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

Abstract ii

Acknowledgements iii

1 Introduction 1

1.1 Background . . . 1
1.2 Problem Statement . . . 2
1.3 The Client . . . 2

2 Literature Review 3

2.1 Software development methodologies . . . 3
2.1.1 Software Development Life Cycle . . . 3
2.1.2 SDLC Models . . . 4
2.1.3 A hybrid approach . . . 7
2.2 Requirements gathering . . . 8
2.2.1 Requirement types . . . 8
2.2.2 Stakeholders . . . 10
2.2.3 Requirement gathering techniques . . . 11
2.2.4 Interview technique considerations . . . 11
2.3 System design . . . 13
2.4 Tech stack . . . 13
2.4.1 Database . . . 13
2.4.2 Backend framework . . . 14
2.4.3 Frontend framework . . . 15
2.5 Machine Learning models . . . 15
2.6 Gap Analysis . . . 15
2.6.1 athenahealth . . . 15

2.6.2 drchrono 16
2.6.3 AdvancedMD 16
2.6.4 Netsmart 16

3 Project Planning and Design **18**
3.1 Project Management Methodology and Tooling 18
3.2 Requirements and Stakeholder Analysis 19
3.2.1 Stakeholder Analysis 19
3.2.2 Requirements 19
3.2.3 Product Backlog 19
3.3 Project Tech Stack 19
3.3.1 Frontend 19
3.3.2 Backend 19
3.3.3 Database 19
3.4 System Design 19
3.4.1 UML Diagrams 19
3.4.2 Database Design 19

References **20**

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 first challenge occurs 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).

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

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

1.3 The Client

To address these challenges, this project, in collaboration with "Nicolae Testemiteanu" State University of Medicine and Pharmacy in Moldova (USMF), seeks to develop a prototype for a modernized EHR system. The client, 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 EHR system in place. As such, the client has expressed a need for a new EHR system prototype that is more user-friendly, accessible on a wider range of devices and browsers, and that can be customized to meet the specific needs of different hospitals, clinics, departments and users.

Chapter 2

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

2.1 Software development methodologies

2.1.1 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).

2.1.2 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.

2.1.3 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.

2.2 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).

2.2.1 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) comes with a some recommendations for writing requirements:

1. Using a standard format for writing all requirements
2. Using simple language in a consistent manner
3. Avoiding the use of technical language unless necessary
4. Avoiding vague or speculative terms such as 'generally', 'sometimes', instead using precise terms or even ranges/values
5. Avoiding use of conjuctions 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:

1. Necessary
2. Appropriate
3. Unambiguous
4. Complete
5. Singular
6. Feasible
7. Verifiable
8. Correct
9. Conforming

The same standard also provides some examples of requirements attributes within a requirement document, such as:

1. Identification
2. Owner
3. Priority
4. Risk
5. Rationale

6. Difficulty
7. Type (functional/non-functional)

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]'. A list of other User Story components is given by the author:

1. Title
2. Acceptance test/criteria - a list of conditions that must be met for the story to be considered complete
3. Priority
4. Story points - estimated time to implement the user story
5. 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.

2.2.2 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

2.2.3 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.

2.2.4 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

2.3 System design

2.4 Tech stack

2.4.1 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 are collections of data where the data is stored in rows and tables, linked through keys that allow different types of relationships: one-to-one, one-to-many and many-to-many (Anderson and Nicholson, 2022). SQL (Structured Query Language) is the programming language used to query relational databases. Anderson and Nicholson (2022) mention the advantages of using relational databases is in their ability to handle structured data in well-organized manner by using schemas and tables, making it easy to query, even for complex queries. Additionally, the authors argue that by following the ACID (Atomicity, Consistency, Isolation, Durability) properties, relational databases ensure that the data within is consistent and reliable. Finally, relational databases are widely used and have a large community of users, which makes it easier to find support and resources. Example of relational databases include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.

NoSQL databases

NoSQL databases, on the other hand, are databases that allow developers to work with unstructured data (such as documents, images, etc) without relying on rigid schemas, tables

or relationships between data (Anderson and Nicholson, 2022). As described by the authors, NoSQL databases generally fall into four different structures: key-value stores, document stores, column-family stores, and graph databases. Similarly to relational databases, NoSQL databases follow their own set of properties, such as the CAP (Consistency, Availability, Partition Tolerance) theorem, which states that a distributed system can only guarantee two of the three properties at the same time. While NoSQL databases are not as widely used as relational databases, they are gaining popularity due to their ability to handle large amounts of data and their flexibility in handling unstructured data. Examples of NoSQL databases include MongoDB, Cassandra, Couchbase, and Redis.

