NTNU | Norwegian University of Science and Technology

Department of Computer Science

IDATT1002 - Software Engineering

Budgeting Application

Project report

Authors:

Magnus Gjerstad

Kristiane Skogvang Kolshus

Henrik Teksle Sandok

Knut Skoe

Melissa Visnjic

April, 2023

Summary

As a part of an assignment in the course IDATT1002 the team made a budgeting application for a bank in Trondheim named BN Bank. Through this report theory regarding the work, processes, methods and results are discussed, as well as a discussion section where we talk about future improvements.

The project resulted in a standalone application that allows for the possibility of future migration into BN Bank's existing systems.

Preface

This report in addition to an application is the main project in the course IDATT1002 software development at NTNU. The project involved creating a standalone application in Java using JavaFX or FXML. A more detailed and defined description can be found in "Assignment description" in the document, as well as in the appendix labeled "A. Vision Document".

The main purpose of this project was to use relevant knowledge taught during the course, into practice. This includes theory about topics such as, software development methods, principles related to universal design, software architecture, HMI (Human Machine Interaction), testing and etics revolving around the profession as a developer and data privacy. In addition to this, a main objective would be to learn how to manage and function in a group in an efficient and structured manner.

The assignment was set when the team started. We were to create a standalone application that would contribute to better financial control, for a customer of our choice. "BN Bank" became our customer, as we thought that an economic related application would be something they would be interested in. In addition it would be a good opportunity to work with a real client of this size and would give us greater learning outcomes and experience. Functionality, design and other details were discussed and set by the client.

Lastly we wish to give our regards to our classmates that always were willing to help, stackoverflow for being such a faithful friend, and our lecturers that have provided us with a great amount of information so that we were able to complete the project.

Knut Skoe

Kristiane Skogvang Kolshus

Magnus Gjerstad

Henrik Teksle Sandok

Melissa Visnjic

Trondheim 28/04/2023

Time and place

Table of contents

Table and Figure List

Assignment Description

The assignment given was to create a tool for private individuals or businesses related to budgeting and accounting that would provide an overview of the customers income and expenses in order to help improve their economy. The team was assigned the responsibility of identifying potential customers themselves, and creating this tool tailored to their needs. There were also technical requirements set as part of the assignment. One of these technical requirements was to create the software as a standalone Java application using JavaFX. In addition, any use of databases would have to be MySQL provided by NTNU.

The team reached out to BN Bank, a Norwegian bank headquartered in Trondheim, with the proposal to develop a tool that could potentially benefit them and their clients. Details regarding their specific wishes can be found in Appendix - Vision Document.

The assignment specified that the development was to be divided in three iterations. The first iteration was focused on prototyping, vision, and requirements. We were assigned to create Wireframes, using BalsamiQ, as the prototypes. The team was also tasked with creating a vision-document in collaboration with the client. The vision-document contains a feature list and all product requirements set by the development team and BN Bank. **(**Appendix - Vision Document.**)**. Use case scenarios were developed in tandem with the development of the product requirements, as described in the Use case diagrams. **(**Appendix - GitLab Wiki**)**. The second iteration focused on creating the MVP, which had to be presented to a lecturer and the client. The third iteration was dedicated to finishing up the project. This included finishing the software system, creating required manuals, and finishing the report and documentation.

1 Introduction

The project in relation to the school subject IDAT2001, is sufficient enough for the purpose of the project, even though there may be more work regarding the application with the client. As the team worked to both please the client and the professor, certain changes were done in order to satisfy them both equally. Therefore there is a possibility for future development with the bank, regardless of this subject, where the application is to be a part of a greater system, rather than a stand alone application.

Organization

The report is divided into 6 sections and the table of contents provides an overview that indicates where to find each chapter and their subchapters.

Section 1 consists of an introduction to the project as a whole, as well as lists containing definitions, acronyms and abbreviations.

Section 2 consists of the theory that has been acquired throughout the duration of this project, from lectures, syllabus books and experience.

Section 3 outlines the methods used in the project and gives a description of choices made during the project, based on the framework established in section 2 and the vision document. This chapter covers requirements such as user testing, prototyping, universal design and more. The team's work and role distribution are also described, in addition to charts and further documentation of the work.

Section 4 presents the results of the project, including tangible results in form of an application and a report, as well as intangible results achieved by the team.

Section 5 is the section that describes the overall successes and failures, and what there is to learn from them. It also describes how issues have been resolved, how it could have been done differently and what limitations that were encountered, and how this affected the product.

Section 6 concludes the rapport and the project as a whole, along with further work that can and/or will be done with the application.

Lastly the Git repository is added next to references and other attachments from the appendix.

Definitions, Acronyms, and Abbreviations

API	Application Programming Interface
BalsamiQ	Software used to create wireframes for prototyping.
DMS	Database Management System
EU	European Union
FXML	XML-based user interface markup language for defining the user interface of a JavaFx application
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HMI	Human Machine Interaction
Java	Object oriented programming language used in this project
JavaFX	Open source software platform for creating applications built on Java.
MVC	Model View Controller
MVP	Minimal Viable Product
MySQL	Open source relational database management system.
RDBM	Relational Database Management System
SQL	Structured Query Language
UD	Universal Design
UI	User Interface

UP	Unified Process
Use case diagram	Graphical depiction of possible interactions a user can have with the system
UX	User Experience
WCAG	Web Content Accessibility Guidelines
TDD	Test driven development????
UT	Usability Test
UAT	User Accseptance test
HCI	Human Computer Interaction

Theory and Relevant Literature

2.1 System Development Methods

System development methods refer to different approaches for developing software. Each method has its own advantages and disadvantages, which are weighed against what one wants to achieve with the project.

2.1.1 Agile System Development

Agile software development is a collective term for iterative and incremental system development. This method differs from other more traditional methods in the sense that it puts a bigger emphasis on the collaboration within the team and with the customer (Norozi, 2023). This close collaboration makes it easier to understand the customer's needs and desires regarding the product, but also allows for more flexibility by avoiding strict formal processes and documentation, as trust and communication play a greater role. The close collaboration also allows for greater adaptability and makes room for continuous changes, as the development team works closely together. This method involves constant change, often following a test driven development pattern (TDD) to ensure the functionality of the code, despite the many changes.

The agile method differs from more traditional methods like the waterfall- and spiral- method, by following a more empirical management system. The agile method acknowledges that there is no way to predict what could happen in advance, and therefore leaves room for changes and learning during the project. This method will therefore allow for the possibility to go back and make changes to the code. Whereas in a more traditional approach, there is a greater emphasis on the planning behind the project, and therefore believes that there is no need for major changes along the way. This results in a more rigid development where one works in sequences, where each task is fully completed in order to move on to the next, and does not go back once it is completed. These methods also required a significant amount of documentation, in addition to very expensive changes when they occurred.

