on the patient's device.
- Sharing data with doctors - patients can share their health data with their doctors by using a QR code, who can access the data through the app.
- Can control the shared data - patients can control what data is shared with their doctors, and can revoke access at any time.
- GDPR and HIPAA compliant - the app is compliant with the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA).

- Tracking of health data - the app allows patients to track their health data, such as blood pressure, weight, heart rate, allergies, vaccinations, etc.
- AI features - the app has recently introduced 3 new AI features: a summarization tool, a translation tool and a simplified medical jargon tool.

While the application offers a robust set of features, there are several areas where the application could be improved:

- Neccessity of an account - both patients and doctors need to create an app to be able to share and access the patient's medical history.
- No automatic extraction of data - the app does not automatically extract data from the documents, such as lab results, which could be useful for doctors to quickly access the information they need.
- Lack of desktop version - the app is only available on mobile, which could be a barrier for doctors who prefer to access the app on a desktop thanks to the larger screen size.

3.5.3 Fasten Health

Fasten Health is an open source and self-hosted electronic medical record aggregator for patients (Health, 2022). While the application can be locally hosted, there are also Windows and Mac versions available for a fee. The application offers several features such as:

- Aggregating medical records - the application allows patients to automatically aggregate their medical records from multiple healthcare providers, such as hospitals, clinics, labs, etc.
- Self-hosted - the application can be self-hosted, allowing patients to have full control over their data, which is only stored locally on their device.
- Support for multiple protocols - the application supports multiple protocols, such as DICOM, FHIR and OAuth2.
- Manual entry of data - patients can manually enter their health data, such as allergies, vaccinations, medications, etc.
- Multi-user support - the application supports multiple users, allowing families to share the same application.
- Multiple dashboards - the application contains multiple dashboards where patients can view their health data, such as lab work, vitals, etc.

- Robust interface - the application has a robust interface, allowing patients to easily navigate through the application and view their health data by using well-made graphs and charts.

While the application offers a robust set of features, there are several areas where the application could be improved:

- Paid versions - the Windows and Mac versions are paid, which could be a barrier for patients who cannot afford the application or cannot self-host it on their devices.
- Manual entry of data - currently, manual entry of data is only possible through an encounter (either existing or new one), which could be cumbersome for patients to navigate through.
- No OCR or text extraction - the application does not extract any data or values from the documents, such as lab results.
- One-way share of data - the application does not allow patients to share their health data with their doctors.
- Self-hosted set up - the application requires patients to self-host the application, which could be a barrier for patients who are not familiar with technology.
- Limited to US providers - the application only supports healthcare providers in the US, which could be a barrier for patients who are not from the US.

Chapter 4

Project Planning and Design

4.1 Project Management Methodology and Tooling

Based on the research above, the student has decided that he will be using a hybrid approach, with Waterfall as the main methodology for planning managing the project. The development part of the project will be done using ScrumBan, so that the student will be able to utilise elements from both frameworks. There are several reasons for this choice:

1. The nature of the project - the student is working on a project that has a limited timeframe (about 6-7 months) and is of a smaller scale.
2. Documentation requirements - the student is required to document the progress during the project in this report, including the requirements gathered, design considerations and implementation decisions and outcomes.
3. Regulatory requirements - the student is required to adhere to the regulations and standards of the healthcare industry, which may require extensive documentation and planning.
4. Customer involvement - the student will be working closely with the project stakeholder, who will be providing feedback and guidance throughout the project.
5. Familiarity with both Agile and Waterfall - the student has experience with both Agile (specifically Scrum and Kanban) and Waterfall methodologies, and has worked on projects that have used both approaches.

The student will use Jira Software as their project management tool, which is one of the most popular project management tool for software development projects that supports

working with Agile frameworks such as Scrum and Kanban (Springer, 2023). The student has experience with Jira Software, having used it in previous projects, and is familiar with its features and capabilities.

4.2 Product Backlog

4.3 System Design

4.3.1 UML Diagrams

UML, or Unified Modeling Language, is a standardized modeling language that consists of a set of diagrams used for modeling business processes and visualizing and documenting software systems (Paradigm, 2024). UML employs graphical notations to represent a system, aiding teams in better communicating potential designs and architectural decisions. Utilizing UML can enhance quality, reduce costs, and provide multiple perspectives on the system.

UML offers various diagrams, each serving a specific purpose. The most common diagrams include:

- **Use Case Diagram** - Illustrates the system's intended functionality in terms of actors, use cases, and their relationships, showing how the system delivers value to users.
- **Class Diagram** - Depicts the structure of the system by showing classes, attributes, operations, and static relationships between classes.
- **Sequence Diagram** - Demonstrates how objects interact in a particular, timed sequence scenario, focusing on the messages passed between objects.
- **Activity Diagram** - Represents the workflow of a target use case or business process through a series of activities, emphasizing steps, choices, iterations, and concurrency.

The student has used UML diagrams to present the stakeholders with a visual representation of the system's design and architecture. The diagrams can be found below.

