University College Nordjylland
Top-Up Degree GTL Project
1st Semester

Ares
(Georgia Tech Library Project)

Collaboration with: Georgia Tech Library University Library
Participants: Ralfs Zangis, Andrei-Eugen Birta, Ádám Blázsek
Supervisor: Nadeem Iftikhar, Brian Hvarregaard, Jakob Bjerggaard, Simon Schmidt-Jakobsen
Submission date: 27-May-2019

1. Introduction

1.1. Problem statement

Georgia Tech Library (GTL) is a facility with nearly 16 000 members and has about 100 000 titles. After carefully reviewing the provided case description of what GTL is and what their interests are we have noticed there was no mention of a currently implemented software solution, which we have identified as the main problem the facility is currently suffering from. We decided that with the help of a software solution, the library would not only be able to conduct their routine tasks, like the management of: books, maps, and other rare materials and lending them to their registered members; but also gain some insight to their users reading behavior.

On top of that, we decided to "pinpoint" and highlight some specific questions that our solution would try to answer:

- What is the average loan time?
- What are the most frequently loaned books?
- What storage environment would be fit for such a solution?
- How can lendable and not lendable materials be easier to distinguish for the library staff?
- How can we automate the process of lending books?

Although we tried to contact GTL, in order to gain some more information about the way things are handled and how they deal with business, but we were sadly not given any response thus we were forced to use our own imagination and assume what was needed to be done and how. All assumptions are written down in appropriate places for appropriate decision.

In this report we will detail our solution to Georgia Tech Library's problems, walking the reader through the different challenges we have encountered and how we have chosen our solution with the relentless research and persistence.

1.2. Proposed solution

The solution offered to the library is a software solution, that would allow the librarians to better manage the library's materials, using an interactive and intuitive user interface. On top of that, it allows information transfer between multiple libraries to be possible, thus accommodating book exchange for the involved parties and exposing statistics of interest. Furthermore, the solution was required to allow people to register and be able to lend certain number of books and return them after the member has stopped using them. One last important function is the notification system which alerts concerned parties in case a material is not returned in time.

We decided to call this project Ares since Ares fits the team's habit of naming projects, based on Greek gods, and this project's theme of triumphing Fear, Terror, and Discord. All of which are related to Ares. Triumph that requires a great amount of courage and bravery, all of which are key elements of Agile development.

To do already expressed goals the group decided to create a distributed system using WCF application. The work was done using C# which is one of the most popular and versatile programming languages commonly used for tasks like this. Information was saved in Microsoft SQL Server using a relational database that would provide adequate storage with appropriate response times for the solution, for the

following 30 years[1]. To display the information to the user, we decided to create a web client using MVC, which is a great architectural pattern with many benefits.

Although we did not have a representative of the library to review the solution in the making, we believe that the current product will be fulfilling the needs of the institution for the perceivable future.

1.3. Facts about the library

- The library has: 16 000 members, 100 000 titles, 250 000 volumes (with an average of 2.5 copies per book)
- 10 percent of the volumes are out on loan at any time
- Most overdue books are returned within a month of the due date. Approximately 5 percent of the overdue books are never returned
- A library card is good for four years
- Members usually return books within three to four weeks. Most members try to return books before the grace period ends. About 5 percent of the members must be sent reminders to return books
- The most active members of the library are defined as those who borrow books at least ten times during the year. The top 1 percent of membership does 15 percent of the borrowing. The top 10 percent of the membership does 40 percent of the borrowing. About 20 percent of the members are totally inactive in that they are members who never borrow
- Staff of library: chief librarian, departmental associate librarians, reference librarians, check-out staff, and library assistants
- The library does not lend reference books, rare books and maps
- Some books may have the same title; therefore, the title cannot be used as a means of identification. Every book is identified by its International Standard Book Number (ISBN), a unique international code assigned to all books. Two books with the same title can have different ISBNs if they are in different languages or have different bindings (hardcover or softcover). Various editions of the same book have different ISBNs
- GTL also cooperates with four other tech libraries. Each library has its own service that it makes available to all the other tech libraries. The different libraries might have different rules for book acquirements and GTL must negotiate agreements with each library

2. System development

2.1. Stakeholders analysis

In this section we will go into greater details describing the various stakeholders of the final software solution. In total, there are four stakeholders which we managed to identify:

1. Georgia Tech Library – One of the stakeholders who has the most power and interest over the application. Its aim is to improve the current way of handling books and gathering statistical information about the library usage, therefor placed in the red area
2. The librarians – The second most influential and interested entities, their interest is in making their work easier and faster by the help of the application
3. Members of the library – Students or Professors who are part of the library both benefit from the software solution as it would be easier to keep track of their loans or the books they want to rent, although they have a high interest in the solution, they have a considerably smaller influence on the final product compared to the librarians and GTL itself
4. Other libraries – Also of high interest but low on influence, the other libraries are looking to ease and speed up communication between them and GTL

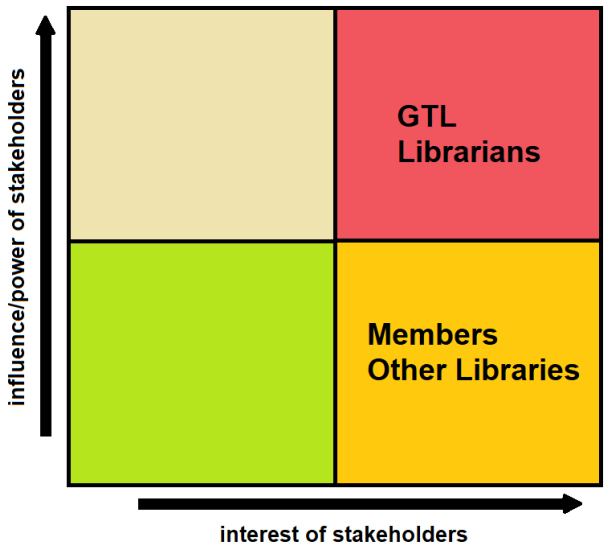


Figure 1 (Stakeholders analysis)

2.2. User stories

In order to get a better grasp of the functional requirements, we decided to create a user story table, and we did that by determining the actors of our application. Afterwards we have written down the actions and events that those actors would be able to interact with.

When considering the user stories, we took inspiration from our past experiences of using libraries and the given problem statement while we looked at the possible scenarios the different actors might go through.

After we finished with the user stories, we decided to prioritize them. Priority assessment is a key procedure when talking about user stories as it will dictate the direction of the application development. We were extra cautious to correctly assign the correct priority to each task. As mentioned in the previous section, we incorporated our experience as a compass to select the correct risk/priority levels for each user story.

The following parts are mere explanations of some of the user stories that can be seen in figure 46 (found in the appendix).

- Expose statistics based on all libraries: for example, the average loan time which the university has lent out most books or which book (or top 10 books) are the most loaned out
- Books**:** are entities which are identified by their ISBN, for each title there is a description in the catalogue

- Librarians**:** actors, who for example ensure that the books that members want to borrow are available
- Applicant**:** an entity who is not yet part of the library but is in the process of becoming so (by filling out a form which is reviewed by the librarians who decide if the applicant is eligible to become a member)
- Member**:** entity who is already part of the library, members can check out books (but no more than 5 at a time) in the period of 21 days, they also have a card renewal process which they must go through to remain a member. A member can either be a student or a faculty member (professors). Students have a single week of grace period.
- Professors**:** are automatically considered as members of the library and their information should be extracted from the employee records. The library card is mailed to the approximate professor's campus address. Professors get card renewal notices sent to professor's campus address. Professors can check out a book for three-month intervals. Professor have two-week of grace periods before a notice is sent

2.3. Development method

At a high-level overview the software development process can be separated into two main methodologies:

- Plan Driven: All the steps are carefully planned, features are fixed early on in development and the overall progress is compared to these predefined criteria
- Agile: All the steps are done in small but frequent increments with minimal initial planning and are modified according to the constant feedback and changes in requirements

2.3.1. Bohem and turner chart

Choosing the right development method is key to a successful product. One way of choosing the right development method is by assessing the project, the team, the project scale and the customer of the final product.

The following diagram (figure 2), is a Bohem and Turner chart, that helped us in picking a development methodology.

According to it, we can conclude the fact that our team needed an agile method of sorts, given that most criteria point towards the center of the star with slight tendencies towards the exterior.

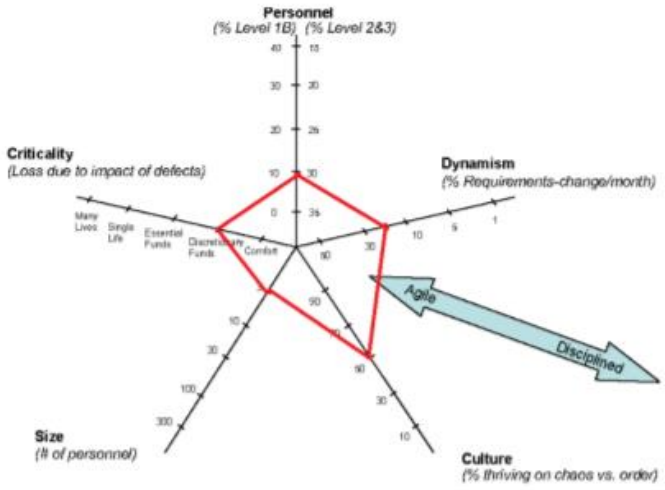


Figure 2 (Bohem and Turner diagram depicting our team)

2.3.2. Considered methods

Following the Bohem and Turner chart, we have decided to take an in depth look at three of the most popular Agile development methods.

2.3.2.1. Extreme Programming (XP)

Extreme Programming (XP) is an agile development methodology which shifts its focus on the overall quality of the software. XP is based on four main values:

1. The first and arguably the most important value is communication. Communication is crucial in any methodology, but especially in Agile as the requirements can change rapidly.
2. Simplicity, the simpler the solution the less chance something fails. Development is to be started with a simple solution and improved with continuous refactoring (also one of the key XP practices).
3. Feedback is the 3rd value which is a must for high product quality.
4. The final value is courage. It encourages making decisions without the need of a formal process.

This is the complete opposite with Plan Driven methods like UP where every significant change is accompanied by an abundance of documentation. XP lists 12 key principles for creating a high-quality software. Among the 12 principles is test driven development or TDD for short. TDD requires writing the tests before developing the product. This ensures the number of errors/bugs is minimized in the later stages of production. XP takes the Agile incremental philosophy into programming with small sized releases and continuous integration. The small releases allow the end users to test the system in early stages of the development process, compared to Plan Driven products, even though they only include part of the intended functionality.

2.3.2.2. SCRUM

Scrum is one of the Agile development methods most suitable for small development teams working in sprints, ranging from a single week up to a month. At the end of every sprint, value should be presented to the customer by being one step closer to completion. Unlike XP, the SCRUM methodology focuses only on the development process. It defines three roles:

- Product owner: who represents the product stakeholders (ex. the company receiving the final product).
- The scrum master (usually a single person, but it can change over sprints): who is responsible for the scrum process and removing any distractions that the team might encounter.
- The team: which represents all the development team who design, create, implement and test the system (the order is not necessarily fixed).

At the start of each sprint, a sprint planning meeting is held. In this meeting the product owner selects the tasks that should go into the sprint backlog based on the current state of the application and the feedback given by the final users/ customers. Each sprint is closed with a sprint review where the product owner shows off the accomplishments to the stakeholders. An internal meeting is also held to discuss and improve upon the development process itself. Scrum also includes daily standup meetings where the members discuss what they have done since the last meeting, what they will be working on and what problems did they run into. These problems are to be resolved by the scrum master.

Besides the usual sprints there is also a special initialization sprint called “sprint zero”. Sprint zero is used to set up the work space for the project to begin. Investigate any problems which can potentially slow down the process (such as unknown technology), set up the development environment, etc. Unlike UPs inception and elaboration phases, the requirements set in sprint zero are not final and are expected to change over the course of the development.

2.3.2.3. Kanban

As with the previous two agile development methodologies, Kanban aims to work with the chaotic nature of variable requirements and the need to shift focus between priorities quickly. Kanban also focuses on making the development as smooth as possible by making transparency, respect and agreement as its values (there are more values which are equally as important, but these are the ones we aimed to retain when considering the method).

Leadership is an integral part and is encouraged at every level (again circling back to its values) and the changes are smaller, more evolutionary rather than revolutionary. Frequent deliveries in small sizes is a must, and the quality must be up to scratch with the rest of the team's work. The scope and progress of the development work is visualized using a Kanban board, where teams agree to which tasks to choose. If a task happens to last longer than the intended time, it will be put back to the backlog and can be chosen again. Making policies explicit is a must. Policies, such as the definition of" ready", or the definition of" done", should be used rarely for maximum impact. They're used to ensure a common ground across both the team and the stakeholders.

Feedback loops, to ensure the upmost relevant tasks, are implemented; and are a good way to minimize time wasted on less important tasks. Those feedback loops include: Strategy Review (Quarterly), Risk Review (Monthly), Replenishment Meeting (Weekly), Kanban Meeting (Daily) and Delivery Planning Meeting (variable)

2.3.3. Chosen method

Since all the considered methodologies contained traits that we considered both useful and impractical, we decided to create a unique combination of all the three major Agile development methods. One that would contain the following guidelines:

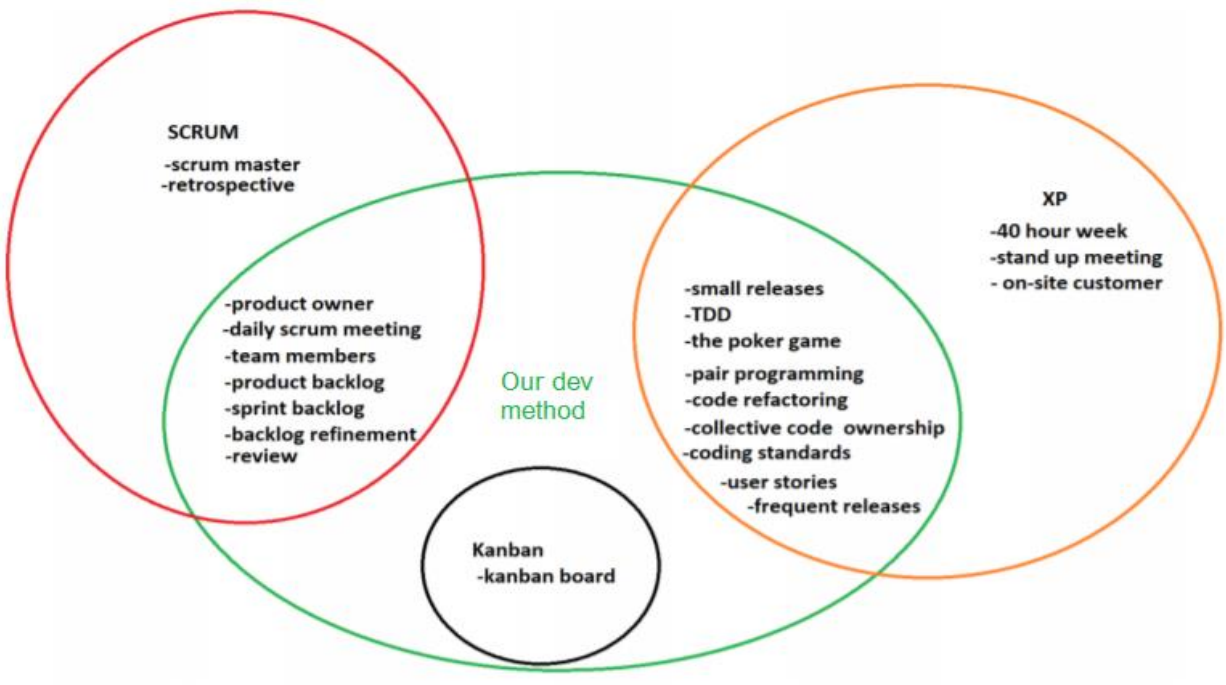


Figure 3 (Ares development method)

Our aim was to first build a Minimal Viable Product (MVP). After the MVP has been finished, with small increments we added functionality without braking the core application.

We also put a heavy emphasis on communication, as we noticed early on the importance of keeping a shared and up to date understanding of the developments state. Alongside communication, Pair programming, and TDD were used extensively. As expected TDD slowed us down in the early stages of development, but it proved to be a major contributor to the quality of the overall application and saved us from the headaches of debugging in the later stages where the application was becoming more complex.

2.3.4. Workflow analysis

After selecting our system development method of choice, we immediately started working on our task board. The task board contains all the features we determined had to be implement into the product. While we aimed to implement most of the features in the backlog we knew (due to time constraints) we would have to leave some features for a later release date or leave it to the next team who will support the product.
The backlog was created by using information provided by Georgia Tech Library and it was prioritized based on the impact each task would have on the customer. The product owner was elected (Adam), and he ensured, that the group was on the right development track and would finish work in time. The resulting product was reviewed by the product owner at the end of each sprint, to receive feedback and use that feedback to further improve the product.