2.1.1.1 Iterative

The iterative part of the development has its focus on developing the software through a sequence of iterations, prioritized so that each iteration provides a partially functional solution. This will provide the customer an

opportunity to provide feedback along the process, so that the possible changes can be taken into account as soon as possible. The iterative system development method therefore provides quick and continuous feedback which can help contribute to fewer errors and deficiencies already from the early stages, as well as throughout the project. One can say that the iterative process divides iterations on time.

2.1.1.2 Incremental

Incremental development is about breaking down the software into smaller, more manageable pieces, also called sprints. Similar to iterative system development, this method will allow for testing the solution along the way and reduce the likelihood of great errors that are hard to debug. In contrast, incremental development mainly focuses on building the software bit by bit, while the iterative method primarily focuses on uncovering errors as early as possible. Opposite to the iterative process, the project is divided into increments that divide the project into tasks, and not time.

2.1.1.3 Scrum

Scrum is an example of a widely used agile development method. In the Scrum process there are three roles; Product Owner, Scrum Master, and the Scrum Team. The product owner decides the functionality the team should work on. The Scrum Master is responsible for optimizing the development process. The Team works on developing the solution for the Product Owner and has continuous contact with them (Glasspaper, n.d.). An important aspect of the structure of the development team is for them to be self managed, with the ability to solve the tasks as they see fit (Glasspaper, n.d.). As an agile method, the Scrum process is divided into sprints. Each sprint focuses on high quality. This means that the sprints should be planned so that the functionality could be put into production once the sprint is over (Glasspaper, n.d.). Other examples of agile methods are XP, TDD, Kanban and Domain-Driven Design

2.1.2 Prototyping

A prototype is the simple first design of a product which lets the developers test the product, before much work is done. Prototypes are a way to explore design, data content, functionality, as well as user interaction with the product. These tests can give feedback to the developers on how they should implement their ideas, in addition to which ideas are worth implementing.

The focus of prototyping is to quickly, easily, and cost effectively come up with ideas and iterate over them based on the feedback from tests. Making these decisions early in the development process has a great effect on the finished product. User experience experiences the biggest improvements when usability data is gathered early in the project (Nielsen, 2003). Factors such as time and money are also important in prototyping. Implementing change during the prototyping phase of the project is much quicker than if it were implemented at a later stage, such as during coding. This can also save a considerable amount of money, as code will not have to be refactored and changed (Nielsen, 2003). One of the most cost-effective prototypes is a paper prototype. This is simply a sketch on paper, drawn by hand that still tests all the basic functionality and design.

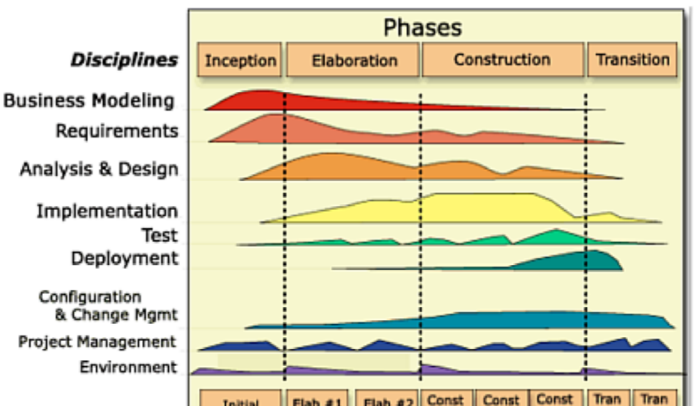
We discern between high- and low fidelity prototypes. High fidelity prototypes are more detailed and contain more functionality than those of low fidelity. In software engineering a low fidelity prototype may be a paper wireframe, while a high fidelity prototype may be a website that users can interact with.

2.1.3 UP – Unified Process

UP is an iterative and incremental development method where the work is done in cycles, a single task can be repeated, and where the developers generally can go back in the program and make changes based on experiences gained along the way.

UP uses a case driven model that puts the users' needs in focus. The technique is used to identify functional requirements and to describe how a user would interact with the system through user scenarios. This makes it easier to develop solutions tailored to the users actual needs. It also helps the developer team to keep focus on what is crucial and what is not, which gives the user a user friendly and efficient program. .

UML modeling is central in UP, as it is used to visually describe the system at different levels of detail. This helps the team to communicate and collaborate later, as well as identify the problem sooner.



Similar to the waterfall method, UP still uses phases like inception, elaboration, construction and transition, but specifies that the different types of disciplines can be performed over several phases (Norozi, 2023). Another important difference is the distinction between roles, which is stricter in the waterfall method, but that with UP allows any team member to be placed wherever they are needed, provided that the models and documentation are good.

2.2 Testing (Johnsen, 2023)

Testing is an important part of development, done to ensure that the finished product is as error-free as possible, does what it is supposed to, and to ensure that the stakeholders' needs and wishes are met. It also makes the development process faster. If the program is not good enough when it is finalized, it can lead to the loss of customers, profit, data and security.

There are many things that should be tested when developing a program. For example, that the program responds correctly to all types of input, has no bugs, performs well enough under high pressure and over time, is user-friendly, meets the requirements set by stakeholders, can be installed in the right environment and does not break when changed. How this is tested may vary. Before, it was common for development and testing to be separated into two different teams, but lately it has become more common to coordinate development and testing in what's called Agile Testing.

In order to save resources, time and costs, it is smart to make a plan on how and which tests you want to carry out. The image (figure 3, testing pyramid) visualizes how one can expect a test process to be carried out. You start at the bottom with many small, simple tests such as unit tests, before building up to more complex and time-consuming tests such as usability tests when the program is more complete.

2.2.1 Test Strategies

It can be difficult to figure out what should and shouldn't be tested in a project and normally it is not possible to test everything. It is therefore important to create good tests that cover the most important parts of the program. There are several different strategies that can be useful to think about in order to create good tests.

2.3.1.1 Black- and White-box

In a Black Box test, the internal structure is not known to the tester. It is only the input and the output from the system that is visible. This is useful as it allows the team to see the program from a different perspective. The counterpart is a White box test where the internal structure is known to the tester.

2.2.1.2 Functional- and Non-Functional

One should test both the functional and the non-functional parts of a program. A functional test tests what the program can do directly, while a non-functional test tests how well the program performs while doing the action. For example, that the program spends an acceptable amount of time on various actions.