4.3.2 Database Design

4.4 Project Tech Stack

4.4.1 Frontend

4.4.2 Backend

4.4.3 Database

References

Alqudah, Mashal and Rozilawati Razali (2018). "An empirical study of Scrumban formation based on the selection of scrum and Kanban practices". In: *Int. J. Adv. Sci. Eng. Inf. Technol* 8.6, pp. 2315–2322.

Alshamrani, Adel and Abdullah Bahattab (2015). "A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model". In: *International Journal of Computer Science Issues (IJCSI)* 12.1, p. 106.

Anderson, Benjamin and Brad Nicholson (2022). *SQL vs. NoSQL Databases: What's the difference?* URL: https://www.ibm.com/think/topics/sql-vs-nosql. (Accessed: 28 Oct 2024).

Bano, Muneera et al. (2019). "Teaching requirements elicitation interviews: an empirical study of learning from mistakes". In: *Requirements Engineering* 24, pp. 259–289.

BBC (2024). *Moldova country profile.* URL: https://www.bbc.co.uk/news/world-europe-17601580. (Accessed: 22 Oct 2024).

Broadcom (2024). *Why Spring?* URL: https://spring.io/why-spring. (Accessed: 14 Nov 2024).

Ciurcă, Aliona (2021). *Sistemul informațional automatizat din spitale: cum funcționează în R. Moldova și ce cred medicii că ar trebui ajustat.* URL: https://www.zdg.md/stiri/stiri-sociale/sistemul-informational-automatizat-din-spitale-cum-functioneaza-in-r-moldova-si-ce-cred-medicii-ca-ar-trebui-ajustat/. (Accessed: 22 Oct 2024).

Cunningham, Ward (2001). *Manifesto for Agile Software Development.* URL: https://agilemanifesto.org/. (Accessed: 25 Oct 2024).

Davis, Alan et al. (2006a). "Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review". In: *14th IEEE International Requirements Engineering Conference (RE'06)*, pp. 179–188. DOI: 10.1109/RE.2006.17.

Davis, Alan et al. (2006b). "Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review". In: *14th IEEE International Requirements Engineering Conference (RE'06)*, pp. 179–188. DOI: 10.1109/RE.2006.17.

Django (2024). *Why Django?* URL: https://www.djangoproject.com/start/overview/. (Accessed: 14 Nov 2024).

Donati, Beatrice et al. (2017). "Common mistakes of student analysts in requirements elicitation interviews". In: *Requirements Engineering: Foundation for Software Quality: 23rd International Working Conference, REFSQ 2017, Essen, Germany, February 27–March 2, 2017, Proceedings 23*. Springer, pp. 148–164.

E-Governance Agency (n.d.[a]). *About EGA*. URL: https://egov.md/en/about-ega. (Accessed: 22 Oct 2024).

— (n.d.[b]). *Citizen's Government Portal*. URL: https://egov.md/en/content/citizens-government-portal. (Accessed: 22 Oct 2024).

Gemino, Andrew, Blaize Horner Reich, and Pedro M. Serrador (2021). "Agile, Traditional, and Hybrid Approaches to Project Success: Is Hybrid a Poor Second Choice?" In: *Project Management Journal* 52.2, pp. 161–175. DOI: 10.1177/8756972820973082. eprint: https://doi.org/10.1177/8756972820973082. URL: https://doi.org/10.1177/8756972820973082.

Google (2024). *What is Angular?* URL: https://angular.dev/overview. (Accessed: 14 Nov 2024).

Health, Fasten (2022). *What is Fasten?* URL: https://docs.fastenhealth.com/. (Accessed: 12 Nov 2024).

Hickey, A.M. and A.M. Davis (2003). "Elicitation technique selection: how do experts do it?" In: *Proceedings. 11th IEEE International Requirements Engineering Conference, 2003.* Pp. 169–178. DOI: 10.1109/ICRE.2003.1232748.

"ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering" (2018). In: *ISO/IEC/IEEE 29148:2018(E)*, pp. 1–104. DOI: 10.1109/IEEESTD.2018.8559686.

Khan, Javed Ali et al. (2015). "Comparison of requirement prioritization techniques to find best prioritization technique". In: *International Journal of Modern Education and Computer Science* 7.11, p. 53.

Laplante, Phillip A (2019). *Requirements engineering for software and systems*. 3rd. Auerbach Publications.

Laravel Holdings Inc. (2024). *The PHP Framework for Web Artisans*. URL: https://laravel.com/. (Accessed: 14 Nov 2024).

Loweth, Robert P. et al. (Mar. 2021). "A Comparative Analysis of Information Gathering Meetings Conducted by Novice Design Teams Across Multiple Design Project Stages".

In: *Journal of Mechanical Design* 143.9, p. 092301. ISSN: 1050-0472. DOI: 10.1115/1.4049970. eprint: https://asmedigitalcollection.asme.org/mechanicaldesign/article-pdf/143/9/092301/6667332/md_143_9_092301.pdf. URL: https://doi.org/10.1115/1.4049970.

Luo, Lu (2001). "Software testing techniques". In: *Institute for software research international Carnegie mellon university Pittsburgh, PA* 15232.1-19, p. 19.