2.3.4.1. Steps of Development

The consensus was that we would separate development into 7 steps (this decision was based, on our previous experience working with Agile methodologies and internship experiences).
When following a task, it must go through the following steps:

1. Report backlog **-** Place where all not started report writing tasks are kept;
2. Product backlog - Place where all not started product development tasks are kept;
3. To Do - Tasks for current sprints, with an already assigned working member;
4. In progress - Location, where the tasks which are currently being worked on are kept;
5. In review - Area where tasks are placed when they are considered to be finished, but still awaiting approval from other members;
6. To deploy - List of approved tasks, that are not yet deployed to the customer;
7. Done - Region which contains all tasks, which were deployed to the customer;

2.3.4.2. Tasks

We referred to user stories as tasks most of the time as we never utilized them outside of creating (in sprint zero), implementation and visualization on the task board (figure 4).
To improve the usability of the task board we decided to utilize the following techniques:
MoSCoW**:** Short for must, should, could, will not do. It is a task evaluation method used for categorizing the importance of a given task. We used MoSCoW in the form of colored badges representing the priority:

1. Green - must
2. Yellow - should
3. Orange - could
4. Red – will not

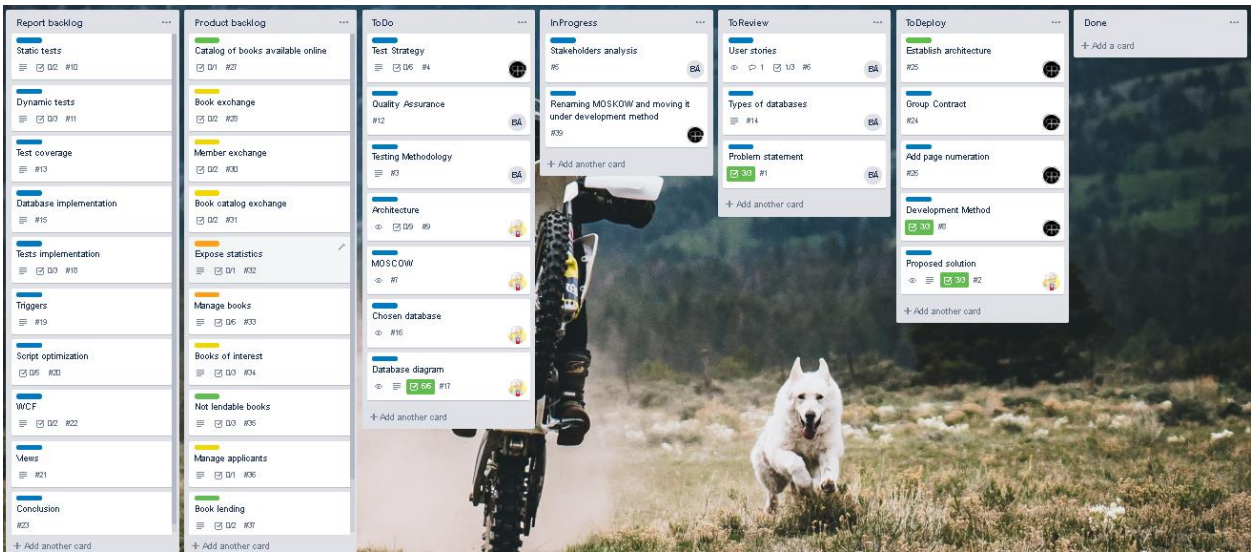


Figure 4 (Kanban board)

Task separation: To make the backlog easier to navigate we decided to not use MoSCoW for document-oriented tasks, instead using the blue color to identify them, as it was decided that all of the planned writing tasks were to be implemented

Descriptions**:** Each task had a separate section where a more detailed description complimented the title to remind other members on the team what the task was about and what were the intended results

Comments**:** Knowing that communication is key when it comes to working in groups, it was important to quickly and reliably provide feedback on the state of the task, this was done by the utilization of comments, which allowed us to comment on the quality of the final implementation and possibly even reject it entirely, but still provide adequate feedback without directly contacting the member

Check list: When a complex task is presented which might include the need of having multiple parameters or sections which must be met, a checklist is used to visually inform other users of what has or has not been implemented yet. A percentage is automatically shown on the task to indicate the finished progress

2.4. System design

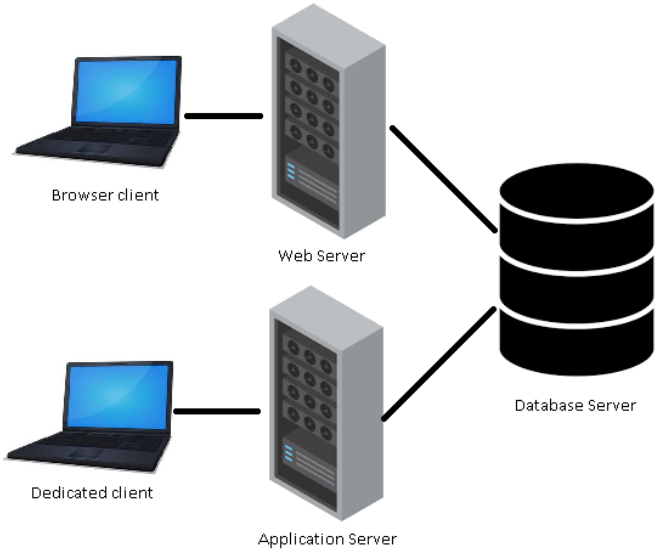
Picking the best architecture which would fit our needs, was a long process, since the requirements of the library written in the problem statement were not quite clear in the begging as they could be interpreted in various ways. Furthermore, information such as how the application would be used and what different types of librarians should be able to do was not provided.

When deciding on the type of system, we would be making, we first had to review the requirements given to us by the library. After reading the request and thinking of possible solutions we came to two possible types of systems we could make, to achieve an adequate solution to the requirement:

2.4.1. Client-Server

A single application which would be given to the client and allow them to manage the database (depicted in figure 5).
Client-server Architecture can be classified into two models based on the functionality–

- Thin-client model- all the application processing and data management is carried by the server. The client is simply responsible for running the presentation software, it places a heavy processing load on both the server and the network.
- Thicc/Fat-client model-the server is only in charge for data management. The software on the client implements the application logic and the interactions with the system user, new versions of the application must be installed on all clients.



Pros

- Easy/fast to make
- Great performance could be achievable

Cons

- If there is a new update, client could be required to get the newest version (without updating the client would not be able to use the newest features or the user's applications might not work at all)
- Client system dependent
- Hard to maintain because of code duplication (if multiple different types of clients are used)
- A lot of work to add a new client

Figure 5 (Client-Server representation)

2.4.2. Multi-Tier (N-Tier)

A simple solution which has its business logic in a separate project which could be shared by many clients, limiting code duplication (depicted in figure 6).

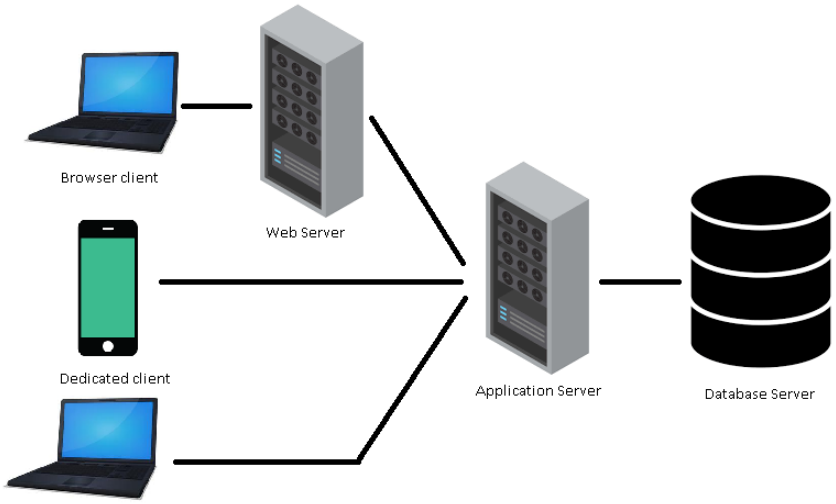


Figure 6 (N-Tier representation)

Pros

- If there is an update to the product all the new functionality is automatically accessible to the user
- Relatively easy maintenance
- Easy to expand with more clients
- Not dependent on the user's hardware capabilities

Cons

- Harder to develop than the dedicated client
- Worse performance due to possible extra steps

2.4.3. Selected architecture

After reviewing the pros and the cons, we determined that the N-Tier approach is better for our needs, as it would be easy to work with (for the user), has possibility to be simply expanded to new types of clients (such as mobile) and functionality updates automatically affect all of the service's consumers.

Furthermore, it is required to have a service available to exchange information between the libraries and this would easily accommodate it. Meanwhile, the cons of having a distributed application are rather minor, for example sacrificing a bit more time in development of the application for easier expansion in the future and minor performance decreases, because now there would be a proxy between the client and the database.

2.4.4. Server side

Deciding the best technology, to use for the server is a hard and time-consuming process, that is why we decided to do more research on this subject and came to a few possible solutions.
Here are some definitions worth knowing, before diving deeper into the rest of this subject:

- Application program interface (API): is a set of routines, protocols, and tools for building software applications. A good API makes it easier to develop a graphical user interface (GUI) by providing all the necessary building blocks.
- Web service: is a generic term for an interoperable machine-to-machine software function that is hosted at a network addressable location (All Web Services are APIs but not all APIs are Web Services).
- Microservices: is an architectural style that structures an application as a collection of services that are highly maintainable and testable, loosely coupled, independently deployable and organized around business capabilities.

Accessing the web services can be done through two different ways**:**

- REST (Representational State Transfer) - Is a relative newcomer and as an architecture style is lightweight and naturally more flexible than SOAP, requesting the user to just use URLs and can get a response in multiple different formats, while SOAP is restricted to only using XML.
- SOAP (Simple Object Access Protocol) - Old but still relevant solutions to the data exchange problem, SOAP uses a more rigid set of messaging patterns than REST. SOAP relies exclusively on XML to provide messaging services, the XML can become extremely complex and in some programming languages, programmers need to build those requests manually, which becomes problematic because SOAP is intolerant of errors, but when working with .NET languages, developers don't have to even see the XML message.

When we decided on a framework to follow for creating our API we concluded, that there are only two frameworks worth considering for the job. Both considered technologies are APIs in the .NET Framework meaning, that they are easy to use in Visual Studio and with other .NET technologies. Besides that, as it just so happens all of us have had previous work experience in both:

2.4.4.1. ASP.NET Web API

Is a framework that makes it easy to build HTTP services that reach a broad range of clients using REST. (Resource oriented)

Pros

- Easy to make
- Speed (Data is kept to the minimum)
- Many content formats (XML, JSON, CSV and more)
- Supports a wide variety of clients (browsers, dedicated and many more)
- Uses the full feature of HTTP (like URIs, request/response headers, caching, versioning, various content formats)
- Open-source
- Allows for caching, compression, versioning

Cons

- Only 1 transport protocol (HTTP)
- No support for higher level protocols such as Reliable Messaging or Transactions

2.4.4.2. WCF (Windows Communication Foundation)

Is a set of APIs in the .NET Framework for building connected service-oriented applications, mostly implements SOAP.

Pros

- Many transport protocols can be used (TCP, UDP, Named pipes and many more).
- Multiple encoding types (Text, MTOM, and Binary).
- Support for higher level protocols (Reliable Messaging or Transactions).
- Improved security.
- Also allows for One way and duplex communication, not limited to request reply.

Cons

- A lot of time is spent in configuring WCF services.

2.4.4.3. Chosen technology

After reviewing the two choices, we came to the conclusion, that we wanted to have a service which would be easily maintainable and expandable, that is why we came to conclusion that we should use WCF instead of Web API.

Although, Web API could be a better solution for exchanging information with other libraries, as it offers more content formats and higher speeds; we determined that both projects would be mostly used by computers connected through LAN, where the TCP protocol would be a better choice, rather than HTTP.

Furthermore, WCF offers better security, which although may not be the main concern is still something to consider. We determined, that due to time constraints WCF should be our main choice, as working in WCF is a mostly automated process and does not require too much code, since a lot of the work is already done behind the scenes.

2.4.5. Client side

For the client we reviewed multiple options; in the beginning we thought of creating a dedicated client, which could be for desktop or mobile, but after considering the request of the library we realized that some of their requirements for the client were:

1. Application should work on multiple different platforms (pc, mobile…)
2. Easily accessible, not requiring any installation, as that would make students less likely to use it
3. Should be easily changed and updated for future expansions

This forced us to make a web-based client, which would be interactive, understandable and would allow us to expose the solution to almost every conceivable type of client. Since we were relatively new to web development and user interface was not the main concern of the solution, we determined to review this subject and decide on what technology fits us best, so it would be easy/fast to develop and to later use. We limited our choice to ASP.NET solutions, since we had decided to work in C#.
Some of the considered options are:

1. MVC
2. Web forms
3. Web pages

After reviewing these choices, we determined that web forms are not for us, even though they are easy to create, they are being depreciated and we would not want to make a solution which would be considered outdated from the beginning.

After comparing the options which were left with, we determined that MVC is the best choice, as it is frequently being used in actual businesses, it is Ideal for developing complex, but lightweight applications and it was the technology which we had the most knowledge in, so less time would be spent learning a new way of implementing functions, instead focusing on developing a solution.
To improve the created client, we decided to use various technologies such as:

1. Ajax
2. JQuery
3. Bootstrap
4. ASPX
5. Sessions

And languages associated to them:

1. JavaScript
2. CSS
3. Html
4. Razor

2.5. System architecture

When working on the architecture plan, we decided to revisit a previously used software solution named Phaethon, a solution which was made for a Latvian based company.

Phaethon's design, just like Ares's, was an N-Tier architecture with a simple web client and separate project for tests and models. Phaethon was somewhat similar to this project, as it tried to achieve similar goals and it was easy to work with, while also allowing for easily testable code. The solution provided several benefits, such as easy addition of new clients and low-cost maintainability. Not only that, but also, this architecture helped us achieve the goal we had set for ourselves, for this project, that being to pursue "high cohesion and low coupling".

Even though the current architecture was inspired by one used previously (figure 7), there were some improvements made. The changes furthered the code testability, simplified the architecture, while also improving the solution based on previous experiences. Most notable changes to the architecture, as shown in figure 8, is that interfaces are more widely used in the project, thus allowing for easier exchange of base classes. Due to need for resulting product to satisfy the requirements of the test and database courses, the implementations were decided to be fulfilled by having the solution carried out twice, once written in a C# project and once in a database using SQL.

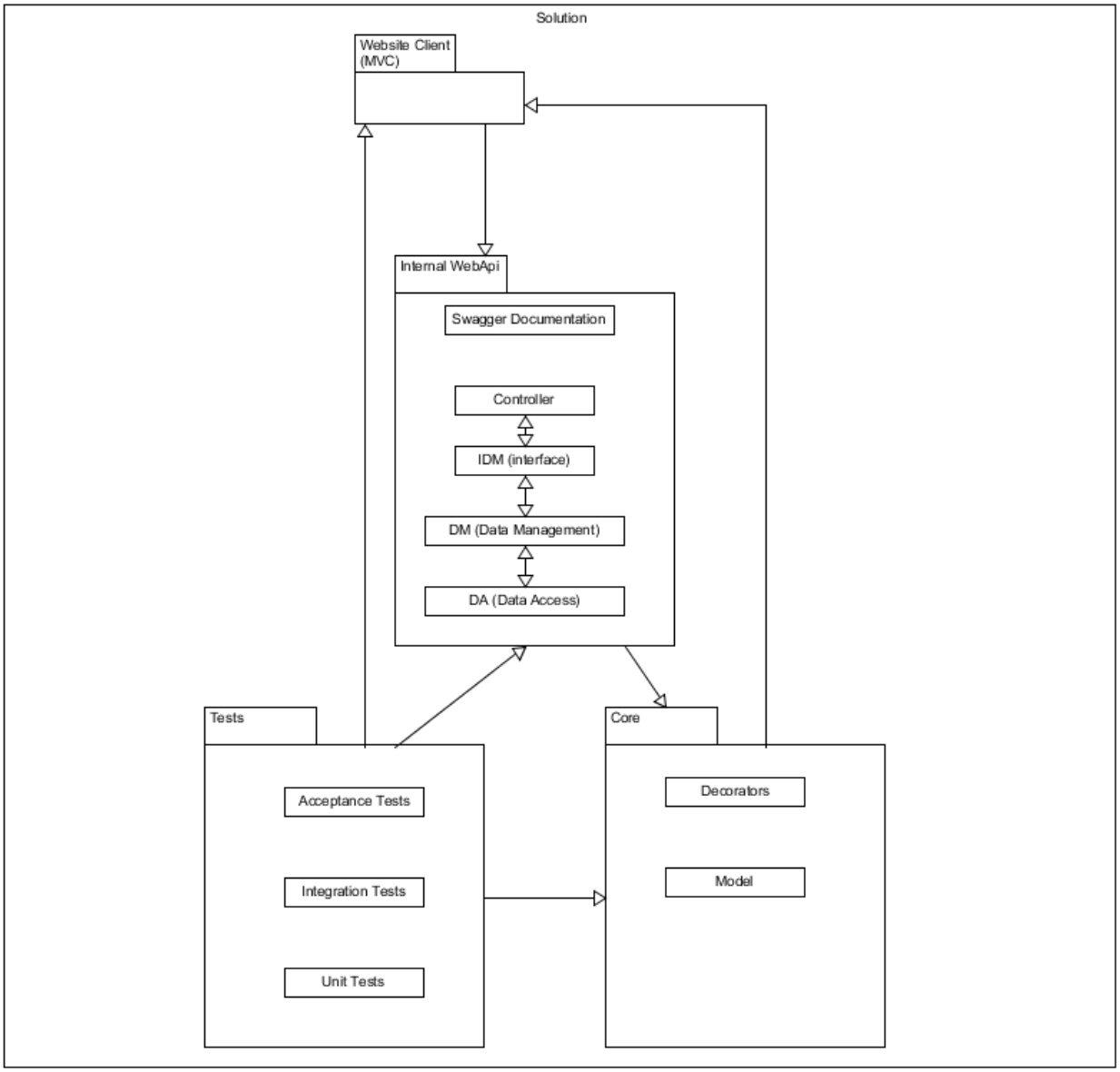


Figure 7 (Phaethon architecture)

As seen in figure 8, we have four projects, with each of them being used for generally different purposes.