2.2.1.3 Positive and Negative

It is important to ensure robust code by implementing both negative and positive tests. Positive tests test desired actions, where the unit runs as the developer intended. Negative tests test undesired or unexpected actions. These tests are made to ensure that the unit can handle unexpected cases.

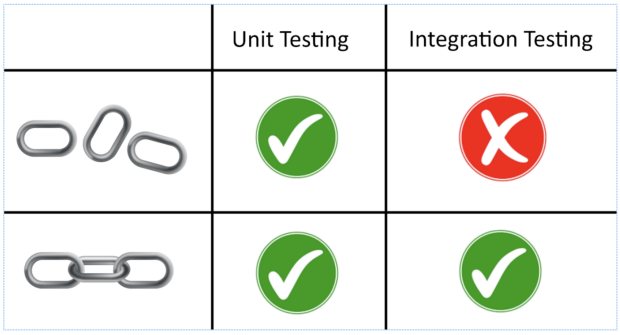
2.2.1.4 TDD and BDD

Test Driven Development (TDD) involves creating a test, then writing the code until the test works. Behavior driven development (BDD) is similar, but is more about automated acceptance tests described in domain-specific language. This could be for example UML use cases or Gherkin. One of the biggest advantages of BDD is that the tests are written in a more readable way. This means that not as much prior knowledge is required to understand what is being tested. In return, it is more resource-intensive than TDD.

2.2.2 Test Methods

2.2.2.1 Unit and Integration Testing

Unit testing is used to test individual parts, or units, of a software. These tests make sure that the units perform as expected on their own, before testing their *integration* with the system. Meaning, testing how and if the different individual parts work together. This procedure takes place at the code level of the development process (Hamilton, 2023).



(The top line is a software where only unit testing has been performed, the bottom line is a software where unit testing and integration testing are combined)

2.2.2.2 Smoke, Sanity, Regression Tests

A smoke test checks whether the basic functions of a program work as they should. The test is mostly carried out at the start of the project.

Sanity Testing is done later in the project after the program has started to be more coherent and become more stable. The point of the tests is to check whether small updates work as they should and do not introduce any obvious new errors.

Regression Testing is a thorough test of the entire system that runs all previously written tests. The testing is done to check if recent updates have affected the program in a negative way, and generally to check that everything works as it should.

2.2.2.3 Performance Testing

Performance testing is a test that evaluates how a system preforms in term of responsiveness and stability under various workloads and stress levels. The goal of this type of test is to identify performance speed, robustness, reliability and application size, in order to ensure the system can handle the expected level of user traffic.

2.2.2.4 Usability Testing and User Acceptance Testing (UT & UAT)

Physical products and digital interfaces are becoming increasingly interconnected. This means that the demands placed on digital devices regarding intuitive design and being easy to use becomes greater and greater. To ensure this, it is important that programs are tested against real users in what is called a usability test. The process takes place when a moderator records the user behavior, user suggestions, pain-points, feedback and areas of concern from a user while they use the program.

Doing a user test allows the developer to take a step back and see the program from new sides. The feedback is particularly useful for the visuals and to check that the program is easy to use. You also gain an understanding of how people use the program, and whether this matches the expectations the developer has. User tests can be run several times throughout the project. For example, in connection with Wireframe, Minimal viable product (MVP) and the final product. It is also common to carry out a comprehensive test at the end of the project called a User Acceptance test (UAT). It focuses more on the entire functionality of the software application.

User tests are often divided into two main categories: qualitative and quantitative testing. Quantitative testing is performed by formulating test questions with short, measurable answers. This same test is then carried out on a substantial number of individuals, with the goal of gathering data to analyze. This data offers the advantage of statistical analysis, often useful for market research. However, this method may miss important test data that can not be conveyed through standardized questions and answers. Qualitative testing, on the other hand, performs testing on fewer individuals, while gathering more data from each tester. The advantage to this kind of testing is the increased understanding of the individual tester, and potentially a better understanding of the user experience. On the other hand, these tests can be time consuming, and limit the number of tests.

2.2.2.5 Test Coverage

Test coverage is a technique that measures the amount of testing done on the software. To have enough test coverage it is important to first make tests for the crucial functions of a software, but also for other less crucial functions and features. If you test everything that needs to go smoothly for a software to work optimally, you have good test coverage. But there is also a thing such as too much test coverage, where there are too many tests on functions that need no use for a test or that have been tested indirectly through other tests. This will not affect the program, but is unnecessary as it is time consuming, and most of the errors that occur in most softwares can be found in only a small percentage of tests (Pareto principle).

2.3 Software Architecture and Design

Design architecture is a way to structure the code to make it robust, secure and easier to manage with a large code base. MVC (Model View Controller), is a well recognized design pattern commonly used for web-development and developing interfaces such as REST. The design pattern is split into three layers. Model, View and Controller. The model represents the object itself, and has no logic. It exists solely to manage the data. The view layer however, is responsible for representing the data to the user. The view layer itself can access the model data, but cannot manipulate it, only present it to the user. The controller layer is the connection between the model and view layer and provides the logic needed to manipulate, fetch or update data between the two layers.

2.4 Relational Database

Databases are a way to store persistent data. They allow for writing and querying data by using SQL. A DBMS is typically used to control the database (Oracle, n.d.). Relational databases help structure data based on the relation between related data points (Oracle, n.d.). In relational databases data is represented in a table, organized by rows and columns. The tables can have primary keys or foreign keys, which are used to demonstrate the relationships they have between each other (IBM, n.d.). The defined relationship between the tables allows for quick access of data (Microsoft Azure, n.d.). An RDBMS is used to manipulate data, provide structure, and manage the database. Relational databases provide a level of consistency between multiple instances of a database which ensures that each instance contains the same data as the other instances at all times (Oracle, n.d.).

2.5 Design

2.5.1 UX Design

User experience is the subjective emotions, thoughts and perceptions when using/interacting with an interface. User Experience (UX) design, however, is a discipline focused on designing with this in mind. UX design can be considered an umbrella term, encapsulating several other disciplines, such as Interaction design, Information architecture, Communication design, Human-Computer Interaction (HCI), and UI design, to name a few.

2.5.2 User Interface Design

The 5 dimensions of interaction design

1. Words
2. Visual representations
3. Physical objects or space
4. Time
5. Behavior

Above are the five languages, or dimensions, of interaction design, meant to highlight the wide range of

consideration an interaction designer must weigh. This includes, but is not limited to wording, images, icons, communication medium, tactile feedback, animation and feedback. To properly design an interaction between a user and a computer interface, all five dimensions should be considered and designed around (where applicable). (*The Five Languages or Dimensions of Interaction Design*, 2015)

Understanding the User

To increase understanding of how a product may be received, it is important to gain insight into potential target groups. Firstly, potential users' demographics are established, by collecting data on these potential users, through surveys. This data can be used to curate fictional personas to use in the design process, when making design decisions. These personas can be used to develop scenarios for expected use, such as *Use case diagrams* (see section 2.8 - UML).