M., Tam (2024). *Stakeholder Analysis | Definition and best method*. URL: https://www.stakeholdermap.com/stakeholder-analysis.html. (Accessed: 31 Oct 2024).

Medvalet (2024). *Despre noi*. URL: https://medvalet.ro/. (Accessed: 11 Nov 2024).

Meta Platforms (n.d.). *React. The library for web and native user interfaces*. URL: https://react.dev/. (Accessed: 14 Nov 2024).

Mohedas, Ibrahim et al. (2022). "The use of recommended interviewing practices by novice engineering designers to elicit information during requirements development". In: *Design Science* 8, e16. DOI: 10.1017/dsj.2022.4.

National Bureau of Statistics of the Republic of Moldova (2024). *Population*. URL: https://statistica.gov.md/en/statistic_indicator_details/25. (Accessed: 22 Oct 2024).

OpenJS Foundation (n.d.). *Fast, unopinionated, minimalist web framework for Node.js*. URL: https://expressjs.com/. (Accessed: 14 Nov 2024).

Oracle (2020). *What is a Database?* URL: https://www.oracle.com/uk/database/what-is-database/. (Accessed: 28 Oct 2024).

Paradigm, Visual (2024). *What is Unified Modeling Language (UML)?* URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#sequence-diagram. (Accessed: 07 Nov 2024).

Prenner, Nils, Carolin Unger-Windeler, and Kurt Schneider (2020). "How are hybrid development approaches organized? A systematic literature review". In: *Proceedings of the International Conference on Software and System Processes*, pp. 145–154.

Ruparelia, Nayan B. (May 2010). "Software development lifecycle models". In: *SIGSOFT Softw. Eng. Notes* 35.3, pp. 8–13. ISSN: 0163-5948. DOI: 10.1145/1764810.1764814. URL: https://doi.org/10.1145/1764810.1764814.

Saeeda, Hina et al. (2020). "A proposed framework for improved software requirements elicitation process in SCRUM: Implementation by a real-life Norway-based IT project". In: *Journal of Software: Evolution and Process* 32.7. e2247 JSME-19-0129.R1, e2247. DOI: https://doi.org/10.1002/smr.2247. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2247. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2247.

Sharma, Shreta and SK Pandey (2013). "Revisiting requirements elicitation techniques". In: *International Journal of Computer Applications* 75.12.

Software, A7 (2022). *Who are we ?* URL: https://www.andaman7.com/en/who-are-we. (Accessed: 11 Nov 2024).

Springer, Andreas (2023). *What is Jira Software, and why use it?* URL: https://community.atlassian.com/t5/Jira-articles/What-is-Jira-Software-and-why-use-it/bap/2323812. (Accessed: 26 Oct 2024).

Stack Overflow" (2024). *Technology.* URL: https://survey.stackoverflow.co/2024/technology#most-popular-technologies-language. (Accessed: 14 Nov 2024).

Sunner, Daminderjit (2016). "Agile: Adapting to need of the hour: Understanding Agile methodology and Agile techniques". In: pp. 130–135. DOI: 10.1109/ICATCCT.2016.7911978.

Svelte (n.d.). *Overview.* URL: https://svelte.dev/docs/svelte/overview. (Accessed: 14 Nov 2024).

Tiwari, Saurabh, Santosh Singh Rathore, and Atul Gupta (2012). "Selecting requirement elicitation techniques for software projects". In: *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, pp. 1–10. DOI: 10.1109/CONSEG.2012.6349486.

United Nations Development Programme (2023). *A 100% digital state: The strategy for digital transformation of the Republic of Moldova for 2023-2030, approved by the Executive.* URL: https://www.undp.org/moldova/press-releases/100-digital-state-strategy-digital-transformation-republic-moldova-2023-2030-approved-executive. (Accessed: 22 Oct 2024).

USMF (2023). *Brief history.* URL: https://usmf.md/en/brief-history-usmf. (Accessed: 22 Oct 2024).

Vailshery, Lionel Sujay (2024). *Most used web frameworks among developers worldwide, as of 2024.* URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/. (Accessed: 14 Nov 2024).

Wahbeh, Abdullah, Surendra Sarnikar, and Omar El-Gayar (Sept. 2020). "A socio-technical-based process for questionnaire development in requirements elicitation via interviews". In: *Requirements Engineering* 25 (3), pp. 295–315. DOI: https://doi.org/10.1007/s00766-019-00324-x.

Yas, Qahtan, Abdulbasit Alazzawi, and Bahbibi Rahmatullah (Nov. 2023). "A Comprehensive Review of Software Development Life Cycle methodologies: Pros, Cons, and Future Directions". In: *Iraqi Journal For Computer Science and Mathematics* 4.4, pp. 173–190. DOI: 10.52866/ijcsm.2023.04.04.014. URL: https://journal.esj.edu.iq/index.php/IJCM/article/view/664.

You, Evan (2024). *What is Vue?* URL: https://vuejs.org/guide/introduction.html. (Accessed: 14 Nov 2024).

Young, Ralph R (2002). "Recommended requirements gathering practices". In: *CrossTalk* 15.4, pp. 9–12.