- GtlService- the brains of the product, providing services that could be consumed by users and performing corresponding actions in the database
- GtlWebsite- holds the user interface, allowing consumers of the service to easily interact with it
- Tests- project dedicated to improving and maintaining the quality of the product
- Core- contains models used in all projects

GtlService is a service which was further subdivided into layers, where each of them are responsible for different tasks:

- Controller- used for communication between projects

- DataManagement- used for making calculations and changes to objects
- DataAccess- used for connecting and working with the database

Most layers have interfaces and dependency injections, as it helped with testing and maintenance of the software. The project called "Tests" contained folders for each type of dynamic tests used in the project, so they would be easy to keep track of.

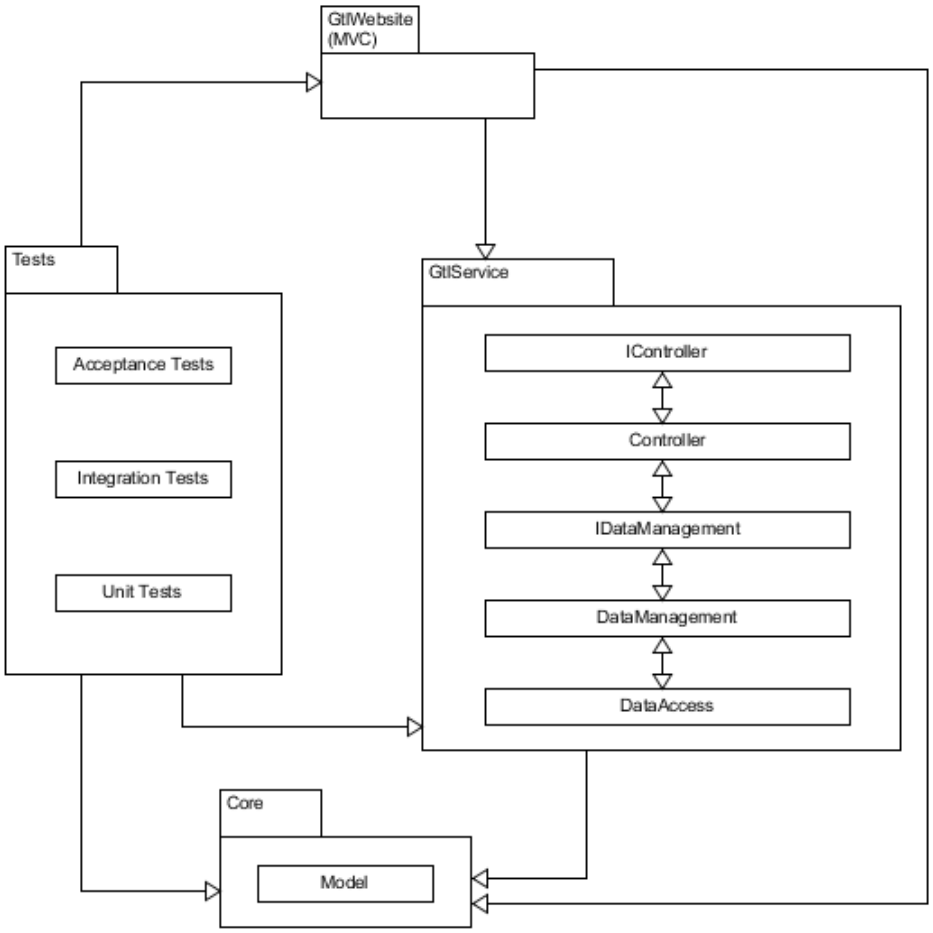


Figure 8 (Ares architecture)

2.6. Quality Assurance

Before we go into detail about the importance of quality, lets first talk about what quality actually means. Quality in software refers to:

- The structural quality of the software – which embodies the robustness of the codebase. Maintainability, supportability, the number of bugs or likeliness to crash are all taken into consideration. In short, quality code has reduced cyclomatic complexity and is testable, so when it is delivered in the form of an application it can properly be expanded on and maintained years after its initial release
- The perceived quality of the software – in this case we focus on the satisfaction of the initial requirements given by the product owner and how highly the target group rates the final

product. Of course, perceived quality can only be achieved if the code of the application is up to scratch as it directly influences the perception of the software

2.6.1. FURPS

A great way to tie up structural quality and perceived quality is by utilizing the acronym FURPS, which is a way to classify and validate the software against the gathered criteria. FURPS is short for:

- Functionality – refers to the features the target audience wants to see in the release. Functionality represents the main functional requirements in the system, in our case (regardless of priority/risk severity) it would be: borrowing a book, returning a book, producing statistics, etc.

The rest of the URPS attributes represent the other non-functional requirements which are architecturally important.

- Usability – how effectively can the target group use the application, focusing on the learning curve of the user interface and how good is the UX (user experience).
  As our project does not really focus on the UI of the application, we will not go in detail about the UX and usability
- Reliability – how robust must the application be when interacted with, we must also take into consideration the up time, the down time and the quality of the data produced/received. Considering the environment and the use case of the application reliability is a high priority, it's one of the main reasons we chose the V-Model (the development process we will go into detail later on in the report), as we felt it would yield a higher quality product than its Agile counterpart, the four quadrants
- Performance – how resource intensive is the application when used by the customer and their users. As with the usability, we cannot accurately give a number or estimate as to how resource intensive the final application will be as we do not have a fully developed UI and have not finished developing all of the user stories
- Supportability – how easily the application can be maintained after its initial release by (possibly) future development teams. The application was built with supportability in mind, hence the reason for the usage of code standards, numerous tests, which test most aspects of the software

While FURPS is used to ensure the overall application measures up to the global criteria of the project, we must also focus on the individual code snippets of the application. Sadly, it is not feasible for us to test every part of the application, but we can define a test coverage by selecting the most important parts of it (done in risk assessment) and focusing on making the vital parts of the application as pervious to errors as possible. For the rest of the application not tested as rigorously we maintain quality by using coding standards and constantly review each other's work, before approving and adding it to the application.

2.6.2. Cyclomatic complexity

One of the most important metrics for code quality, refers to the complexity of the code, how many decisions must it go through for it to execute a specific function, the higher the number, the worse it is. As the higher number of decisions an application must make the greater the risk of it to produce an error. If we convert a method's code to a flow graph, the number of linearly independent paths will be the cyclomatic complexity.

The value to complexity ratio is the following:

- 1 – 10 is a low-risk program
- 11 – 20 is a moderate-risk program
- 21 – 50 is a high-risk program
- anything greater than 50 is a highly unstable program and should be refactored for simplification

As shown (figure 9) our application scores favorably when looking at the maintainability index and the cyclomatic complexity. When looking at the cyclomatic complexity score, they mostly range from 1 to 9 which are considered to be low risks, with one method being an exception with a moderate risk level of 13, indicating the application's code is of high quality.

Hierarchy	Maintainability I...	Cyclomatic ...
GetAvailableCopyId(string) : int	56	5
GetOutOnLoan(string) : int	59	5
LoaningDm_Code	61	16
LoaningDm_Code(LoaningDa_Code, MemberDa_C	76	1
ReturnBook(int) : bool	62	3
LoanBook(int, int) : bool	60	6
NoticeFilling() : bool	57	6
MaterialDm_Code	56	41
MaterialDm_Code(MaterialDa_Code, LibraryDa_Co	68	1
DeleteCopy(int, int) : bool	61	5
DeleteMaterial(int, string) : bool	61	5
CreateMaterial(int, string, string, string, strir	55	8
CountAvailableCopies(List<readAllMaterial>, List<	58	9
ReadMaterials(string, string, int, string, string) : Lis	46	13
Core (Debug)	93	244

Figure 9 (Cyclomatic complexity)

3. Test

This part of the report focuses on the importance of testing a software solution, how it can be done, how we decided to do it, and why it was done like that.

As previously mentioned, we have decided to implement the "business logic" of this program twice, once in a code approach and once in a database approach. This section will focus on testing the code approach, as it allows for the use of dependency injection, mocking, spying, etc. techniques that we did not manage to find in sql server unit tests. On top of that, testing the code approach also allowed us to exercise our "testable architecture" planning skills. As for Integration tests, they are done for both approaches.

3.1. Testing Methodology

3.1.1. V-Model

The V-Model also known as the verification and validation model is a software development process which is considered to be an extension of the traditional waterfall method with extensive focus on testing.

Instead of moving in a linear fashion, the V-Model first defines everything from the concepts, down to the low-level design, but at the same time it specifies all the criteria it must be verified against.

After the implementation, it verifies the application based on the previously written validation points. If the application passes the verification step it is ready to be shipped to the customer.

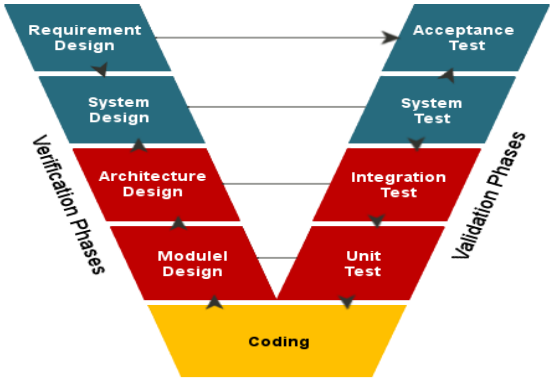


Figure 10 (V-model)

Pros

- The final product will always be at a very high-quality standard as the product is only shipped once the application has passed all the tests
- It shows a clear roadmap which the developers can follow (step by step process)
- Since testing is so tightly incorporated, less time will be wasted in later stages of development, thus making the application less time-consuming in the later stages of development

Cons

- As it is an extension of the waterfall method it is inherently less flexible when compared to agile methodologies such as The Quadrants model
- May not be correct for a more chaotic development team/environment
- It may trick developers into a false sense of security because of the high focus on tests
- Documentation is necessary at all times, which is very time consuming
- Since no prototypes are created there is a very high risk involved with meeting customer expectations

3.1.2. The Quadrants model

The quadrants merely act as a map to guide developers when they plan their tests and make sure all resources are available to accomplish them. The Quadrants consist of four sections (Q1,Q2,Q3 and Q4, note the numbering system does not imply any particular order as it's an arbitrary numbering style to shorten the naming of each quadrant)

To decide if a test is business facing or technology facing:

A test is business facing if it answers business domain related questions. These questions are best understood by business experts and help answer how the application will function in a specific scenario.

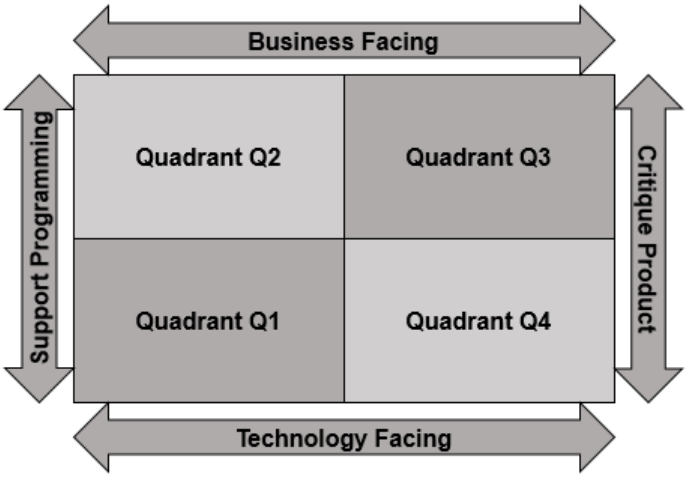


Figure 11 (Quadrants model)

A test is technology facing if it answers a technology framed question. These questions are best understood by programmers/system architects who understand what needs to be implemented based on these questions.

Pros

- As the quadrants model is an agile testing methodology it is very flexible in the environment it exists in
- Customers can have firsthand experience with the product and can give immediate feedback
- If set up correctly, it can provide a comparably high-quality product to the V-Model

Cons

- The lack of documentation can be problematic in certain projects
- There are no specific rules about what goes into which quadrant
- Initial discussions need to be held for each quadrant, so some time is spent at the start of each quadrant discussing testing

3.1.3. Our choice

After carefully reviewing the methodologies, we decided to choose the V-Model rather than the agile quadrants model, as it better suits our current needs of focusing on testing in the GTL project. Although the Agile Quadrants does better suit our chaotic style of work, the project requirements put a heavy emphasis on testing, which is more appropriate to the waterfall methodology, so we thought the V-Model would be a natural fit. Also, as the quadrants require time to negotiate as to what goes into which quadrant (and the fact that we are working on a tight schedule) it further pushed us towards the alternative.

Although we can't fully adopt the V-Model, we can integrate it into smaller sections of the project, which are mainly related to programming tasks. This is done so every time we finish a task we know its instantly ready to be integrated into the final software solution as it already meets to the quality standards (due to the validation nature of the approach).
In the actual implementation of the methodology we did modify the V-Model to fit our needs, for

example: we omitted the sequentially as it limited our flexibility which we were not prepared to give up as it is a major plus point of agile.

3.2. Strategy

A test strategy is an outline that describes the testing approach of the software solution. It is a crucial piece of document which helps the entire development team better understand key issues of the testing process. On top of that, it helps the managers organize and handle risky areas of the project, as well as keep track of possible issues and development progress.

3.2.1. Project Scope

The purpose of this project is helping the Georgia Tech Library in their most usual tasks, of lending books to several thousands of users, by creating a software solution that can achieve just that. Thus, the business goals of this project are:

- Improved response times, when it comes to members loaning and returning books
- Better material tracking
- Easy, fast and reliable inter-institutional communication

However, due to the lack of time, we have decided to test our UI using only the market leader browser, Google Chrome.

3.2.2. Test Approach

On top of the aforementioned testing methodology, we have decided to conduct our tests using the following code standards:

- Tests will be written using the AAA (Arrange, Act, Assert) pattern, as it helps with readability by clearly defining sections within a testing method
- Acceptance Test method names are created as follows: UserStory_TestCase_ExpectedBehaviour
- Test classes are created and organized within dedicated folders, for each type of test (Unit, Integration, Acceptance)
- Tests are created and are able to run independently
- Tests are running in fresh environments, unless otherwise specified
- Integration tests are capable of testing both Code and Database approaches using the factory pattern

Alongside those code standards, testing documents, diagrams and other non-code related material, will be tested using informal reviews and/or walkthroughs, with each member taking one of the roles typically encountered in such meetings (moderator, author, reviewers), intentionally omitting the scribe, as the value such a role brings is minimal, given the team size.

3.2.3. Test Environment

As there are several tests that we have decided to work with, the test environment varies widely from the safe development environment of the IDE, to brand new, fresh and isolated nodes in the continuous development solution, to actual running Azure servers on which the application can be Alpha tested.

The following table (figure 12) describes how each of the tests are spread across which types of environment:

Test type	Environment	Environment description
Unit	Local / Appveyor	Personal laptops and virtual nodes using Visual Studio 2017
Integration	Local / Appveyor	
Acceptance	Local / Azure Cloud	Microsoft Azure free servers, on which the application is deployed after each release.
Alpha	Azure Cloud	

Figure 12

3.2.4. Testing tools

In order to make testing easier and faster, we have decided to automate it using the following tools:

- N-unit tests
- Selenium with chrome web driver
- Appveyor

These tools have been chosen for their unique abilities and ease of use, in order to amplify our project's testability and to ensure it works with as little impediments as possible.

In terms of test automation, we have compared several tools that might help us, tools like: Selenium, Katalon, Squish, Tricentis, or TestCraft. And we decided to go with a Page Object Model-based framework, rather than a "record-n-play" framework, as although the record-and-play function of Katalon, might be beginner friendly and not require any programming skills, we consider the flexibility offered by a Page Object Model-based framework, to be vastly superior and freeing at the same time as code snippets might be reused in future tests or easily changed and refactored as the product evolves, where recorded tests must be constructed from scratch, with every single change. In the end, we have chosen Selenium as our test automation tool due to the following reasons:

- Compatibility across Operating Systems
- Support for multiple scripting languages (including C#, which we are most familiar with and are using for this project)
- Support for running tests on multiple browsers
- Large and open community (in case we needed assistance)
- Extended documentation (in case we needed assistance)
- Open Source (free to use and a large community behind it)

Often, checking that all tests are still successfully executed, and the implemented changes did not affect any of the previously implemented functionality, are forgotten. Thus, we decided to use Appveyor as our continuous integration solution for Ares. With Appveyor, the "burden" of rechecking tests and functions, is taken from the tester's shoulders and put on remote, fresh and isolated nodes. We have integrated Appveyor with our GitHub repository and configured it such that it will automatically run with every new push to an open pull request, allowing for the pull request to be merged with the main branch only, if the build was successful and if the tests run successfully. This integration has helped us

ensure that new code changes don't affect previous functions, as well as making sure that the code will run on other machines.