Prototyping

To explore various designs and solutions it is important to develop prototypes. Doing this facilitates product testing at an early point in the development. This can save manpower as well as time, as it is possible to avoid working on designs that will later be discarded. The earliest prototypes are often "low fidelity" prototypes, made during the ideation phase. These are often in the form of sketches, and allow little to no user interaction or working functionality. After further development the interface can be tested through "high fidelity prototypes", also known as "wireframes". Simulation of the planned interface, with working functionality that allows interaction. These wireframes are often created in tools like Figma and Balsamiq.

Don Norman' Seven Design Principles for Interaction Design

Don Norman, a prominent name in the Interaction Design field, postulated seven principles of interaction design in the book "The Design of Everyday Things" from 2013 (Norman, 2013):

Discoverability - An interface should present its components clearly and distinctly, so the user can easily determine how they can be interacted with.

Feedback - Interacting with the interface [returns] feedback to inform the user that the action was successful, or, alternatively, why it was not.

Conceptual models - High level descriptions/models of a system's operation and possible uses.

Affordance - Connection between how components look and how they are used, prediction of users' interpretation of the system. Necessary to consider users' intuition and existing knowledge.

Signifiers - Ways to communicate where relevant actions are performed. It is important to consider potential users' already established contextual understanding to determine these signifiers.

Mapping - Coherence/relation between controls/interactive elements on the page, and the effect they have on other elements.

Constraints - Increasing usability by implementing constraints on user interaction
help restrict the kind of interactions that can take place.

Application

These seven principles can be used as tools to design and analyze user interaction, to increase understanding of how users may interpret and use various different types of interfaces. This can be used to improve the design of these products and services. Designers may use these postulations to formulate a plan to effectively communicate with users.

2.5.3 Universal Design - WCAG 2.1 Principles

Web Content Accessibility Guidelines (WCAG) 2.1, (Kirkpatrick et al., 2018) published by the World Wide Web Consortium (W3C) are recommendations for making website interfaces more accessible to individuals with various disabilities. The report details the three different levels defined, ranging from Level A to level AAA, level AAA

implying the highest degree of accessibility. Below are excerpts from these guidelines that are relevant to the technologies used in this project.

Perceivable

Information is not communicated in multiple ways, to avoid limiting the user to a single sense. Text and other visual media can be interpreted through text alternatives such as screen readers, videos can contain captions and audio descriptions. Should the application use a markup language, identifying the purpose of user interface components enables assistive technologies such as screen readers to interact with these components. When information is communicated visually, the elements should be designed to be easily distinguishable. Firstly, when using colors to convey information, the same information should also be accessible through other means, such as text or icons (Level A). Additionally, sufficient contrast between text and background improves the readability of the text, (Level A-AAA).

Operable

Operable applications are accessible applications. The page should provide functionality accessible through a keyboard interface without keyboard traps, where it is difficult or impossible to exit (Level A). This is often implemented as a way to navigate the page sequentially, but can be improved further by adding the possibility to bypass entire blocks. When implementing this, focus order should be considered (Level A).
Navigation of the interface should also be considered. Pages and blocks should be clearly titled, and the page navigable. The user should also be made aware of where they are located on the page at all times (Level AA-AAA).

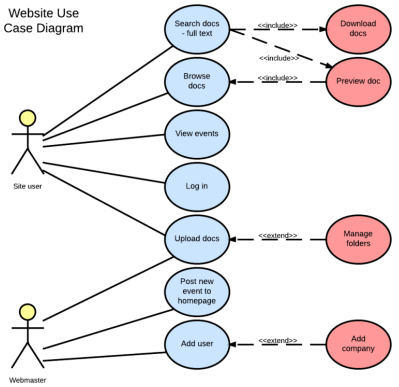
Understandable

To improve the understandability of a page and its content, the content should be understandable and consistent. Components being focused or receiving input should not initiate a change of context (Level A). Additionally, components should be designed consistently, and clearly communicate their purpose and interaction. Similarly, navigation should be consistent across the pages of a site (Level AA). Input assistance helps the user provide the page with necessary or relevant information, by providing labels or instructions when content requires user input (Level A), error identification (Level AA) and error prevention (Level AAA).

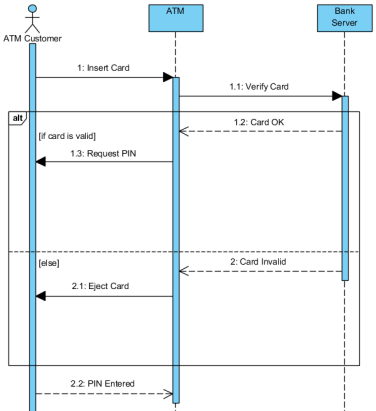
2.6 GDPR and Ethics

GDPR refers to a set of rules and regulations regarding data protection which applies to all EU- and EEA-countries (Datatilsynet, 2021). These regulations come into effect when personal data is handled automatically, as well as all data that is organized in a register (Datatilsynet, 2021). The GDPR sets a basis of ethical conduct which businesses have to follow. Not complying to these regulations will result in sanctions to the business. The EU can sanction the responsible parties through international law, but extensions of GDPR in local law can also result in local sanctions. NITO has a set of ethical practices that Norwegian engineers and technologists have to follow.

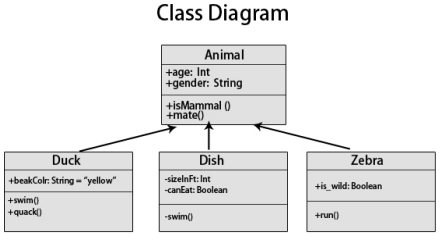
2.7 UML



According to Visual Paradigm; “UML is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting of software systems” (Visual Paradigm, n.d.). UML is a way for the development teams to easily communicate their ideas of how the different components in the software system function together. Creating models based on the UML can significantly lessen time spent developing the software, and therefore the money spent as well. The standard approach the UML provides allows the developers to focus on the conceptual work of the software, which also contributes to the time spent developing.