2.4.2 Backend framework

It is important to choose the right backend framework for the project, as it is responsible for handling multiple functions on the server side of the application, such as routing, database access, authentication, real-time functionality and more (*Backend Frameworks List: 2024 Overview* 2024). The author mentions that an advantage of choosing a backend framework is that it provides better security and scalability, but also improves the quality and speed of development thanks to existing community support through libraries or tools. However, due to the number of existing frameworks, the choice of a framework is not an easy task - experience and familiarity with the programming language is important, but also how well the framework fits the project requirements. Currently, the most popular backend frameworks are Express.js, Koa, Django, Ruby on Rails, Flask, Spring or Laravel (*Backend Frameworks List: 2024 Overview* 2024).

Some frameworks are more suited for smaller projects, while others are more suited for larger, complex projects. For example, Express.js, Flask or Koa are minimalistic frameworks that are easy to use and are well suited for smaller projects, while Django, Ruby on Rails, Spring or Laravel are more comprehensive frameworks that offer a range of built-in features and are more suited for larger projects (*Backend Frameworks List: 2024 Overview* 2024). Due to the nature of the project being more complex and the student's familiarity with Python and Javascript programming languages, the next sub-sections will focus on Django and Express.js.

Express.js

Express.js is a simple, but powerful framework that uses Node.js to make web applications (*Backend Frameworks List: 2024 Overview* 2024). In the article, the author explains that Express.js offers functionality such as routing, middleware, templating and restful API creation. Thanks to its simplicity, flexibility, performance, large community and ecosystem due

to Node.js and Javascript, the author argues in favor of Express.js. However, the author also mentions that Express.js can be very minimal and its lack of structure may make it unsuitable for larger projects, as it may require additional setup or libraries and tools to be used.

Django

Django is a high-level Python framework, with a range of built-in features such as an ORM, authentication and admin interface (Mashutin, 2024). The article mentions key features of using Django, such as its comprehensive suite of built-in features, its focus on security (which includes both user authentication but also protection from common web attacks) and structure of the application but also its support and popularity in the user community. However, even though Django offers a lot of features, the author mentions that the framework can be limited in some things, such as support for NoSQL databases, as it is primarily focused on relational databases, or its complexity, which may be overwhelming for beginners. As such, Django seems to be well suited for mostly large, complex projects that will utilise its range of built-in features.

2.4.3 Frontend framework

2.5 Machine Learning models

2.6 Gap Analysis

2.6.1 athenahealth

https://www.athenahealth.com/solutions/electronic-health-records https://www.selecthub.com/medical-software/ehr/ehr-examples/

Positive things:

1. Patient portal to manage appointments
2. Mobile applications
3. Usage of coding systems like ICD-10 and CPT
4. Clean UI
5. Easy to view patient history
6. Easy to add vitals

7. Easy to create patient report through clicking of boxes with pre-defined text
8. A lot of reporting/chart features
9. Auto population of info from scanned documents
10. Billing management

2.6.2 drchrono

https://www.drchrono.com/

Positive things:

1. Virtual visits through telemedicine
2. Clean UI
3. Billing management
4. Mobile app
5. Online scheduling tools
6. Easy to add vitals - default and custom view
7. Lab integration
8. Custom templates and forms (drag and drop elements) - filter by speciality or keywords
9. Speech to text
10. Library of forms, easy to create charts
11. Easy to add text in fields by selecting existing options
12. medical codes

2.6.3 AdvancedMD

https://www.advancedmd.com/

Positive things:

1.

2.6.4 Netsmart

https://www.ntst.com/

Positive things:

1.

EHR system features:

1. Customization and flexibility 2. Use of a patient portal and automated patient communication features - access to records, appointment management, medication requests, virtual comms 3. System integration (billing, test labs, etc) 4. Business Intelligence (resource management, scheduling, etc) 5. Analytics 6. Telehealth 7. Access to patient history 8. Processing claims 9. Digital vital charts 10. Voice recognition and other AI features 11. Generating reports, notes, etc 12. Prescriptions 13. Mobile design 14. Security and privacy

Chapter 3

Project Planning and Design

3.1 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).

3.2 Requirements and Stakeholder Analysis

3.2.1 Stakeholder Analysis

3.2.2 Requirements

3.2.3 Product Backlog

3.3 Project Tech Stack

3.3.1 Frontend

3.3.2 Backend

3.3.3 Database

3.4 System Design

3.4.1 UML Diagrams

3.4.2 Database Design

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).

Backend Frameworks List: 2024 Overview (2024). URL: https://daily.dev/blog/backend-frameworks-list-2024-overview. (Accessed: 02 Nov 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).

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.

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.

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.

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

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).

Mashutin, Denis (2024). *Django vs. Flask: Which Is the Best Python Web Framework?* URL: https://blog.jetbrains.com/pycharm/2023/11/django-vs-flask-which-is-the-best-python-web-framework/. (Accessed: 02 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).

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

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).

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.

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).

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.

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