3.2.5. Release Control

Following the decision of working under an agile framework, we have decided to release new updates at the end of every sprint. Updates that would bring both product stability and value to the customer.

We have also decided to use a cloud hosting service (Microsoft Azure) as grounds for our release environment, where the customer can get a hands-on experience of the new changes.

During a release, the following procedure would typically be followed:

- Merge the Development branch into the Master branch
- Continuous integration checks
- Deployment of the Master branch to Azure
- Product owner reviews changes
- Feedback session

3.2.6. Risk Analysis

In order to ensure that our application does not encounter any major defects and the solution runs as smoothly as possible, we have conducted numerous peer reviews to determine which scenarios are most likely to occur. We used a risk matrix to categorize and visualize the risk/priority, a given scenario happens to be in. A risk level is chosen based on the likelihood of it occurring and the impact it has on the software. Each risk level is color coded for less ambiguity

Likelihood ->	Impact ->		
	Low	Medium	High
	Low	Medium	Medium
	Low	Low	Low

Figure 13 (Risk level color meaning)

Scenarios are ranked as follows:

- The scenarios with the highest risk rating are given full priority as they are vital to the workings of the application and their test coverage must be as close to fully covered as possible (over 80%)
- Medium risk rating is considered to be important but not a vital scenario, so the test coverage, while must be adequate, does not require to be fully covered
- Scenarios with low risk ratings are to be covered only to the extent that ensures the user does not run into problems as long as they do not deviate from the preplanned usage path

Below (figure 14) are some examples of various scenarios with their corresponding risk levels

Scenario	Risk level
Checking out a book when not allowed	High
Returning a book which was not loaned	High
Adding new books to the library	Medium
Removing old books from the library	Medium
Renewal of membership card even if the person is not a member anymore	Low
Removal of member who is not a member anymore	Low

Figure 14 (Scenarios and their perceived risk level)

3.2.7. Review and Approval

In order to ensure that new features are added correctly, we have decided to instate several guidelines upon how the project should be expanded:

- To keep a level of traceability, always work in new branches, based on the Development branch, with the same name as the task you're currently working on (including the Trello card number) ex: #cardNumber-card-title
- When you believe a task is finished, create a pull request (always write a short description of what you did when creating the pull request)
- Once the branch is approved (passes Appveyor checks and gets at least one positive review), the owner of said branch, will merge with Development branch
- Merging with the Master branch will be done once every sprint

As for approval, we have decided to employ the use of Acceptance tests, where, at the end of each sprint, the customer would get to see and experience the new changes, followed by their feedback and approval/dismissal.

3.3. Static testing

Static tests involve the examination of the software without the use of software code and documentation as it mainly consists of reviews, setting standards and meetings. In this section developers agree on a set of rules which will be used throughout the development phase (such as documentation and coding standards). Static testing consists of checks which are used to avoid errors in the early stages of development and can usually prevent problems which would have occurred in the later stages of development, as it focuses on avoiding potential defects.

3.3.1. What we used

Static testing techniques we used include:

- Informal reviews – in the early stages of development informal reviews are used to inform other members of the development team without the need of documentation. We regularly used informal reviews through Slack as it provided us with a reference point as to how well the team is handling the given tasks
- Walkthroughs – they are a form of peer review where developers walk each other through the product and ask questions, make comments about its state. It is used to familiarize others with the software. We mainly used walkthroughs to familiarize each other with the newly written code, and what the thought process was behind it
- Technical reviews – another form of peer review, where developers ensure the technical concepts are well understood and implemented correctly
- Coding standards – coding standards are great for maintaining cohesion within the application and helps with maintenance in the short (done by the team) and long term (handled by future development teams). In general, we agreed to keep code as minimal and simple as possible to avoid a large cyclomatic complexity and variables, methods, classes including any other code-related elements, should be named properly and decently, following the CamelCase format

These reviews were mainly used in the beginning and in tandem with teachers to verify if we have correctly understood the requirements, they were really useful as we have received instant feedback.

Static analysis is used before and during integration testing to confirm that the predefined guidelines and coding standards are adhered to.

3.3.2. Data flow

Data Flow is a form of white box testing and its purpose is to reveal mistakes which may result in incorrect implementations and usage of the data variables/values (data anomalies), as the inconsistency can lead to failure.

In figure 15 we show the data flow of logging in to our application.

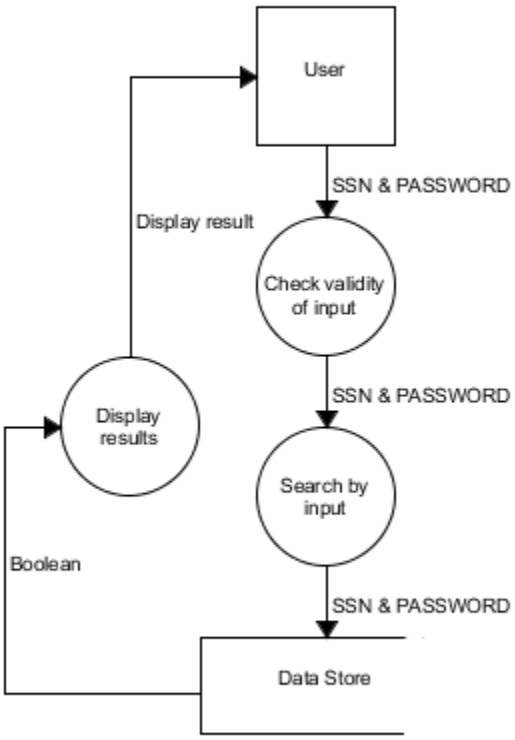


Figure 15 (Login Data flow diagram)

3.3.3. Control flow

Control Flow is the sequence in which individual functions, statements or instructions are (or can be) executed in the application.

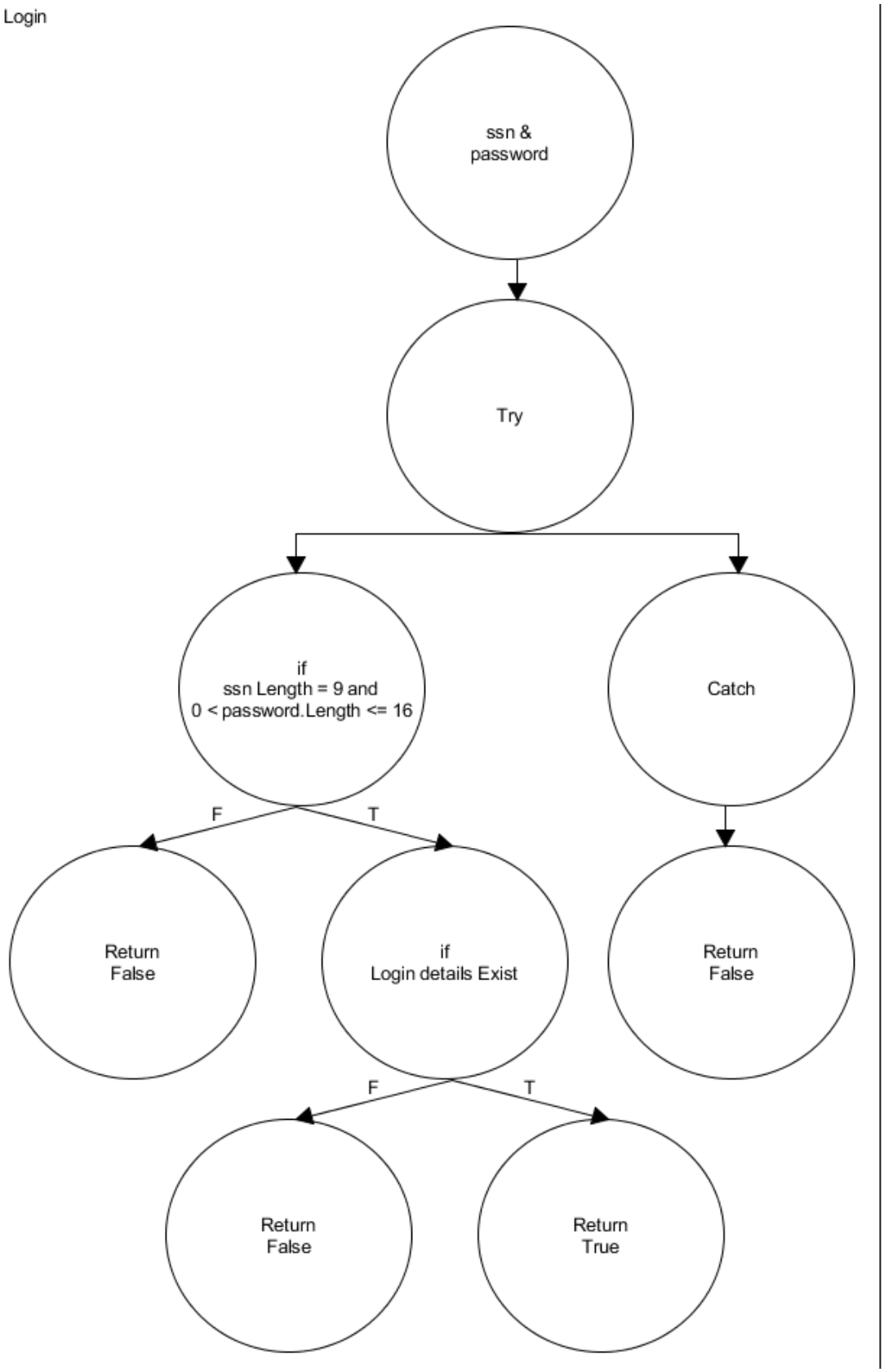


Figure 16 (Login Control flow)

3.4. Dynamic testing

Dynamic tests are great for verifying correctness and overall improve the quality of the software. In tandem with static testing they are one of the main pillars of quality in the development phase of the application.
In dynamic test we test the behavior of the software's code, the software must be completed or be close to completion for it to be tested. By giving input values we check if we get the expected result. This is done by executing different test cases (done automatically via an automated process or done manually).

Test: specification, cases and scenarios

Test cases are documents in the form of tables consisting of a set of variables and conditions under which the correctness of the software application is to be pre-determined in order to verify its functionality. Test scenarios are test procedures which are a set of test cases built on the basis of business requirements. The way we identified those is by putting ourselves in the shoes of the final users and go through the software to execute the test cases in different sections of the application (being the scenarios).

On top of that, in order to decide the input values for each of the test cases, we did a boundary value analysis. Therefore, based on the boundaries that each variable has, we decided to test values before, at (where possible), and right after the boundary.

We have included two different scenarios consisting of various test cases: Logging into the application (figures 17 and 18) and checking out a book (figures 47 and 48, in the Appendix due to their size)

Identifier	16
Description	Member wants to log into the application
Precondition	Must have a valid SSN, a password consisting of both numbers and letters with the minimum length of 6 characters and the maximum of 16
Estimation	No estimate given

Figure 17 (Login scenario)

Scenario: Log In – 16	Valid test cases		
	SSN	Password	Expected result
1.		123abc	invalid
2.	1	123abc	invalid
3.	12345678	123abc	invalid
4.	123456789	123abc	valid
5.	1234567891	123abc	invalid
6.	12345679a	123abc	invalid
7.	123456789	123	invalid
8.	123456789	abc	invalid
9.	123456789		invalid
10.			invalid
11.	123456789	1a	invalid
12.	123456789	123456789abcdefghi	invalid

Figure 18 (Login cases)

As seen in the Test case tables above, there are several possible test cases. Although this is a very achievable plan, in order to save time (both writing and execution), we have decided to reduce the number of scenarios by doing some equivalence partitioning. In order to do this, we have used Pairwise, an online tool that generates the smallest number of scenarios.

3.5. When to stop testing

There are no specific rules as to when to stop testing, the only rational one being when everything has been covered, but as mentioned before, that this option is not feasible in a project of this size.

The next best thing is to look at the feasibility of the option of finding the next defect, does the cost of finding the next error exceed the expected loss, if it does and the development team is satisfied with the quality and coverage goals have been met, it is a viable option to stop testing and release the application.

Another reason to stop is when the project leader or the boss says the necessary requirements have been met and the application is ready to be shipped.

3.6. Test coverage

In software testing, test coverage is a metric which measures the amount of testing done by a set of tests and how much of the program is actually tested once they are ran. The bigger the coverage the better, as it increases the chances of discovering faulty code leading us to fix the errors and improve the overall quality of the application.

As 100% test coverage is impossible due to various constraints (time being the biggest factor in our case) developers need to select the most important scenarios to be covered as much as possible. The most important scenarios (high risk ones) are the ones which will be accessed/used the most.

The following table (figure 19) describes our minimum, mandatory test coverage levels, in relation to a tasks risk level.

Risk level	Coverage level
High	80%
Medium	50%
Low	20%

Figure 19

4. Database

Due to the nature of this project, we decided to implement the business logic twice: once in a Code approach, once in a Database approach.

The focus of this part of the project was getting as much business logic as possible, to run just like it would've in the generic Code approach; and to better understand and see how they would converge in a practical manner, concepts like: normalization forms, ER/EER, etc. Most of the functionality in the Database approach was implemented using triggers, views and stored procedures, although it still relies on some C# code in order to communicate with the end users via the internet.

4.1. Types of databases

When considering the correct database for our project, we had the option of choosing between four options: the standard relational databases, non-relational databases, or go with a left field choice such as data warehouses or a solution which is radically gaining popularity, blockchain. To come to a final decision, we have decided to list the pros and cons of each solution to make an informed final decision to see which approach fits our needs the most.

4.1.1. Relational database

Relational databases are great at organizing and retrieving structured data. All data is stored and accessed with the help of relations (usually described as tables). This is the most widely used solution as its very robust and great for fast transactions and is our choice for the project.

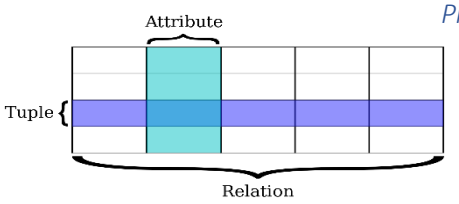


Figure 20

Pros

- Relational databases work with structured data
- They support ACID transactional consistency and support "joins"
- They come with built-in data integrity and a large eco-system
- Relationships in this system have constraints
- There is limitless indexing. Strong SQL

Cons

- Relational Databases do not scale out horizontally very well, only vertically
- Data is normalized, meaning lots of joins, which affects speed
- They have problems working with semi-structured data

4.1.2. Non-relational database

Non-relational databases or NoSQL are best applicable when data is inconsistent, incomplete or its amount is massive, this approach is very popular in the bigdata field. The top NoSQL database engine, which we are familiar with, and is a document store database, would be MongoDB. NoSQL databases are also a great option for the project as they scale better (better results without the need of upgrading existing hardware) and are more flexible (can accommodate different types of documents as there is no set schema) compared to traditional SQL databases.

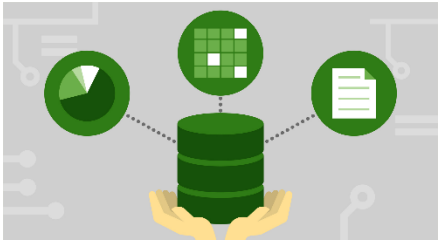


Figure 21

Pros

- They scale out horizontally and work with unstructured and semi-structured data
- Schema-free or Schema-on-read options
- High availability
- Many NoSQL databases are open source and so “free”

Cons

- Weaker or eventual consistency (BASE) instead of ACID
- Limited support for joins
- Data is denormalized, requiring mass updates
- Does not have built-in data integritybi
- Requires considerable training, to be used properly, although quite easy for simple use cases

4.1.3. Blockchain[i]

Blockchain, while a radically different approach to the traditional SQL/NoSQL solution it would be interesting to see how this innovative way of storing data could bypass the need for a future system administrator and open up possibilities to create decentralized applications or Dapps for short, which members of GTL could use to track the status of books or even be incentivized to borrow books in the form of rewards.