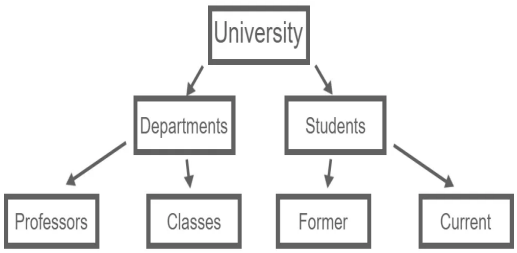


The Use Case diagram is a commonly used UML diagram, which helps visualize the different ways a user might interact with the product. This diagram is a valuable tool for developers and external parties, such as the product owner, to communicate how they think the product should function. It visualizes the way an actor in the



system might use it to reach their desired goal (Visual Paradigm, n.d.).

Class diagrams are a modeling technique used by the development team to conceptualize the relationships between classes.



Sequence diagrams visualize how a method in one class interacts with other classes through its life cycle.

Database models are used to model the relationship between entities, such as tables, in the database.

Domain models are a simple way to visualize and understand how different entities of a system interact with each other. This type of model is typically used by developers to communicate with external stakeholders.

2.8 User Authentication

Security is an important section of software development, and when dealing with user information it is of high priority that only they can access their data. Tokens are pieces of data that only have meaning if paired with the correct tokenization system (Okta, n.d.). Token based authentication is a way to authenticate requests made from a client to a server. Using this type of authentication ensures that only the data with the correct token results in a valid response from the server. JWTs transmits data as an encoded JSON object. This token is made up of three parts: the header, payload and signature.

2.9 Caching

Caching is used to store data which is frequently accessed to improve the speed at which the data is accessed. It retrieves the data faster by not having to access the data's primary storage location (AWS, n.d.). Caching can greatly reduce the time spent retrieving data when dealing with external storage locations such as databases (AZURE, n.d.). Caches are typically either private or shared. A private cache stores data locally, usually in the RAM (AZURE, n.d.). This type of cache is limited by the amount of memory the local machine has, which means problems can occur if the amount of cached data is too large. Private caching can also result in inconsistencies between clients as they do not share changes made in the local cache (AZURE, n.d.). As an alternative, shared caching can help with the problems that come with the inconsistencies in private caching, in addition to providing a larger storage capacity. Shared caching does however reduce the access time compared to private caching. A higher level of complexity is added, as shared caching deals with information between machines (AZURE, n.d.).

2.10 Relevant Literature

"Introduksjon til interaksjonsdesign" (Nordbø 2017)

"Interaksjonsdesign is a book by Tone Nordbø that talks about several relevant topics on being a software developer, with an extra focus on interaction design.

Eden coding

Eden coding (Eden coding 2023, *https://edencoding.com/*) is a site that has methods regarding GUI applications. The site answers common questions, has theory regarding the basic JavaFX functions and how to use them, and how to make design changes to create better looking GUI . The site is made by a developer and analyst named Ed Eden-Rump.

BroCode

BroCode (BroCode 2023, *https://www.youtube.com/@BroCodez*) is a YouTube channel that provides videos on various programming languages. These videos are useful as they answer a number of questions about coding with great detail and visuals.

Stack Overflow

Stack Overflow (Stack Overflow 2023, *https://stackoverflow.com/*) is a well known site for its code related content. The site allows users to ask questions that are answered by other users, and also allows other non users that have a similar problems, see examples on possible solutions. While this site is very useful to test possible solutions to coding, the theory that is posted and/or the way that the code is written, is not necessary best practice as anyone can reply to the submitted questions. Therefore it is important to be critical to this information.

3 Method

Work and Role Distribution

The team decided early on that a flat structure, following the Scrum method (reference to theory?), would be the best way to organize. This meant that no one should do considerably more work in one part of the project compared to others. Programming tasks, documentation tasks, and reporting tasks were all part of a collaborative effort. We did not allocate this type of work based on predefined roles, but kept to the UP method and allocated work based on what needed to be done in the sprint.

To ensure that all required tasks of the project were met supervisory roles were given to all the team members. Each team member supervised two or more specific aspects of the project. These supervisory roles consisted of Product Design supervisor, Quality Assurance supervisor, Lecturer contact, Meeting Organization supervisor, Document supervisor, Submissions supervisor, Test supervisor, and Referent. Other, non supervisory roles were HR, Chairman, Scrum Master, and Client Contact. Each supervisor had the responsibility to ensure that the tasks regarding their specific role were carried out. They were not responsible for doing these tasks on their own. The roles were distributed to the team members based on previous experience and individual wishes.

Product design

Design specifications are detailed in the Design Document, see appendix - Design Document

GDPR and Ethics

Developing a system for a bank has put a focus on the importance of data security and ethics. The team spent a considerable amount of time discussing the best solution to keep the system secure and follow GDPR regulations, as well as Norwegian law. Users can use the system by creating an account using an email address and a password. Data such as name, location, age, or IP-address is not registered anywhere. The data registered to the user which we store is information manually added by the user themselves. Having this data manually registered is one of the ways in which we comply with the GDPR. Measures were also taken to ensure all data and user information was securely stored, with only the user themselves having access to it. We did this by tokenization, bla bla bla, need some theory

Prototyping

The team started the prototyping process by creating low fidelity prototypes based on their efficiency as described in the theory section on prototyping. Each team member created their own prototype with their ideas of how the functionality of the product should work, as well as a basic design of the product. The work done during this iteration was done before we had a formal client to work with, which meant that the features we tested were less specific than what they were in future iterations.
The team then used the best elements in this first iteration of prototypes to create a wireframe using the software BalsamiQ. The wireframe provided a higher level of fidelity, making it more similar to how the application would look and feel. It was important to keep a consistent design for the second iteration of prototypes. This meant that the colors, fonts, buttons and containers all had a similar design. The team also had to make sure that the design was consistent with the design of BN Bank's website. As a result of this, feedback from user tests regarding core design elements of BN Bank had to be discarded, because we had to keep a certain level of design consistency with our client. Once the wireframe had gone through a couple iterations the team scheduled a meeting with our client for

final feedback. The feedback we received from our client was of great help as it helped focus our attention on the functionality they deemed most important. This was especially helpful when we started work on the MVP with limited time.

Databases

The team decided to use a database for storing the data for the project assignment. This decision was made early by the team, as it would not be responsible to store transaction data and bankdata locally. Using a database would provide the application an extra layer of security, as the data is not stored locally and would also make the application more flexible, as users can access the application with the same data from different IOT devices.
Our application is going to handle income data, expense data, budget data and user data, and for that, a relational database provides an efficient way to store, fetch and query that information. The scrum master of the team created a MySql database from NTNU. The tables were split into expenses, income, budget and user. There was also going to be a table for categories to normalize the database more. This has not been implemented for this time, as it would require a lot more work to implement.

Backend

The team decided to split up the program into an application that fetches and requests data from a backend. The backend is supposed to create a layer for interacting with the user and the database. This creates a layer of security as the user cannot interact with the database itself. It allows us also to set up authentication and allow user specific resource allocation. With the backend hosted on a production server, the application can also be used by multiple users simultaneously from different locations and IOT devices.

Caching

Loading in data takes resources from both the backend and the frontend. Sending a multitude of GET requests can make the application slow, and will generate duplicates of the same data. Caching is a good and efficient way to only load the data one, and then update the cache registry and the database when data is changed. This was done in the application by creating a singleton class responsible for caching. The singleton class and a cache class. The singleton class is responsible for getting the same instance and holding the cached data, which can be updated and fetched from multiple classes. The cache class simply stores the different data objects in a hashmap with a descriptive key that can be used to retrieve the data. After implementing this, the group noticed a remarkable difference in load speed when changing pages, as the application did not fetch the same data each time when the user was going to a new page.

User Authentication

When dealing with sensitive data, authentication and user based resource allocation is essential. There are two important things to remember. The authentication of the user, and the resources they have access to. One way to do this is for the backend to compare user credentials with the database and then give out access based on the user id. This can bring a security risk if the password is not hashed in the database. This method also has potential security risks. Authentication based on a string user id can be modified in a post request. We decided to use JWT user authentication. When a user logs into the application, a JWT bearer token is generated with the user's credentials. The users can then use that token to gain access to user specific resources. The token has a payload of the expiration date of the token and the user id. That token is then validated and decrypted by the backend each time the user makes a request.

4 Results

This section describes the results of the project. It is divided into two parts: project results and administrative results. The project results describe the features of the product and their state at the end of delivery. The administrative results describe the goal fulfillment according to the project plan, as well as the process goals described in the vision document.

Project Results

No.	Feature	Priority	Result
1	Check if users will go over budget	Medium	Partially completed

2	Login	High	Completed
3	Display budget	High	Completed
4	Change budget	High	Completed
5	Visualized economic overview	High	Completed
6	Display all transactions	High	Completed
7	Edit transactions	High	Completed
8	Overview of users economic history	High	Partially completed
9	Manually add transactions	High	Completed
10	Automatically add transactions	Low	Incomplete
11	Retrieve market information on prices	Medium	Incomplete
12	Advisor	Medium	Incomplete
13	Demo view	Low	Incomplete

In collaboration with our client we have created a detailed feature list, as shown in the table above (figure 9), of all the desired functionality of the product. A more in depth explanation of each feature is listed in the vision document (see Appendix - Vision Document). The team attributed each feature with a level of priority, depending on the feasibility of implementation within the given time frame. The degree to which features were considered to be “core-features" also had an impact on priority. Core-features were seen as high priority, while more complex features that required access to external sources or APIs were seen as low priority.

State of Features at the Time of Delivery:

1. Check if users will go over budget. This feature was

2. Login. The login feature was prioritized highley for multiple reasons. The team wanted to best simulate the experience of using a bank service, and the ability to have your own user was a big part of this. This naturally meant that a login system had to be created. The security aspect of a bank service was also a central part of the project, which we could not have done successfully without a login feature. Much of the initial work was spent on creating a simple login page, which meant that it was created early on in the process. The team spent time iterating over the login feature, making it more responsive and secure.

3. Display Budget. The display of a users budget was seen as a core-feature and was given high priority. This feature included a list of the budget categories, as well as a visualization of the budget. This feature went through many iterations of user tests, as well as iterations from the team. The final version of this feature worked as desired.

4. Change Budget. This feature included the ability to edit individual elements of the budget, as well as deleting them, and was considered a core-feature. This was a simple feature to implement, and was therefore implemented early in the development process.

5. Visualized economic overview. We planned for the product to visualize the users incomes and expenses graphically, represented by pie charts, bubble charts and line charts. During development we decided to exclude bubble charts from the list, and keep a consistent design by just using pie charts and line charts. We still consider this feature to be successfully implemented, since it was a conscious design decision to not

use bubble charts.

6. Display all transactions. This feature was considered a core-feature, and one of the first to be fully implemented.

7. Edit Transactions. Incase of wrong information given by the user, the need for editing transitions became a high priority. The implementation of this was successful upon delivery. The team did however come to the conclusion that a part of the edit functionality should be a feature to delete transactions, which we did not implement.

8. Overview of users' economic history. This feature was implemented as part of the users' economic overview. The team also discussed implementing this at the display of budget and transactions, but this was not done.

9. Manually add transactions. One of the highest priority features, as most other features rely on transactions being added. It was therefore implemented early on.

10. Automatically add transactions. The client saw this as an important feature in a real implementation of our product. But since it was quite a complex feature that would require time and special legal access we decided it was of low priority and subsequently not implemented.

11. Retrieve market information on prices. This feature was of great interest for our client, and we discussed the possibility of getting access to APIs through our client which would allow us to implement this feature. That resulted in this feature being considered medium priority, because a lot of complexity was still involved. In the end it was not implemented, as we focused more on the core-features of the program.

12. Advisor. The advisor was a feature that would use most of the other features as a way to give financial help to the users. It would implement both low, medium, and high priority features. The team put the advisor feature as medium priority, as we reckoned some of the simple functionality could be implemented without implementing the complex functionality. In the end none of the functionality of the advisor feature was implemented.

Feedback from final user tests

In the vision document, we described the result goals as the delivery of a finished product according to the vision document, an application which will visualize and simplify personal economy, and provide financial advice based on the individual user (see Appendix - Vision Document). Our client found user engagement to be the most important factor for this product. The simple visualization and easy to understand financial advice was our way of solving creating an engaging application. During our final round of user tests we received comments from three testers who asked if it was possible to download the application while eagerly grabbing their phones, because it was something that would be really useful to them when fully finished. (see Appendix - User Tests) This feedback indicated that the product was engaging.

Accessibility Results

Colors	Contrast Ratio	WCAG Level, Normal Text	WCAG Level, Large Text
White text on red background	5.13:1	AA	AAA
White text on dark red background	5.87:1	AA	AAA
White text on dark gray background	5.42:1	AAA	AAA

Black text on light blue background	17.79:1	AAA	AAA

- Screen reader: supported, but lacking functionality.
- Information communicated through color: was planned to be included in the chart functionalities, but did not get completed in time, as core functionality was prioritized.
- Navigation: consistently achieved through navigation bar and popup windows. All pages now contain a header, to inform the user of their current location.