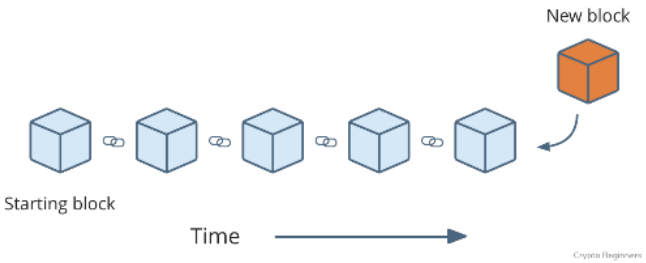


Figure 22

Pros

- Transactions are immutable
- All transactions are linked to one and other and are stored forever, great for the statistics/analysis requirement given by GTL
- High availability
- Increased transparency over traditional databases

Cons

- Can be costly depending on the type of implementation
- Low interoperability with traditional databases
- No experience with the technology

4.1.4. Datawarehouse

Data warehouses compared to regular databases use a different design, the latter focuses on being optimized for strict accuracy by being able to rapidly update real-time data. Compared to the fast response nature, of operational databases, data warehouses are designed to be a non-volatile alternative, trading transaction volume to data aggregation and are designed to scale and are great for analytics. We decided

to not choose Data warehouses as we need to focus on handling transactions when monitoring books and handling users (although a great solution for the analytic requirements of the project)

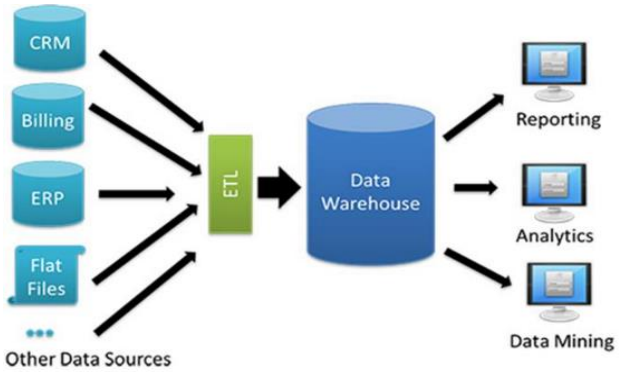


Figure 23

Pros

- Great for analytics on transactional data for businesses with high IT resources
- Error identification and correction, as data warehouses show inconsistencies and correct them before loading data
- Increased query performance (when talking about large volumes of data) and is easier to extract information

Cons

- Preparation is often very time consuming as considerable efforts are needed to create a quality storage solution with properly compatible storage and retrieval
- Security flaws, as inadequate data protection
- Needs regular maintenance regularly which can provide a costly downtime

4.1.5. Chosen database

By reviewing the database choices, we had for this project we considered the use of relational databases, non-relational databases, blockchain and data warehouse. Information provided stated that there are approximately 250 thousand volumes and 10 percent of them are out at loan at any given time, meaning, that close to 300,000 new entries would be created in one year just by lending books. However, it was determined that relational database's performance will not be seriously affected with this growth of information, as the most commonly performed action: create is not affected by data size change and all other actions are expected to have their execution time increase no more than half a second in the following 30 years (if the lending rate is not to increase/stays consistent)[ii]. Furthermore, SQL has the following benefits:

- We have more experience working with relational databases (less time is wasted getting to know new technology)
- Support for ACID
- Limitless indexing
- Relationships have constraints
- Data is structured and normalized

4.2. Database Engines

As for engines, there are several choices that we considered, for a relational database, some of which are: Oracle Database, SQL Server and MySQL; and since all three of them were using dialects of the same language (SQL), it came down to the very basics when we made the decision on which to use. As a final decision, we chose SQL Server 2017, because of the following:

- SQL Server executes and commits each instruction, unlike Oracle which requires explicit commands to commit the changes
- It's included in visual studio 2017
- Ease of use, since not only were we thought on how to use it, but also compared to Oracle, which gives so many other settings and configurations that can be set to the wrong value and effect the performance
- Performance

4.3. System-to-Database Middleware

When deciding on which data access technology to use for development of the application, we came to the 3 most popular technologies on the internet right now:

1. ADO.NET
2. LINQ to SQL
3. ADO.NET Entity Framework

We chose to follow in the footsteps of the most popular products, as they most likely are the best solution and they have great support for them. Afterwards we obtained knowledge of these technologies and started comparing them. ADO.NET was one of our initial vision in creating the application, although it is easier to use in difficult scenarios, we determined, that its extensions - LINQ to SQL and Entity Framework are easier to use in casual scenarios and is faster to develop for and easier to maintain than ADO.NET. After reviewing and comparing the last two options we had, we concluded, that Entity Framework is easier to maintain and more powerful than LINQ to SQL, also as of the release of .NET 4.0, LINQ to SQL is often considered by many to be an obsolete framework.

Entity Framework approach

When starting work in Entity Framework, we must decide which of the following methods to use for our project:

4.3.1.1.1. Model-First

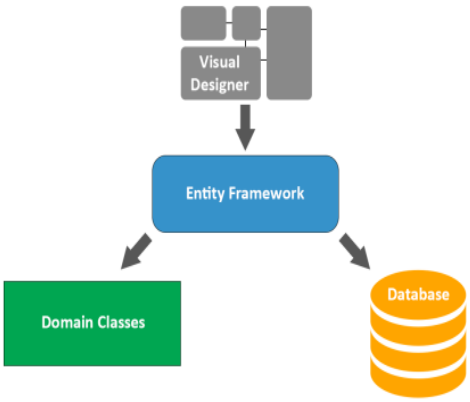


Figure 24 (Model first)

Pros:

- Great when there is need to have a visualization of the database
- Easy to use and understand when dealing with large data structures
- Models can be updated accordingly, without data loss

Cons:

- Autogenerated SQL scripts can lead to data loss in case of updates
- Hard to have precise control over generated model classes

4.3.1.1.2. Database-First

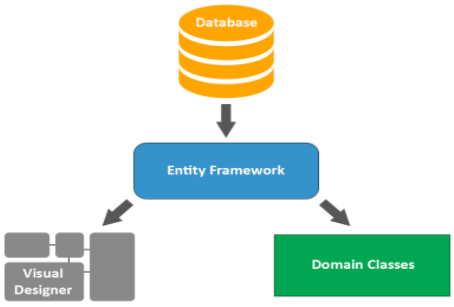


Figure 25 (SQL script used to create the Database and from that

Pros:

- Easy to use existing database for creating models
- Change will always be performed on the database, so no data loss can occur

Cons:

- Can be hard to update database when dealing with multiple instances
- Less control over generated model classes, even more so than the Model-First approach

4.3.1.1.3. Code-First

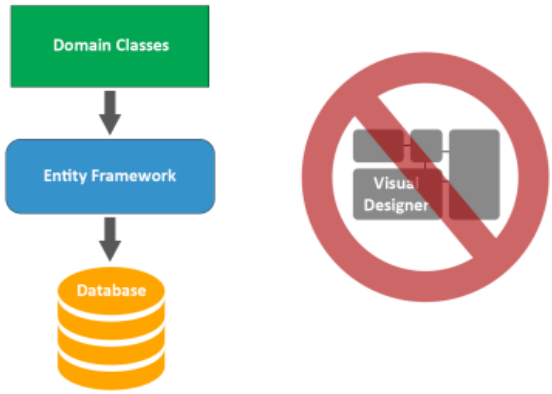


Figure 26 (Model classes from project used in Entity Framework generate the database accordingly to it)

Pros:

- No need for diagrams
- Easy to maintain and develop code
- Ideal for small-to-medium sized projects
- Saves development time
- Good control over generated database

Cons:

- Requires knowledge of object-relational mapping
- Maintaining the database can be tricky without suffering data loss (Mostly overcome by using Migrations, added in EF 4.3)

4.3.1.1.4. Chosen approach

After reviewing the options, we decided to go with the Database-First approach as we did not want to waste too much of our time on making the solution work, since we do not have a lot of time to begin with. Besides the already mentioned reason, this solution by many is considered one of the best for projects with main concern being development of the database, we wanted to gain more experience in using this approach, have an easily maintainable code and there was no real need for diagrams. In the end the actual cons of Database-First were rendered useless, since they do not affect us at all, or are not the main concern of this task.

4.4. Conceptual database design

4.4.1. ER

An Entity Relationship diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships and their attributes.

The Entity Relationship Diagram was made by reviewing the Georgia Tech Library’s request for the solution. Based on the information provided an early implementation of the diagram (figure 27) was created, this would be further improved in future iterations. To make our ER diagram we found the initial entity types (member, member card, librarian, library, book) and their attributes, following that

the relations between the entities were established and gradually perfected until the development group was satisfied with the results.

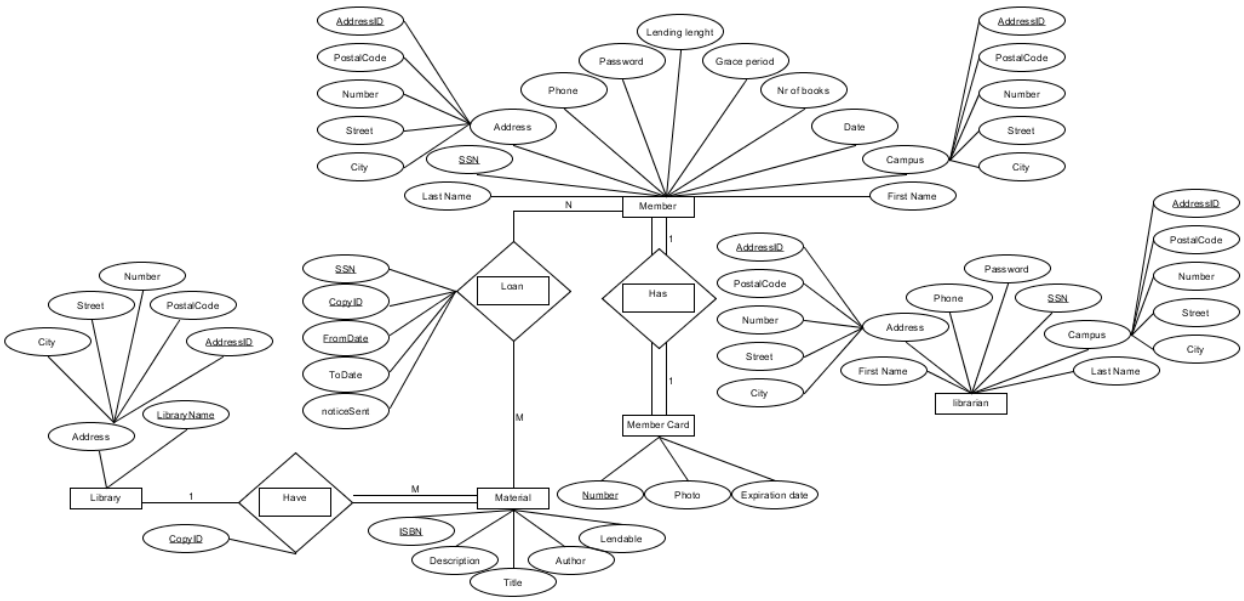


Figure 27 (ER diagram)

Focus when creating this diagram was set on borrowing books, as it is the main use case of a library. It was determined, that to identify a unique lending of material, SSN and CopyID would not be enough, as books then could not be repeatedly borrowed by the same member. Furthermore, it was decided to have each book reference the library from which it has been borrowed from, as GTL has agreements with other libraries allowing them to share books between them. To give a more detailed view of what was required it was decided to further improve this diagram by using an Enhanced ER diagram.

4.4.2. EER

Enhanced entity-relationship (EER) diagrams are basically an expanded upon version of ER diagrams. EER models are helpful tools for designing databases with high-level models. With their enhanced features, we can plan databases more thoroughly by delving into the properties and constraints with more precision.

An EER diagram provides us with all the elements of an ER diagram while adding:

- Attribute or relationship inheritances
- Category or union types
- Specialization and generalization
- Subclasses and super classes

Both ER and EER diagrams provide the ability to design our database, while ER diagram gives us the visual outlook of our database (it details the relationships and attributes of its entities, paving the way for a smooth database development in the steps ahead), the EER diagrams, on the other hand, are perfect for taking a more detailed look at the information, so we opted to create one.

Since we wanted our diagram to have higher amount of detail than provided by ER, we created Enhanced Entity Relationship Diagram, by using the previously made ER diagram. The results of this can be seen in figure 28.

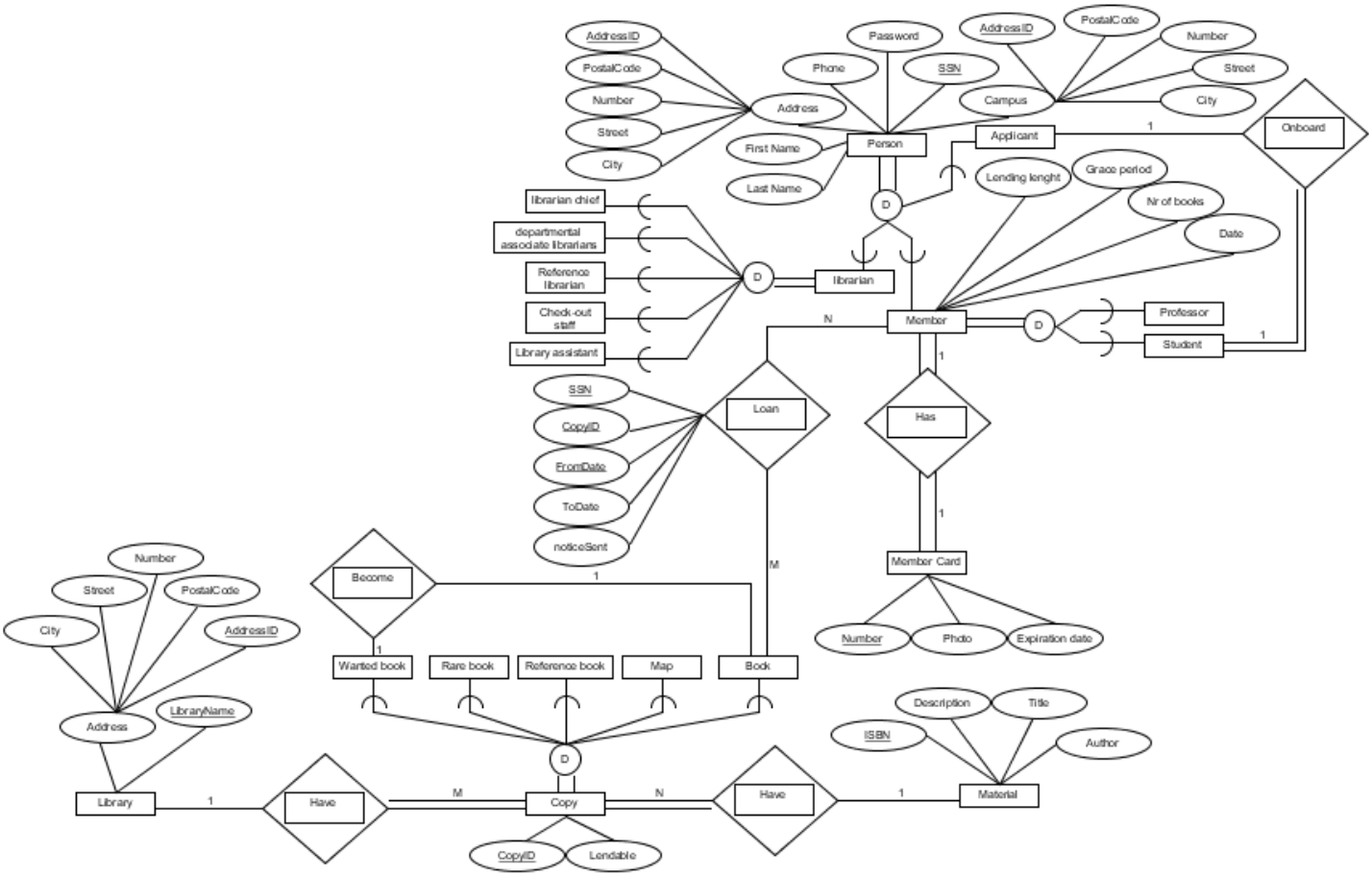


Figure 28 (EER diagram)

The connection between figures 27 and 28 can be easily observed, with main difference there being the implementation of subclasses and super classes. The reason to add them is that each of the specializations have their own rules and when making the solution we did not want to forget about them, causing us to solve the issue later at a higher cost.

4.4.3. Normalization forms

Normalization is performed, to facilitate reduced data redundancy and improved integrity of the information. Normal forms are very important for relational databases and if not followed could result in the database being impossible to use and/or inefficient. This could affect the project in a major way in the long term, possibly requiring a complete rework of the system. There are many different normalization forms, with each being a step towards improved data management. The following is a list of normal forms which are satisfied by our database:

1. First Normal Form
    a. Atomic values (no composite and multivalued attributes)
    b. Doesn't have repeating groups
    c. Identify each tuple with a unique primary key
    d. Disallows nested relationships
2. Second Normal Form
    a. Full dependency on the primary key
3. Third Normal Form
    a. No transitive dependency (All the non-prime attributes must depend on the primary key only)
4. BCNF (Boyce-Codd Normal Form)
    a. Prime attribute dependent on a non-prime attribute

The implementation of the normal forms can be seen in the following paragraph, where the final database diagram is displayed.

4.4.4. Database diagram

Database diagram shows how the information in the database is organized and thus gives the reader a better understanding of how it works and what can be done with it. In a relational database, the schema can define information like the tables, fields, relationships, views, indexes, procedures, triggers, and other elements.

When we started work on creating the database schema it was decided to use the following steps, to be sure information saved is well representative of what was requested by the project, shown in the EER diagram. (first 7 steps are shared with the ER diagram, but the last two are unique only to the enhanced version of ER)

1. Mapping of Regular Entity Types
2. Mapping of Weak Entity Types
3. Mapping of Binary 1:1 Relation Types
4. Mapping of Binary 1:N Relationship Types
5. Mapping of Binary M:N Relationship Types
6. Mapping of Multivalued attributes
7. Mapping of N-ary Relationship Types
8. Mapping Specialization or Generalization
    a. Multiple relations-Superclass and subclasses
    b. Multiple relations-Subclass relations only
    c. Single relation with one type attribute
    d. Single relation with multiple type attributes
9. Mapping of Union Types (Categories)

Our database diagram, just like the EER, changed quite a lot early on, but we managed to get a solution satisfying all the needs quite fast. As seen in figure 29, we translated the EER into a database diagram which we later implemented and used throughout the project for data management. Mapping of specialization was done using "8a" for the super class: person, and subclasses: member, librarian. This was done as we wanted all closely related information (persons) to be saved in one easily accessible table. The rest of inherences (different types of: materials, librarians, members) were mapped using "8c", but none of them had any unique attributes to ones provided by the superclass, only by having a different business logic associated.