Administrative Results

The team created a set of process goals which can be found in the vision document. (Appendix - Vision Document). The process goals focused on making personal improvements in software development, cooperation and teamwork, and gaining experience in project work for an external client. During the project we have had to balance learning and finishing tasks on time. This has led to an improved understanding within the team of how to effectively organize teams and delegate work to ensure timely delivery. Throughout the project, the group has shared knowledge with each other to improve the whole of the group.

The overarching milestone of each iteration were unchanged, while less critical milestones were subject to change. We followed the GANTT diagram as much as possible, but as problems arose under development it became difficult to stick to the plan. Results of this were mostly delays in beginning new tasks, as we focused more on fixing the problems and completing the unfinished tasks.

As a result of our client being a business we have received much experience working with an external client. During this project we have gotten experience on starting a collaboration process with a client, and communicating with them to achieve the best possible end result.

5 Discussion

This section provides a discussion and analysis around the results of the project.

Product Requirements

As described in the result section, most of the high priority features were successfully added. During the first meeting with our client we discussed many possibilities of what our product could be. Many of these suggestions were of high complexity and either required access to external products or certificates for lawful conduct. The team decided early on that most of these features were not feasible to implement given the time, resources or knowledge we possessed. As such, we decided that the core-functionality of the application was the most essential part. The feature list was therefore created with the core-functionality in mind.

We still wanted to be able to implement some of the special features the client wished for. Features such as “Advisor” and “Retrive market information” were therefore marked as low priority. During this initial stage of development we planned for these features to possibly be implemented after all the core-functionality was complete. However, we quickly realized that the core-functionality would take up most of the development time, and so the low priority features were disregarded.

The feature “Automatically add transactions” was not considered to be part of the final implementation, but rather to be used as a focus in further work. This feature could only be implemented in an ethical and lawful manner if the system was completely integrated with BN Banks systems. In retrospect the features that would not be possible to implement during this project should not have been included in the vision document.

In addition to focusing on low-priority features we focused on the safety of data transfer. Potential time we could spend on higher priority features instead would be spent on implementing user authentication.

Process and Procedure

The team was quick to create plans for the project. Initially the focus was set on creating formal documents. That meant the collaboration agreement was one of the first parts to be written. The collaboration agreement went through a few iterations before it was finalized. The first of these iterations was too vague and did not include sufficient meeting standards or a penalty system. Though the team started work early on the collaboration agreement it took considerable time to finish and sign it.

A general project plan was set up quite early, however it was not as detailed or accurate as the GANTT-diagram would become. The first iteration of this plan included time for discussion on what features a program such as ours could include, but was limited to quite basic features, as we did not yet have a client. The GANTT-diagram was planned in such a way that all the milestones of each iteration, according to the assignment description, were completed before the end of the iteration. More detail was added in order to create a clear structure that we could plan our sprints around. As a result of the team taking a long time to find a client most of the beginnings stages planned in the GANTT-diagram were pushed back. The team felt it was more important to start the development of the high-fidelity prototype and the requirement list in the vision document after we had landed on a client. The report was therefore also delayed. Instead we focused work on the more general parts of the vision document, and made a solid collaboration agreement.

After putting in the work of creating a collaboration agreement, and discussing what our product could be we started researching and finding a possible client for our product. We were in agreement that the client should be a business and not a private individual, as this would give us a lot of relevant experience. The process of finding a serious client and getting in contact with them proved to be a long process. This led to a late start in the development of the prototype, vision document and subsequently the MVP. If the team had decided on a private individual as the client these parts of the project might have started quicker, but the relevant work experience may not have been as prevalent.

Most of the work done on the vision document was done after the first meeting with the client. After this meeting we began creating the requirement list as well as a risk analysis. (Appendix - Vision Document.) The risk analysis was separated into the event, the likelihood of the event, the impact it would have on the final product, and preventative measures we could take. One of the five risks accounted for fully occurred. This was “Access to APIs” which was put at a medium likelihood of happening. No external APIs were used in the final product since these features were complex and getting access to the APIs was a difficult process. Risks such as “Data security” and “Time constraints” partially occurred. “Data security” concerned itself with access to proper data from the client. We followed the preventative measure for this risk however. We decided to generate our own test data, instead of relying on our client. If we had gotten access to test data from the client we could have developed our application to more easily be integrated with theirs, but since this project was considered a prototype or a test for future work it was decided that our generated test data was sufficient. “Time constraints” focused on the possibility of an incomplete product. This is considered to have partially occurred since the low-priority features were not implemented. We still took the preventative measures into account by making these low priority to start with.

The product was required to use JavaFX for the GUI, as per the project assignment. The team used a programmatic approach to JavaFX, which differs from an XML-style approach using FXML. This resulted in issues with connecting the different pages in our application together in a safe and secure manner. As a result of this the team had to switch the JavaFX approach to FXML. Much unnecessary work could have been avoided had this issue been discovered early on in the development process. This was however discovered one week before the MVP had to be finished. Most of the sprints that were planned had to be discarded since the code had to be rewritten.

The team used a variety of collaboration tools during the project. We still wanted to keep the amount of tools we used to a minimum. This was to avoid confusion, the risk of not using one of the programs and to keep all the team members engaged. Discord was the main channel of communication between team members. We created our own server and added multiple text channels that we could use for different parts of the project. Notion in collaboration with Google Disk was used to keep track of documents such as the report, vision document, collaboration agreement and different diagrams. GitLab was mainly used for code. Because Discord was so frequently used we decided to do our daily stand-ups there. We started the first iteration by using the GitLab issue board but found that creating the issue board alongside the other documents in Notion was more effective. As limited time became a big factor in our project we decided that we would focus on creating the issue boards as effectively as we could, so we switched completely over to Notion for issue boards. an example of the Discord issue board can be found in the appendix (Appendix - Issue Board)

6 Conclusion and Further Work

6.1 Conclusion

The final program is not quite where the team wanted it to be at this stage of the project, for several reasons. Yet, the team is mostly satisfied with the core functionality of the program. The most important of the features were added, and further implementations have been taken into consideration and would be made possible if the project were to continue. Looking back at our client's wishes, most of the rational functions were added, but as we struggled to make contact with the client at the end of the project, we could not get specific feedback. The usability tests on the other hand were more positive than expected, which the team is happy with.

The project has given the team valuable experience regarding working with clients and in a team, as well as theoretical knowledge regarding the topics. As the team made progress we quickly learned what worked, but more importantly, what did not work. The team is certain that if the same project were to be given again the knowledge that we gained from this project, would help us manage time, tasks, problems and generally unknown theretory better.

6.2 Further work

Even though the program has integrated many things, there is a lot that can be improved. Based on the user tests we have run and the team's own experiences, a number of things have emerged that should be worked on if the project were to continue.