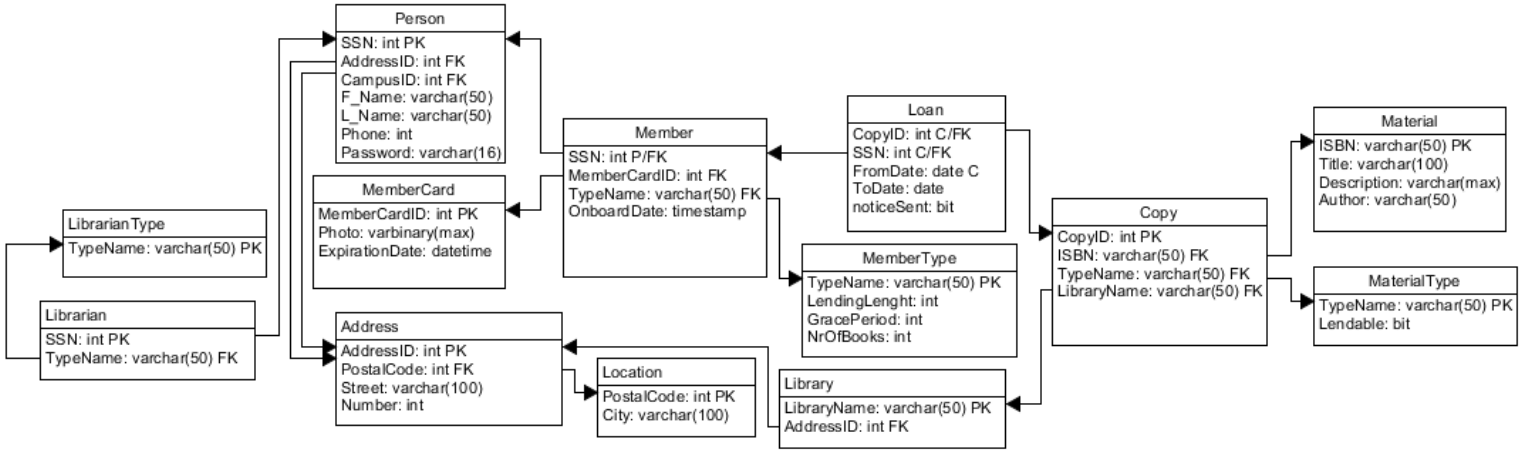


Figure 29 (Database diagram)

4.5. Triggers

One of the main focuses in the development of this project was the database and information management performed in it. Since we chose to work on the relational databases throughout the extent of this project we wanted to use as many of the features provided by SQL Server as possible. Triggers being one of the most important and used procedure, were a necessity to be implemented.

4.5.1. What are triggers

Triggers in relational databases are special stored procedures that are executed automatically in response to an event occurring in the database server.

SQL Server provides three type of triggers:

1. Data manipulation language (DML) triggers which are invoked automatically in response to events against tables or views (INSERT, UPDATE, and DELETE). They fire when any valid event fires, whether table rows are affected or not
2. Data definition language (DDL) triggers fire in response to statements changing the data behavior (CREATE, ALTER, and DROP) and certain system stored procedures that perform DDL-like operations
3. Logon triggers fire when a user's session is being established

4.5.2. How the use of triggers was chosen

The use of triggers was determined based on the requirements provided by the GTL. Although a higher number and different types of triggers could have been used when developing the solution, we decided to focus more on the basic requirements of the library system, thus growing our project slowly with future development and usability in mind. Furthermore, it was not determined that more extensive use of triggers was not requested or information necessary to create more triggers was not provided.

4.5.3. What triggers were used for

In the solution provided alongside this document it was decided to implement only one trigger, that being for inserting a new entry into the "loan" table. We chose to implement this trigger since we wanted to be sure that the book being borrowed would be available for borrowing (book not being loaned out already) and so that the member would not be able to borrow more than the allowed limit of materials. If either of the two conditions were violated, then the transaction would be rolled back and no changes to the database would be made. However, if none of the previously mentioned rules were broken the entry would be saved in the database updating the "FromDate" field (See figure 30), so that we ensure the wrong date, which the user could have entered to modify the expected results, would not be saved.

```
CREATE OR ALTER TRIGGER Lending
ON Loan
FOR INSERT
AS
BEGIN
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
    BEGIN TRANSACTION
    DECLARE @CopyID INT;
    DECLARE @SSN INT;
    DECLARE @FromDate Date;
    SELECT @CopyID = CopyID, @SSN = SSN, @FromDate = FromDate FROM INSERTED

    DECLARE @NrOfBooks INT;
    SELECT @NrOfBooks = NrOfBooks
    FROM Member INNER JOIN MemberType ON Member.TypeName = MemberType.TypeName
    WHERE SSN = @SSN;

    IF (SELECT COUNT(*) FROM Loan WHERE SSN = @SSN AND ToDate IS NULL) > @NrOfBooks OR --user limit not exceeded
        (SELECT COUNT(*) FROM Loan WHERE CopyID = @CopyID AND ToDate IS NULL) > 1 --book is available
        BEGIN
            ROLLBACK
        END
    ELSE
        BEGIN
            UPDATE Loan
            SET FromDate = GETDATE(), ToDate = NULL
            FROM Loan
            WHERE CopyID = @CopyID
                AND SSN = @SSN
                AND FromDate = @FromDate
            COMMIT
        END
End
GO
```

Figure 30 (Lending trigger script)

4.6. Script optimization

4.6.1. Why it's important

Performance is quite an important part of any software solution, no matter if we're talking about the applications in which the users click buttons to display data or if we're writing a query directly into, let's say Sql Server Management Studio (SSMS). Nobody likes to click a button, go get a coffee and then hope the results are ready by the time they come back. As computers get faster and technology moves forward users get more impatient and want things immediately, without having to wait.

The SQL Select statement is the primary mechanism to retrieve data from a relational database. Often even clicking a single button requires query performance optimization because everything that's actually happening under the hood is just SQL Server pulling the data from a database. Therefore, we need to make sure that our queries are performing well. This is something that we can design with a focus on the query performance, but we can also find and troubleshoot slow performance queries by identifying bottlenecks in them.

4.6.2. How we did it

We decided to improve the performance of our queries, by following a few simple rules, when composing different database interactions. And those simple rules are:

- Adding as many "WHERE" clauses as possible: This was done simply because the more filters we put in the query the less data will have to be processed in future actions
- Selecting only columns that we need: By simply selecting the specific columns we needed, instead of performing a "SELECT(*)" on the tables, we can drastically reduce the amount of data, the server will have to process in order to finish executing the query
- Mindful use of the "JOIN" statement: Joining tables generally is a resource intensive action, because the server has to check each of the entries in both of the tables. Therefore, using as little "join" statements as possible, and even when they're used, they're applied on indexed columns, drastically improves the overall performance
- Revisiting indexes: Creating an index on a table is like a double edged-sword. On one hand it can help reading information from that table, on the other, inserting or updating existing information can cause major problems, in terms of performance. Therefore, revisiting existing indexes and even temporarily removing them, while bulk data is inserted/updated, can be a great idea
- Moving queries to stored procedures
- Adding index on attributes which are used for = conditions; or are used for joins
- Refactoring queries that use "IN" to use "JOIN" instead

Those were only a few ways we used to improve our queries. But one other way we used, especially for some of the more complex queries, like the query seen in (Fig 31) was the use of execution plans. A query execution/explaination plan AKA execution plan is an ordered set of steps used to access data in a SQL Server. It's basically a map that the SQL Server is drawing to the shortest, ideally the most efficient,

path to the data in our database. Such a plan is created when a query is accepted by the SQL Server and it's coming from either an application or it's coming from us when testing query performance.

```
use GTL
go
    WITH a AS (
      SELECT Material.ISBN, Material.Title, Material.Author, Material.Description, copy.LibraryName as Location, copy.TypeName
      FROM Copy LEFT JOIN Loan ON Copy.CopyID = Loan.CopyID
       INNER JOIN Material ON Material.isbn = Copy.isbn
       WHERE Loan.CopyID IS NULL OR (Loan.ToDate IS NOT NULL and Loan.CopyID IS NOT NULL)
      )

    SELECT DISTINCT Material.ISBN, Material.Title, Material.Author, Material.Description, copy.LibraryName AS Location, copy.TypeName,
    (SELECT COUNT(*) FROM a WHERE copy.TypeName LIKE a.TypeName AND a.Location LIKE copy.LibraryName AND a.ISBN = copy.ISBN) AS Available_Copies
      FROM Copy INNER JOIN Material ON Material.isbn = Copy.isbn
```

Figure 31 (Read all materials and the amount of available copies, view script – unoptimized)

Now, there are two types of execution plans:

- Estimated execution plan: an execution plan generated by the SQL server using various statistical tools, in order to "guess-timate" the most efficient way this query can be converted into smaller tasks
- Actual execution plan: an execution plan that is generated only after the query was executed against a specific database. This execution plan is way more useful when it comes to optimizing a query as this plan contains information such as: run time, returned number of entries, etc. But it also comes with the downside of requiring to actually execute the query first, which might not be the brightest idea if the only available database, is the one that the customers are currently using

Since we only have one database, the one we used to develop this application, we decided that the actual execution plan would yield better results when trying to optimize our queries.

Optimization process

The following part would depict the process of our query optimization, with the help of a specific query, depicted in figure 31. This entire process was done using ApexSQL, a software solution for SQL query optimization, that smoothens the entire process with a slick and intuitive user interface.
figure 32 represents the execution plan of the query presented in figure 31. And we can see that its original execution time is at 94ms. Not too bad, but it could be improved.

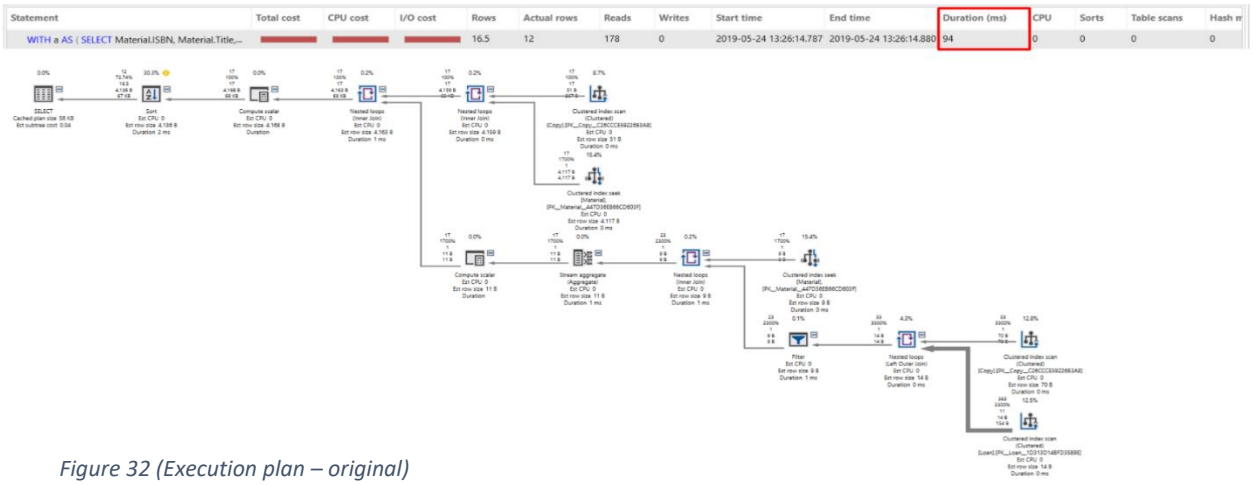


Figure 32 (Execution plan – original)

After refactoring the query, by making it "JOIN" the already existing table "a" instead of re-joining with "materials", the execution time has been drastically reduced, as can be seen in figure 33, to 14ms, even though the number of reads performed has essentially doubled.

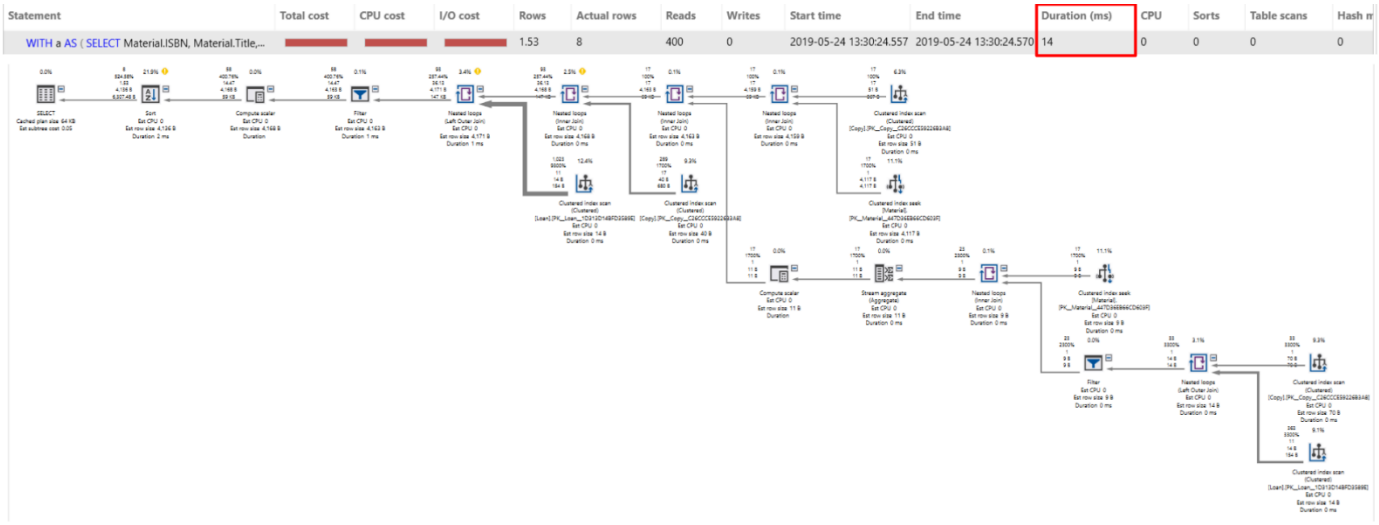


Figure 33 (Execution plan – after refactoring)

Now 14ms, is a pretty good time, and as far as we can see, there isn't too much optimization that can be further performed on the query itself. However, we can still improve the execution time, by changing how the data inside some of the used tables, is saved. And in that regards, we can create an non-clustered index on the "copy.isbn" field, which would result in the following execution plan (figure 34).

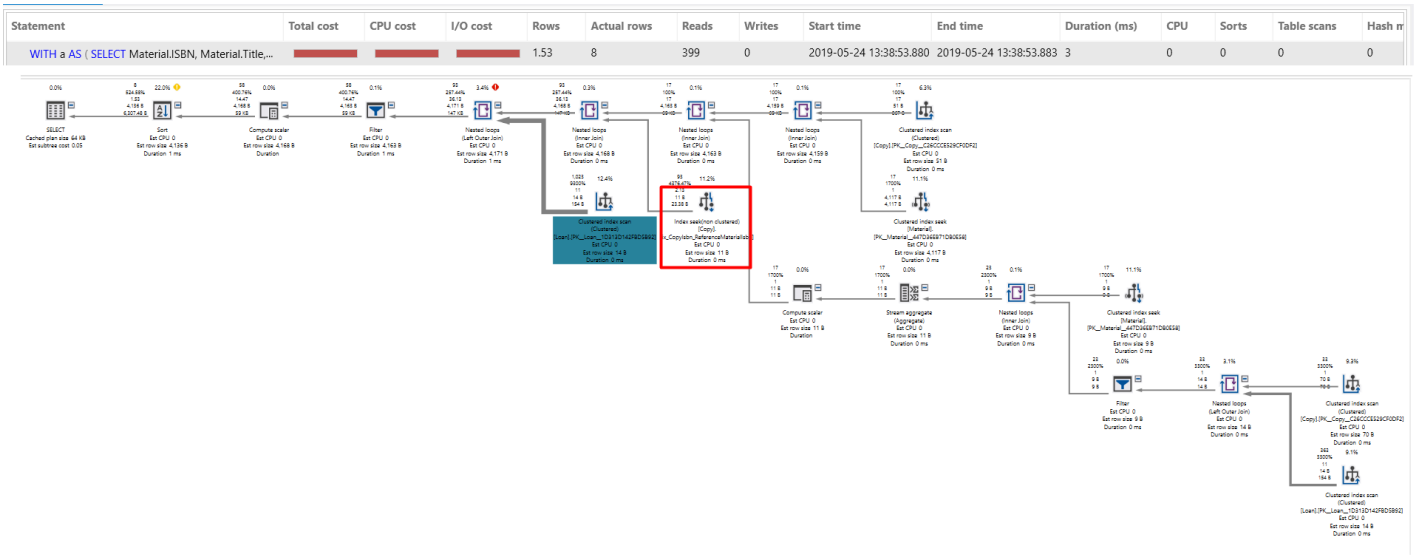


Figure 34 (Execution plan – indexed Copy)

The simple creation of an index resulted in a 3ms execution time. Change done by the fact that a "index scan" became an "index seek" action inside the execution plan.

Now there are still some table "scans" inside the current execution plan, scans which perhaps could be changed to "seek" actions instead. Changes which will further reduce the execution time of this particular query. One good example being the "index scan" performed on the "loan" table, however due to the fact that the "loan" table will be under constant "threat" of writing and updating actions; in fact, according to the problem statement, we expect around 300.000 writes and around 299.250 updates (5% of the overdue books are never returned), per year; we decided to keep the "loan" table with as few indexes as possible.

4.7. Views

Database views are searchable virtual objects in relational databases. Unlike ordinary base tables a view is not part of a physical schema and they are searchable and defined by stored queries. Views can consist of two or more combined tables, if data is changed in one of the tables used to create the view, the view will be updated once the view in question is called again.

The benefits of using views are:

- Storage – as storing views only consist of storing the definition of it and not an actual copy, it takes up very little space in the database and helps us avoid needing a large storage solution with the absence of duplicate data
- Helps with enforcing business logic – with the use of views it is easier to define business rules, such as when a book is borrowed and by who it is borrowed

The tradeoffs of using views:

- Performance – to create views we need to query multiple tables, it can take a toll on the database engine
- Update restrictions – depending on the complexity of the view, it might be a read only, as updating it would require the update of the tables which it consists of

As shown in (figure 35), we create the view of the most loaned books by joining the materials, copy and the loan table as the count was only achievable this way.

```
DROP VIEW IF EXISTS topLoanedBooks
GO
CREATE VIEW topLoanedBooks AS
    WITH a AS (
    SELECT ISBN FROM Loan
    JOIN Copy ON Copy.CopyID = Loan.CopyID
      )

    SELECT DISTINCT top(10) a.ISBN, Material.Title, Material.Author, Material.Description,
    (SELECT COUNT(*) FROM a WHERE a.ISBN = copy.ISBN) AS loaned_count
      FROM Copy INNER JOIN Material ON Material.isbn = Copy.isbn
      JOIN a ON a.ISBN = Copy.ISBN
GO
```

Figure 35 (Top loaned books – view script)

5. Implementation

5.1. Database

5.1.1. Concurrency

Concurrency is the execution of multiple transactions, which may access the same database rows during overlapping time period. Such simultaneous accesses (called collisions) may result in errors or inconsistencies in data if not handled properly, causing the following phenomena to arise:

- Lost Update- a transaction is rendering previous one obsolete
- Dirty Read- a transaction can read data from a row that has been modified by another running transaction and which is not yet committed
- Non-Repeatable Read- during a transaction, a row is retrieved twice and the values within the row differs between reads
- Phantom Read- while a transaction, new rows are added or removed by another transaction to the records being read

To deal with the already mentioned issues there are two commonly used approaches, with each of them tackling the problem in different ways:

- Pessimistic locking: used when collisions are anticipated, the transactions violating synchronization are simply blocked. May cause deadlocks if not properly implemented
- Optimistic locking: costs less to do operations, especially when many collisions are not expected, but if the collisions occur the transaction should abort, losing the progress

For our solution we opted to use pessimistic locking (explained further in section transactions). However, the optimistic locking might have been better solution for tasks like borrowing and returning books. The decision to use pessimistic locking everywhere was made, because the intended amount of operations, that this application would experience during a day, no noticeable change should be seen and the group was on a tight schedule to deliver a working product before the deadline, but in the future it will not be hard to update the code implementing the optimistic approach.

5.1.2. Isolation levels

Transactions specify an isolation level that defines the degree to which one transaction must be isolated, from the resource or data modifications, made by other transactions. A transaction always gets an exclusive lock on any data it modifies and holds that lock until the transaction completes, regardless of the isolation level set for that transaction. A lower isolation level increases the ability of many users to access data at the same time, but increases the number of concurrency effects, such as dirty reads or lost updates, that users might encounter. Conversely, a higher isolation level reduces the types of concurrency effects that users might encounter but requires more system resources and increases the chances that one transaction will block another. Choosing the appropriate isolation level depends on balancing the data integrity requirements of the application against the overhead of each isolation level.

The following table (figure 36) shows the concurrency side effects allowed by the different isolation levels.

Isolation Level	Dirty Read	Non-Repeatable Read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed	No	Yes	Yes
Repeatable read	No	No	Yes
Snapshot	No	No	No
Serializable	No	No	No

Figure 36

5.1.3. Transactions

A transaction is a package consisting of a single or more commands, that is executed as one unit of work. If a failure occurs at one point in the transaction, all of the updates can be rolled back to their pre-transaction state. Transactions that involve multiple resources can lower the concurrency, if locks are held too long, therefore, it is advisable that transactions are kept as short as possible. Our wish to have the data consistent and reliable meant that while developing the application we have made use of transactions in places where they were necessary (shown in figure 37). With each of the transactions using isolation levels, we determined to best fit the requirements of the task.

```
DROP PROCEDURE IF EXISTS DeleteCopy
GO
CREATE PROCEDURE DeleteCopy @SSN int, @CopyId int
AS
BEGIN
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED
    BEGIN TRANSACTION
    IF EXISTS(SELECT * FROM Librarian WHERE Librarian.SSN = @SSN) AND
       EXISTS(SELECT * FROM Copy WHERE Copy.CopyID = @CopyId)
       BEGIN
        DELETE FROM Copy WHERE Copy.CopyID = @CopyId
        COMMIT
        SELECT 1
        END
    ELSE
        BEGIN
        ROLLBACK
        SELECT 0
        END
END
GO
```

Figure 37 (Delete copy – Procedure script)

5.1.4. Security

An SQL Server instance contains a hierarchical collection of entities, starting with the server. Each server contains multiple databases, and each database contains a collection of securable objects (tables, views etc.). Every SQL Server securable entities, have associated permissions that can be granted to an individual, group or process. The SQL Server security framework manages access to securable entities through authentication and authorization:

- Authentication establishes the identity of the user or process being authenticated
- Authorization is the process of determining which resources and operations are the authenticated users allowed to use

Every securable entity, has permissions that can be granted to a user using permission statements. Granting permissions to roles rather than to users simplifies security administration, as permission sets that are assigned to roles are inherited by all members of the role. When assigning permissions to database users developers should always follow the principle of least privilege (grant the minimum permissions necessary to a user or role to accomplish a given task).

Currently our database does not use any permissions or roles, as we were still developing the solution at the time of writing this document and planned to focus more on security after we had established some basic application satisfying the needs of GTL.

5.1.4.1. Encryption

SQL Server provides functions to encrypt and decrypt data using a certificate, asymmetric key, or symmetric key. However, it was decided to not implement it, as all expected communications, at the moment, would be performed through our application, which is located on the same machine, and this would result only in degraded performance with little to no improvement to security.

5.1.4.2. SQL Injection

Applications frequently take external input and perform actions based on the information provided, if the information is not handled properly, it could be used to inject malicious code which could make the application perform actions that it was not designed for. These attacks are called SQL injections and they are most commonly used for website but can be used for any other application using relational databases (figure 38). SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause refusal issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

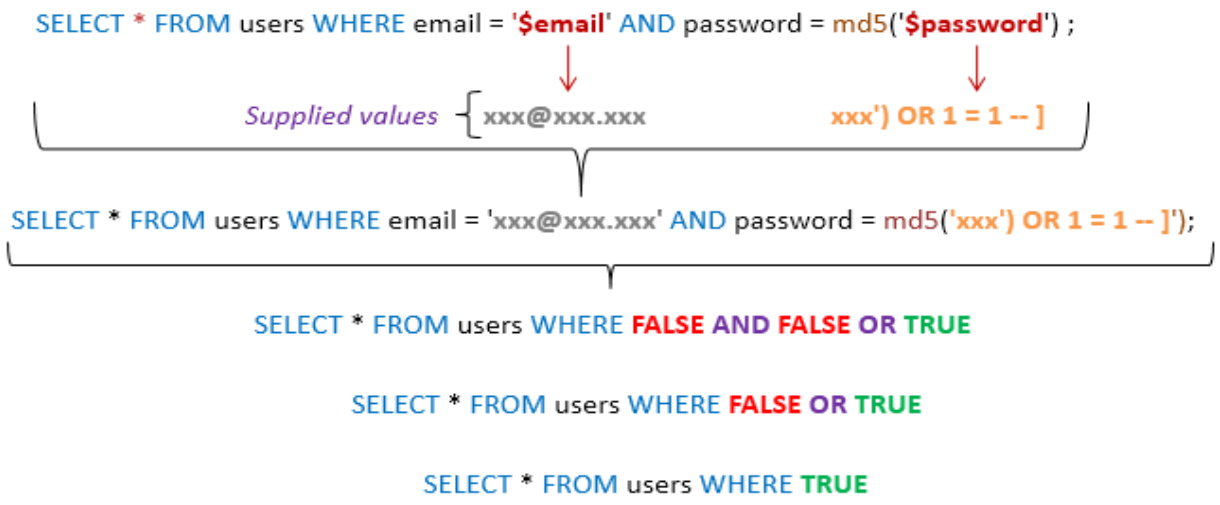


Figure 38 (Sql injection example)

The most basic way of defending ourselves against simple attacks like these is to parameterize the results passed to database, that way the information passed will not be considered as a part of the SQL statement and will not be executed. Since we used entity framework LINQ for communication with the database, where information is automatically passed as parameter, thus making it not susceptible to traditional SQL injection attacks. Furthermore, we never used Entity SQL with input supplied by the user, thus following the steps ensuring safety provided by Microsoft when using entity framework.

5.1.5. Procedures

5.1.5.1. *Stored Procedures*

A stored procedure is a prepared SQL query that we can save, so it can be reused multiple times. Stored procedures resemble constructs found also in other languages, by allowing input parameters and return values; they contain programming statements that perform various operations and return a status value to a calling program to indicate success or failure (and the reason for failure). This made it attractive to our solution, where we wished to do as much as possible in the database itself. The use of stored procedures, not only reduced the duplication of code which could have arisen, but also let the solution to be easier to maintain.

Figure 39 shows one of the main procedures found in our solution, that being creation of materials. This procedure has 8 parameters which have to be provided in order to perform the enclosed statement. One of the first actions that are performed, when calling the procedure is establishing transaction (since we wanted to be sure each call to database is atomic), following that, the intended operations are performed on the data, ending with changes being either committed or undone.

```
DROP PROCEDURE IF EXISTS CreateMaterials
GO
CREATE PROCEDURE CreateMaterials @SSN int, @ISBN int, @library varchar(50), @Author nvarchar(50), @Description varchar(max),
                                 @Title nvarchar(100), @TypeName varchar(50), @Quantity int
AS
BEGIN
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
    BEGIN TRANSACTION
    IF EXISTS(SELECT * FROM Librarian WHERE Librarian.SSN = @SSN) AND
        EXISTS(SELECT * FROM Library WHERE Library.LibraryName LIKE @library) AND
        EXISTS(SELECT * FROM MaterialType WHERE MaterialType.TypeName LIKE @TypeName)
        BEGIN
        IF NOT EXISTS(SELECT * FROM Material WHERE Material.ISBN = @ISBN)
            BEGIN
            INSERT INTO Material (ISBN, Title, Description, Author)VALUES (@ISBN,@Title, @Description, @Author)
            END
        WHILE @Quantity > 0
            BEGIN
            INSERT INTO Copy (ISBN, TypeName, LibraryName)VALUES (@ISBN, @TypeName,@library)
            SET @Quantity = @Quantity - 1
            END
        COMMIT
        SELECT 1
        END
    ELSE
        BEGIN
        ROLLBACK
        SELECT 0
        END
END
GO
```

Figure 39 (Create materials – procedure)

5.1.5.2. Computed column

A computed column is a virtual column that is not physically stored in the table, unless the column is marked "PERSISTED". A computed column expression can use data from other columns in the table, to calculate a value for the column to which it belongs.

The idea to use computed columns was suggested and considered, but due to its limitations it couldn't be implemented as the we envisioned the solution.

5.1.6. Scheduling

Scheduling and execution of jobs, which would be needed to be performed at certain time periods, was done using SQL Server Agent. The use of SQL server agent however was limited to just running single SQL script once every day (figure 40), this would in return allow librarians to know what books have been borrowed and not retuned before the end of the grace period, thus making it simpler for them to manage it and send notices to involved members. It could have been possible to automate this even further and automatically send email messages to the members in question, but due to limitation in information required to become a user of the library system this was not yet possible to implement.

```
UPDATE Loan
SET noticeSent = 0
FROM Loan
    INNER JOIN Member ON Loan.SSN = Member.SSN
    INNER JOIN MemberType ON Member.TypeName = MemberType.TypeName
WHERE ToDate IS NULL
    AND Loan.noticeSent IS NULL
    AND CONVERT(date, getdate()) >= DATEADD(DAY, MemberType.LendingLenght + MemberType.GracePeriod, Loan.FromDate);
```

Figure 40 (Loan – scheduling script)

5.2. Tests

In this section we will talk about how we have implemented tests in our project and why each level of testing is integral to a high-quality final product.

5.2.1. Unit tests

Unit tests are the smallest "unit" of tests. These tests are made to validate if a single individual unit of code functions as expected. These tests are basic and have no logic, they consist of a single or a few inputs and produce a single output. Unit tests have the greatest effect on quality.

In the image bellow (figure 41) we have conducted a test on materials. In this test we return a book and see if the results comply with our standards. We also mock the methods which are called by the test method.

```
public class MaterialTests
{
    [Test]
    //pass
    [TestCase("", "", 10, null, "0")]
    public void MaterialDmReadMaterialTest(string materialTitle, string author, int numOfRecords = 10, int isbn = 0, string jobStatus = "0")
    {
        //Arrange
        var materialDa_Code_Mock = new Mock<MaterialDa_Code>();
        var libraryDa_Code_Mock = new Mock<LibraryDa_Code>();
        var personDa_Code_Mock = new Mock<LibrarianDa_Code>();
        var copyDa_Code_Mock = new Mock<CopyDa_Code>();
        var lendingDa_Code_Mock = new Mock<LoaningDa_Code>();
        var context_Mock = new Mock<Context>();
        var objects = MaterialsSetUp();

        materialDa_Code_Mock.Setup(x => x.ReadMaterials(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>(), It.IsAny<int>(), It.IsAny<Context>()))
            .Returns(objects.Item2);
        copyDa_Code_Mock.Setup(x => x.ReadCopies(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<Context>()))
            .Returns(objects.Item1);

        var materialDm = new MaterialDm_Code(materialDa_Code_Mock.Object, libraryDa_Code_Mock.Object,
            personDa_Code_Mock.Object, copyDa_Code_Mock.Object, lendingDa_Code_Mock.Object, context_Mock.Object);

        //Act
        var result = materialDm.ReadMaterials(materialTitle, author, numOfRecords, isbn.ToString(), jobStatus);

        //Assert
        Assert.IsTrue(result.Count == 2);
        Assert.IsTrue(result[0].Available_Copies == 2);
        Assert.IsTrue(result[1].Available_Copies == 1);
    }
```

Figure 41

If the result complies with a favorable test case, it passes. If any parameter of the result is returned as false, it does not pass.

These tests are the first level of testing developers carry out before moving on to integration testing

```
[Test]
//pass
[TestCase(true, true, true, true, true)]
[TestCase(true, false, true, true, true)]
//fail
[TestCase(true, false, false, false, false)]
[TestCase(false, false, true, false, false)]
[TestCase(false, true, false, true, false)]
[TestCase(false, true, true, true, false)]
public void MaterialDmCreateMaterialTest(bool ssnPassing, bool isbnPassing, bool libraryNamePassing, bool typeNamePassing, bool testPassing)
{
```

Figure 42 (Test cases)

5.2.2. Integration tests

After the units of tests have been verified to be working in correct order, they are ready to be integrated into the application. At this level the units are combined, and they are tested as a group, where we search for defects when they are interacting with the already integrated units of code.

```
[Test]
//Code approach
[TestCase(0, 0, "Code", false)] //invalid ssn and isbn
[TestCase(123456785, 0, "Code", false)] //valid ssn, invalid isbn
[TestCase(0, 1, "Code", false)] //invalid ssn, valid isbn
[TestCase(123456785, 8, "Code", true)] //valid ssn and isbn
//Database approach
[TestCase(0, 0, "Database", false)] //invalid ssn and isbn
[TestCase(123456785, 0, "Database", false)] //valid ssn, invalid isbn
[TestCase(0, 1, "Database", false)] //invalid ssn, valid isbn
[TestCase(123456785, 8, "Database", true)] //valid ssn and isbn
public void DeleteMaterial(int ssn, int isbn, string approach, bool passing)
{
    //Arrange
    Setup(approach);

    //Act
    bool result = _materialService.DeleteMaterial(ssn, isbn.ToString());

    //Assert
    Assert.IsTrue(result.Equals(passing));
}
```

Figure 43

In this test we test the code by giving invalid attributes and see if we are thrown an error or not. We expect this test to pass and if the value in the passing parameter to be true, it means the unit of code has been successfully integrated and functions as intended.

5.2.3. System testing

After the successful integration tests, system tests are meant to be run when the integrated software is ready to be tested as a whole and measured against the initial requirements.

5.2.4. Acceptance tests

The final level of software testing which is meant to test the software's compliance to the original business requirements. These tests in contrast to the previous ones, are best performed by a select group of individuals who represent the final users of the product. If the software passes acceptance tests it is ready to be launched and go commercial.