One of the most demanding tasks for further development will be to integrate the application with various banking services. In order to have a secure system, it will be necessary to use bank ID as login. This will make registration of new users unnecessary in the existing program. The main benefit of connecting with the banksystem will be to automate transactions. This will make the product much easier to use and reduce the margin of error.

To improve the transaction page, the team recommends making it possible to enter earnings into different accounts. It should also be possible to transfer money between private accounts. New transactions should also end up at the top of the list when they are added.

The budget and overview page of the program received some feedback on the layout. To solve this, it can be smart to go through the test data from the usability tests and make changes based on this. Specific visual improvements to the charts could be adding animations when hovering over the different parts so the pieces move a bit away from the chart. Another would be showing the sum/percentage the chartpiece presents when hovered on. Other things that could improve the budget page are a field that shows how much of the budget has been used while making the budget, so that the user does not set up a budget higher than their income.

The program should get a working tab navigation system to satisfy requirements for Universal design. This can also make it generally easier to use the program. One of the most important things to add to the program is the ability to delete budget and transaction elements. Without this function it will be difficult for the user to manage the application. The possibility of adding a savings goal should be considered. This can, for example, be added in its own window or as a side product on one of the already existing pages. The option to download a pdf with information from the program should also be added.

BN Bank showed a lot of interest in the assistant feature during our meetings with them. An implementation of this would build on top of the already implemented features. The functionality of the assistant would help BN Bank stand from their competitors. As described in the sections above this feature could be implemented in stages. This includes the ability to retrieve market information and adjust the budget for inflation, check electricity prices, and compare the transactions history to check if the user will go over budget.

The bar chart in the overview page should be clickable, in a way that allows the user to view their expenses for the selected month. For improved accessibility the line charts should be named to accompany color blind people. Here it would also be possible to get to the site that summarizes the selected month, and that can convert the data into a summarized PDF file.

Better universal design when it comes to the line chart in the overview page. The first improvement would be to name the x and y axis. The second would be to name the different lines. The line chart shall later on add the possibility for viewing several months at a time, which makes these changes crucial for any person, and not only one with vision impairment.

Repository

Link to public GitLab repository: https://gitlab.stud.idi.ntnu.no/group-6/budgetplan
Link to wiki on GitLab repository: https://gitlab.stud.idi.ntnu.no/group-6/budgetplan/-/wikis/home
Link to issueboard on GitLab: https://gitlab.stud.idi.ntnu.no/group-6/budgetplan/-/issues
Link to JavaDoc for final iteration, front-end: https://group-6.pages.stud.idi.ntnu.no/budgetplan/javadoc/
Link to JavaDoc for final iteration, back-end: https://group-6.pages.stud.idi.ntnu.no/budgetplan/JavadocBackend/

References

AWS. (n.d.). *What is Caching and How it Works | AWS*. Amazon AWS. Retrieved April 25, 2023, from https://aws.amazon.com/caching/

AZURE. (n.d.). *Caching guidance - Azure Architecture Center*. Microsoft Learn. Retrieved April 25, 2023, from https://learn.microsoft.com/en-us/azure/architecture/best-practices/caching

Datatilsynet. (2021, 10 12). *Om personopplysningsloven med forordning og når den gjelder*. Datatilsynet. Retrieved April 24, 2023, from https://www.datatilsynet.no/regelverk-og-verktoy/lover-og-regler/om-personopplysningsloven-og-nar-den-gjelder/

Glasspaper. (n.d.). *Hva er egentlig Scrum?* Glasspaper. Retrieved April 23, 2023, from https://www.glasspaper.no/artikkel/hva-er-egentlig-scrum/

Hamilton, T. (2023, March 4). *Unit Testing Tutorial – What is, Types & Test Example*. Guru99. Retrieved April 19, 2023, from https://www.guru99.com/unit-testing-guide.html

IBM. (n.d.). *What is a relational database?* IBM. Retrieved April 19, 2023, from https://www.ibm.com/topics/relational-databases

Microsoft Azure. (n.d.). *What is a Relational Database Management System?* Microsoft Azure. Retrieved April 19, 2023, from https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-relational-database/

Nielsen, J. (2003, April 13). *Paper Prototyping: Getting User Data Before You Code*. Nielsen Norman Group. Retrieved April 23, 2023, from https://www.nngroup.com/articles/paper-prototyping/

Okta. (n.d.). *Token Based Authentication Made Easy*. Auth0. Retrieved April 25, 2023, from

https://auth0.com/learn/token-based-authentication-made-easy

Oracle. (n.d.). *What Is a Database*. Oracle. Retrieved April 19, 2023, from https://www.oracle.com/database/what-is-database/

Oracle. (n.d.). *What Is a Relational Database*. Oracle. Retrieved April 19, 2023, from https://www.oracle.com/database/what-is-a-relational-database/

Visual Paradigm. (n.d.). *What is Unified Modeling Language (UML)?* Visual Paradigm. Retrieved April 25, 2023, from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/

https://www.uio.no/studier/emner/matnat/ifi/INF1050/v14/timeplan/inf1050.smidig.22.1.2014.pdf

https://www.edureka.co/blog/types-of-software-testing/ (Figure 3)

https://www.guru99.com/test-coverage-in-software-testing.html (Figure 4)

https://www.google.com/search?q=use+case+diagram&rlz=1C1VDKB_enNO1020NO1020&sxsrf=APwXEdewpoqWQWMrB119M4nEzBvv6ngaKQ:1682605293004&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiu9KSWocr-AhWQSvEDHbCUDDwQ_AUoAXoECAEQAw&biw=1152&bih=599&dpr=2.5#imgrc=lL4H4VmsQwWI9M (Figure 5)

https://www.google.com/search?q=class+diagram+easy&rlz=1C1VDKB_enNO1020NO1020&sxsrf=APwXEdd586udtCRiouwSCi6qgBg1LHfyRg:1682605334728&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjK65eqocr-AhVfbvEDHRUgCxoQ_AUoAXoECAEQAw&biw=1152&bih=599&dpr=2.5#imgrc=1qNIkRjyZKvjaM (Figure 6)

https://www.google.com/search?q=sequence+diagram&rlz=1C1VDKB_enNO1020NO1020&sxsrf=APwXEddvBiJkhq8xCgxxZj52ZFVS66eDkw:1682605370399&source=lnms&tbm=isch&sa=X&ved=2ahUKEwih-Ji7ocr-AhXpSvEDHZeQC0oQ_AUoAXoECAEQAw&biw=1152&bih=599&dpr=2.5#imgrc=YhAOeS9chHXGQM (Figure 7)

Appendices:

- idata1002_2023[SK1] _appendix_0006.pdf