In this test we mock a user logging into the system. If the result of the test is the home page, the test has been successful.

```
[Test]
public void As_a_User_I_Can_Log_in()
{
    //Arrange
    string ssn = "123456789", password = "test";
    IWebElement ssnBox = _chromeDriver.FindElement(By.Id("SSN"));
    IWebElement passwordBox = _chromeDriver.FindElement(By.Id("Password"));
    IWebElement navBarElement = _chromeDriver.FindElements(By.ClassName("nav-link"))[0];

    //Act
    ssnBox.SendKeys(ssn);
    passwordBox.SendKeys(password);
    _chromeDriver.FindElement(By.Id("Login")).Click();
    WaitForStaleness(navBarElement);

    //Assert
    string element = _chromeDriver.FindElements(By.ClassName("nav-link"))[0].Text;
    Assert.IsTrue(element.Equals("Home"));
}
```

Figure 44 (Login - Acceptance test)

5.3. WCF

Defining an endpoint is necessary as it is the only way of communicating with it. To define an endpoint we need its address, its binding and its contract (the ABCs of WCF)

- Address defines where a service is hosted, an address can be an IP address, a server name or even a URL
- Binding defines how messages are handled in both the client and server side. There are multiple binding types each with their own strengths and weaknesses. We chose to use the HTTP binding as it is very well suited for connecting web services
- Contract is the agreement between the client and the server that specifies the structure and the contents of the messages sent and received. The data which is the agreement's structure of the message, and the message contract is about the content

To communicate with the service establishing communication is necessary, meaning a client proxy needs to be created, this is done by adding a service reference to the client which will be using it.

Dependency injection is great as it means code becomes less coupled and more flexible as the dependencies of classes are injected during runtime and it simplifies the running of tests by mocking dependencies in places of real classes. For our project we used Castle Windsor, a dependency container which is extensively used by the developer community and comes preloaded with WCF integration facilities.

Using Windsor we can instantiate an object required in the parameter of the class's constructor. This allowed us to use the dependency injection, even though the default constructor of WCF service should be parameter-less. In the already shown snippet of the code (figure 45) we can observe how it was done (If IMaterialService was required, the new MaterialService would be returned). This approach, although not being the only possible solution to this problem, allowed us to keep code simple and organized, not requiring multiple constructors for each class, while also providing improved testability.

```
namespace GTLService
{
    public class WindsorInstaller : IWindsorInstaller
    {
        public void Install(IWindsorContainer container, IConfigurationStore store)
        {
            string approach = "Code";
            //used: http://scotthannen.org/blog/2016/04/13/wcf-dependency-injection-in-5-minutes.html
            container.Register(
                Component.For<IMaterialService, MaterialService>().LifeStyle.Transient,
                Component.For<ICopyService, CopyService>().LifeStyle.Transient,
                Component.For<ILoginService, LoginService>().LifeStyle.Transient,
                Component.For<ILoaningService, LoaningService>().LifeStyle.Transient,
                Component.For<IStatisticsService, StatisticsService>().LifeStyle.Transient,
```

Figure 45

6. Conclusion

In conclusion, during this semester we managed to gain knowledge about various testing rituals, such as: test strategies, testing methodologies, test automation, and proper peer reviewal techniques; and database practices, such as: normalization forms, entity relational diagrams, and various functions.

Our project turned out to be quite close to what we imagined when we envisioned the final product. Our list of features includes:

- User authentication
- Managing materials (books, maps, etc.)
- Loaning and returning materials
- Ability to gain meaningful insights into user's reading behavior, through the use of statistical tools
- Automated notifications for lent books outside the grace period
- Available alpha testing at: https://gtlwebsite.azurewebsites.net/

Although there is room for improvement, for example: more features, enhanced API documentation and additional clients; we are satisfied with what we have achieved, given the time and resource limitations.

We suspect due to hand-in website limits, the entire repository cannot be uploaded. To see how we worked and what files we created, one has to follow the link, which will take the reader to our GitHub repository: https://github.com/RaidenRabit/GeorgiaLibrarySystem-

As an ending note, we would like to thank all the readers, who invested their time in reading this paper, also the guiding teachers, who helped and guided us throughout the entire process.

7. References

https://pairwise.teremokgames.com/
https://docs.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-2017
https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/overview-of-sql-server-security
https://en.wikipedia.org/wiki/Isolation_%28database_systems%29#Isolation_levels
https://docs.microsoft.com/en-us/sql/relational-databases/tables/specify-computed-columns-in-a-table?view=sql-server-2017
https://docs.microsoft.com/en-us/sql/connect/jdbc/understanding-isolation-levels?view=sql-server-2017
https://crate.io/docs/sql-99/en/latest/chapters/37.html
https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/security-considerations
https://en.wikipedia.org/wiki/SQL_injection
https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?view=sql-server-2017
https://www.ibm.com/support/knowledgecenter/en/SSZLC2_9.0.0/com.ibm.commerce.developer.doc/refs/rsdperformanceworkspaces.htm
https://www.apriorit.com/dev-blog/381-sql-query-optimization
https://www.w3schools.com/sql/sql_create_index.asp
https://docs.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-2017
http://www.sqlservertutorial.net/sql-server-triggers/
https://slideplayer.com/slide/7640839/
https://www.tutorialspoint.com/software_architecture_design/distributed_architecture.htm
https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model
https://en.m.wikipedia.org/wiki/First_normal_form
http://jcsites.juniata.edu/faculty/rhodes/dbms/eermodel.htm
https://chsakell.com/2015/10/15/wcf-proxies-from-beginner-to-expert/
http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/
https://airbrake.io/blog/sdlc/v-model
https://www.tutorialspoint.com/agile_testing/agile_testing_quadrants.htm
https://lisacrispin.com/2011/11/08/using-the-agile-testing-quadrants/
https://reqtest.com/agile-blog/v-model-versus-scrum-who-wins/
https://rbcs-us.com/site/assets/files/1203/agile-v-model.pdf
https://www.slideshare.net/dhanajagli1/3static-testing
https://whatis.techtarget.com/definition/static-testing
https://www.guru99.com/testing-review.html
https://www.tutorialspoint.com/software_testing_dictionary/static_testing.htm
https://www.guru99.com/dynamic-testing.html
https://www.tutorialspoint.com/software_testing_dictionary/dynamic_testing.htm
https://www.guru99.com/test-coverage-in-software-testing.html
https://martinfowler.com/bliki/TestCoverage.html
https://www.tutorialspoint.com/software_quality_management/software_quality_management_sqa_components.htm
https://www.geeksforgeeks.org/software-engineering-software-quality-assurance/
https://www.test-institute.org/What_is_Software_Quality_Assurance.php
https://businessanalysttraininghyderabad.wordpress.com/2014/08/05/what-is-furps/
http://agileinaflash.blogspot.com/2009/04/furps.html

https://searchdatamanagement.techtarget.com/definition/relational-database
https://aws.amazon.com/relational-database/
http://nosql-database.org/
https://www.mongodb.com/nosql-explained
https://it.toolbox.com/blogs/craigborysowich/some-pros-cons-of-relational-databases-050108
https://www.quora.com/What-are-the-pros-and-cons-of-relational-database
https://tradeix.com/distributed-ledger-technology/
https://blockgeeks.com/guides/what-is-blockchain-technology/
https://www.guru99.com/testing-methodology.html
https://www.seguetech.com/waterfall-vs-agile-methodology/
https://airbrake.io/blog/sdlc/waterfall-model
https://linchpinseo.com/the-agile-method/
https://project-management.com/what-is-stakeholder-analysis/
https://www.mindtools.com/pages/article/newPPM_07.htm
https://www.essentialsql.com/what-is-a-relational-database-view/
https://help.anylogic.com/index.jsp?topic=%2Fcom.anylogic.help%2Fhtml%2Fconnectivity%2FView.html
http://ecomputernotes.com/fundamental/what-is-a-database/what-is-a-database-view
https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf
https://www.c-sharpcorner.com/UploadFile/rkartikcsharp/abc-of-wcf/
https://www.c-sharpcorner.com/UploadFile/dhananjaycoder/abc-of-an-endpoint-in-wcf/
https://netmarketshare.com/browser-market-share.aspx?options=%7B%22filter%22%3A%7B%22%24and%22%3A%5B%7B%22deviceType%22%3A%7B%22%24in%22%3A%5B%22Desktop%2Flaptop%22%5D%7D%7D%5D%7D%2C%22dateLabel%22%3A%22Trend%22%2C%22attributes%22%3A%22share%22%2C%22group%22%3A%22browser%22%2C%22sort%22%3A%7B%22share%22%3A-1%7D%2C%22id%22%3A%22browsersDesktop%22%2C%22dateInterval%22%3A%22Monthly%22%2C%22dateStart%22%3A%222018-05%22%2C%22dateEnd%22%3A%222019-04%22%2C%22segments%22%3A%22-1000%22%7D
https://github.com/RaidenRabit/Phaethon/blob/development/Report.docx

8. Appendix

User stories table

ID	As	I want to	So that
1.	GTL	have a catalogue of books available online	members interested can see books by author, title, and subject area
2.	GTL	Be able to request books from the other tech libraries	If book is missing it could be received from other libraries
3.	GTL	Be able to lend books to other tech libraries	If another library is missing a book it could be lent
4.	GTL	Be able to see members of other tech libraries	GTL could see if user is registered in another library
5.	GTL	Expose members for other libraries to see	Other libraries could see if user is already registered
6.	GTL	Be able to see other tech library catalogue of books and their status	GTL would be known if book is available
7.	GTL	Expose catalogue of books and their status to other libraries	Other libraries would know if book is available
8.	GTL	expose statistics based on all libraries	It is possible to see information of interest
9.	librarian	know how many copies of each book is in the library or out on loan	Librarian can make informed decisions when buying new books or lending them
10.	librarian	Add book to catalogue	Book is possible to be lent
11.	librarian	Remove book from catalogue	Book is no longer possible to be lent
12.	librarian	have a list of some books that I am interested in acquiring	They are easy to access and keep track of
13.	librarian	have a system that keeps track of books that cannot be lent	They are easy to access and keep track of
14.	librarian	Approve applicant	Card can be issued
15.	member	have card renewal notices sent	There is enough time to renew the library card
16.	member	check out books	Book can be read
17.	member	return book	Other interested users can use it

18.	applicant	become a member of the library	library card can be issued
19.	professor	have professors considered as a member of library automatically	library card can be issued
20.	reference librarians	access description of a book	Member could read it

Figure 46

Loan test case and scenario tables

Identifier	17
Description	Member wants to checkout a book
Precondition	Book must have a valid Copy ID, from date (date of checkout), member must have a number of books borrowed lower than 5 (but not null) and a valid SSN
Estimation	05:00:00

Figure 47 (Loan scenario)

Scenario: Checkout - 17	Valid test cases				
	Copy ID	SSN	From date	Number of books borrowed	Expected result
1.					invalid
2.		12345678	12-12-12	0	invalid
3.		1233456789	12-12	1	invalid
4.		123456789	12-12-12	4	invalid
5.		123456789	12-12-12	5	invalid
6.		123456789	12-12-12	6	invalid
7.	1	12345678	12-12	4	invalid
8.	1	1233456789	12-12-12	5	invalid
9.	1	123456789	12-12-12	6	invalid
10.	1	123456789	12-12-12		invalid
11.	1	123456789		0	invalid
12.	1		12-12-12	1	invalid
13.	1	123456789	12-12-12		invalid
14.	1	123456789	12-12-12	0	valid
15.	1	123456789		1	invalid
16.	1	123456789	12-12-12	4	valid
17.	1		12-12	5	invalid
18.	1	12345678	12-12-12	6	invalid
19.	1	123456789		4	invalid
20.	1	123456789	12-12-12	5	invalid
21.	1	123456789	12-12	6	invalid
22.	1		12-12-12		invalid
23.	1	12345678	12-12-12	0	invalid
24.	1	1233456789	12-12-12	1	invalid
25.	1	123456789	12-12		invalid
26.	1	123456789	12-12-12	0	invalid
27.	1		12-12-12	1	invalid

28.	1	12345678	12-12-12	4	invalid
29.	1	1233456789		5	invalid
30.	1	123456789	12-12-12	6	invalid
31.	1	123456789	12-12-12	4	invalid
32.	1		12-12-12	5	invalid
33.	1	12345678		6	invalid
34.	1	1233456789	12-12-12		invalid
35.	1	123456789	12-12	0	invalid
36.	1	123456789	12-12-12	1	invalid

Figure 48 (Loan Cases)

Group contract

I Definitions:

PROJECT – Semester 1, Software Development Top-up, project. The official name of the project being: Georgia Tech Library system.

MEMBER – Refers to each individual that signed the contract.

Official means of communication: Slack group, Physical meetings, Provided email addresses.

II Contract Terms:

This Temporary Collaboration Contract states the terms and conditions that govern the contractual agreement between members of Group who agree to be bound by this Contract.

Whereas, The Group is engaged in the PROJECT; and

Whereas, The Group desires to employ and retain the services of MEMBER on a temporary basis according to the terms and conditions herein.

Now, therefore, in consideration of the mutual covenants and promises made by the parties hereto, The Group and MEMBER covenant agree as follows:

- TERM. The term of this Temporary Collaboration Contract shall commence on 01.11.2018 and continue until the completion of the PROJECT.
- TERMINATION. MEMBER agrees and acknowledges that, just as they have the right to terminate their collaboration with The Group at any time for any reason, The Group has the same right, and may terminate their collaboration with The Group at any time for any reason
- Group 7 shall employ MEMBER as Project Worker. MEMBER accepts collaboration with Group 7 on the terms and conditions set forth in this Temporary Collaboration Contract and agrees to devote their full time and attention to the performance of their duties under this Agreement, as

stated on Exhibit B attached hereto. In general, MEMBER shall perform all the duties as described on Exhibit A attached hereto.

- RETURN OF PROPERTY. Within Seven (7) days of the termination of this Temporary Collaboration Contract, whether by expiration or otherwise, MEMBER agrees to return to The Group, all products, samples, or models, and all documents, retaining no copies or notes, relating to the Company's business including, but not limited to, UML Models, ID numbers, Code lines, Contracts, in any form or measure, obtained by MEMBER during its representation of The Group.

- GROUP'S PROCEDURES. MEMBER agrees and acknowledges that he or she shall comply with The Group's established disciplinary code as well as any other rules, policies, and procedures that may be introduced from time to time. Copies of such documents are available upon request.

- NO MODIFICATION UNLESS IN WRITING. No modification of this Agreement shall be valid unless in writing and agreed upon by all parties.

- WORKING CONDITIONS. All the employees of The Group must strictly follow the code stated on Exhibit C, attached hereto. Any violations of the code will immediately lead to the consequences stated on Exhibit B attached hereto.

- APPLICABLE LAW. This Temporary Collaboration Contract and the interpretation of its terms shall be governed by and construed in accordance with the laws of the Danish State and subject to the exclusive jurisdiction of the federal and state courts located in Denmark, unless specified in the contract.

IN WITNESS WHEREOF each of the parties has executed this Temporary Collaboration Contract, both parties by its duly authorized officer, as of the day and year set forth below.

Exhibit A – Duties

In general, the duties of the position to be filled by the employee shall encompass the following:

[see trello board]

Exhibit B – Working Performance

In general, the employee shall fulfill his duties, as stated on Exhibit A attached herein. In case one of the employee is not able to fulfill his duties, he must announce at least one of the other employees of The Group which can be seen on The Group participants, attached hereto, in case the employer who is not capable of fulfilling his job and does not announce at least one of the other employees of The Group, or has no valid reason for doing so, the other employers of The Group get to decide which consequences he must suffer, decision made by voting. For a vote to pass it requires at least 51% of the votes to be positive. In case on or more of the employers are not present for the voting, they must contact The Group and clearly state their vote, otherwise it will be considered that they voted for immediate termination of the contract. Each employer has the write to propose one consequence. Immediate termination of the contract is always an available consequence that must be voted over.

Exhibit C – Working Code

Working place and hours are decided on a weekly basis, unless otherwise established through vote with a positive percentage of at least 51%, through **official means of communication**. Clothing and Language are to be kept under normal social rules. Any aggression shown towards another worker is strictly forbidden.

- Participate to the daily stand-up meetings, held in Slack and conducted by the AliceBot;
- Participate to the Refinement and Retrospective meetings, regularly held;
- In order to keep a level of traceability, please always work in new branches, based on the Development branch, with the same name as the task you're currently working on (including the trello card number) ex: #CardNumber-card-title;
- Commits should be named as follows: #CardNumber commit message;
- When you believe a task is finished, create a pull request (always write a short description of what you did when creating the pull request), DO NOT MERGE WITH Dev or Master;
- Once the branch is approved (passes appveyour checks and gets a positive review), the owner of said branch, will merge with development branch;
- Merging with master will be done at the end of each sprint or when enough tasks are done;
- Master Branch is the one we will send teachers to review, it is to be considered as ready to go branch;
- Work from anywhere at any time, but finish your tasks on time/as fast as possible, and attend scheduled meetings;

The Group Participants

• **Ralf Zangis**

-Date of birth: 08.03.1997

-Contact: 1062012@ucn.dk

• **Andrei Eugen Birta**

-Date of birth: 11.02.1998

-Contact: 1062021@ucn.dk

• **Ádám Blázsek**

-Date of birth: 25.03.1997

-Contact: 1062084@ucn.dk

[i] According to research done by: Ádám Blázsek: https://github.com/AdamBlazsek/Documents/blob/master/A%20comparison%20of%20Proof%20of%20Work%20and%20Proof%20of%20Stake%20consensus%20mechanism%20-%20Adam%20Blazsek.pdf

[ii] According to research done by: Ralfs Zangis: https://github.com/bubriks/Review-of-NoSQL-and-SQL/blob/master/Review%20of%20NoSQL%20and%20SQL.pdf