T-404-LOKA, Lokaverkefni

Reportinator

Emil Hjaltason
Magnús Þorsteinsson

12th May 2017

Instructor:
Gunnar Sigurðsson

Examiner:
Stefán Freyr Stefánsson

Contents

List of Figures

List of Tables

1 Introduction

Reportinator is a system designed for Kvika to automate report delivery to financial regulators in Iceland. In the aftermath of the financial crisis in 2008, financial regulators increased their supervision over the financial system in Iceland. They put in place new regulatory measures that resulted in a much higher workload for the companies that operate under their control. One form of this workload is preparing, making and turning in regular (monthly, quarterly or annually) reports to the regulators. The goal of this project is to create a straightforward and easy-to-use system that does this repetitive work automatically to save time and reduce the chance of human errors.

This document contains all information about the development of Reportinator and presents the work structure, functionality, implementation, design, and scrum related documentation. The work structure chapter explains how the team worked on the set task as well as time table and release plan for the duration of the project. The general functionality of the system and the implementation is presented in a few chapters, but the main ones are the technology stack, scenarios, and the product backlog. The design chapter goes through the process of making the appearance of the system as appealing as possible to the future users of the system. Scrum related documentation gives a detailed description of the progress of the project at various pre-scheduled intervals.

1.1 The problem

Every year Kvika's employees have to turn in over 600 reports in various formats to financial regulators, with the majority of them created either manually or semi-manually. These reports take an enormous amount of time to make especially since each report must be handed in for each individual subsidiary, there are often up to 34 different ones.

1.2 The Goal

As previously stated the goal of Reportinator is to create an easy-to-use system that automates hand-ins to financial regulators. The rewards for using Reportinator is saving time for employees that can be better spent on other projects and consequently save salary cost for Kvika. Another benefit of the system is that it can help reduce human error because of how the reports are prepared today. The last step in preparing a report is to manually copy & paste data between worksheets among other manual tweaks that can be error prone.

1.3 Kvika

Kvika is a specialized investment bank operating in Iceland. They offer a broad range of services for individual and institutional investors such as asset management, corporate finance, and capital markets. They established themselves as an asset management company in 1999 and have an established reputation in asset management and offer comprehensive solutions covering major asset classes, including fixed-income securities, equities, and real estate in both domestic and international markets. Kvika is also a direct member of the stock markets in Reykjavik, Copenhagen, Stockholm, Tallin, Riga, Vilnius and has direct access to all principal markets in Europe and the United States.

1.3.1 Departments within Kvika

Kvika operates under a regular company management hierarchy with departments responsible for each section of the business. Throughout the duration of the project, we worked with three departments within the company. They are the IT department, the risk management department and the financial department. The employees of these three departments are the primary users of the system.

1.3.2 Workstation

Kvika provided the team with an office within their headquarters, on the 7th floor of Borgartún 25. They also provided us with a computer and two monitors each. In Figure 1 here below one can see the workstation provided by Kvika.

Figure 1: Workstation at Kvika headquaters

2 About the system

2.1 General Description

The system is a platform for automatic creation and delivery of reports to financial regulators and is an addition to the intranet of Kvika. The system gathers data from Kvika's databases and during the creation of every report, all data is validated to make sure that the system is not sending away mismatching information.

One of the primary goals for the system is to be an overview for deliverables which have to handed in on a regular basis, implemented through the calendar- and report log view (see Chapter 9).

All of the reports that have to be handed in follow a specific template which is provided by the financial regulators.

Our initial plan included a solution for three report bundles, one for each government regulator SÍ, FME, RSK, but as we will cover in our risk analysis chapter that goal was somewhat ambitious in retrospect, since we only managed to complete one of them, the hand-in for SÍ is named "Sundurliðun Fjárfestinga" but will be refered to as "SFSI" hereafter.

Figure 2, here below, contains an explanation of what we call a report bundle. Lets assume that each report bundle in the left table is a SFSI report submitted monthly and that Kvika is operating 8 subsidiaries shown in the right table. We will have to create and deliver 8 individual SFSI reports each month, one for each subsidiary. These 8 reports are grouped together in a report bundle.

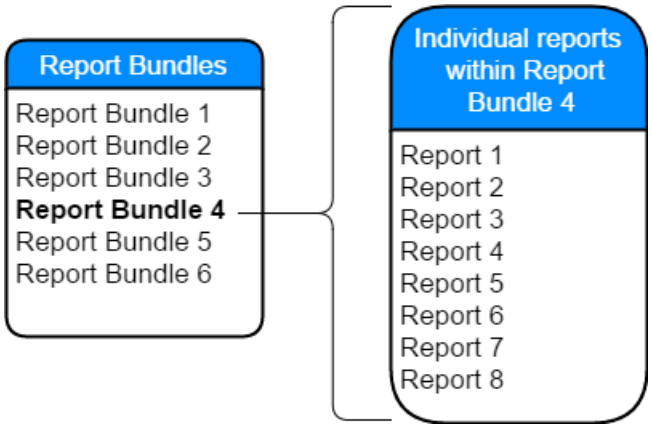


Figure 2: Report bundle breakdown

Whenever the system starts an automatic build of a report bundle the system gathers data and constructs an individual report for each subsidiary. When the system has collected the data and created the object to be handed in, it hands in the report via some service provided by the regulators and notifies the appropriate employee at Kvika, via email, that the build process has finished. The system also implements a logging system to see what reports are upcoming, what reports have been sent and also if anything went wrong with sending or creating the reports.

The flow chart in Figure 3 shows how the system should behave.

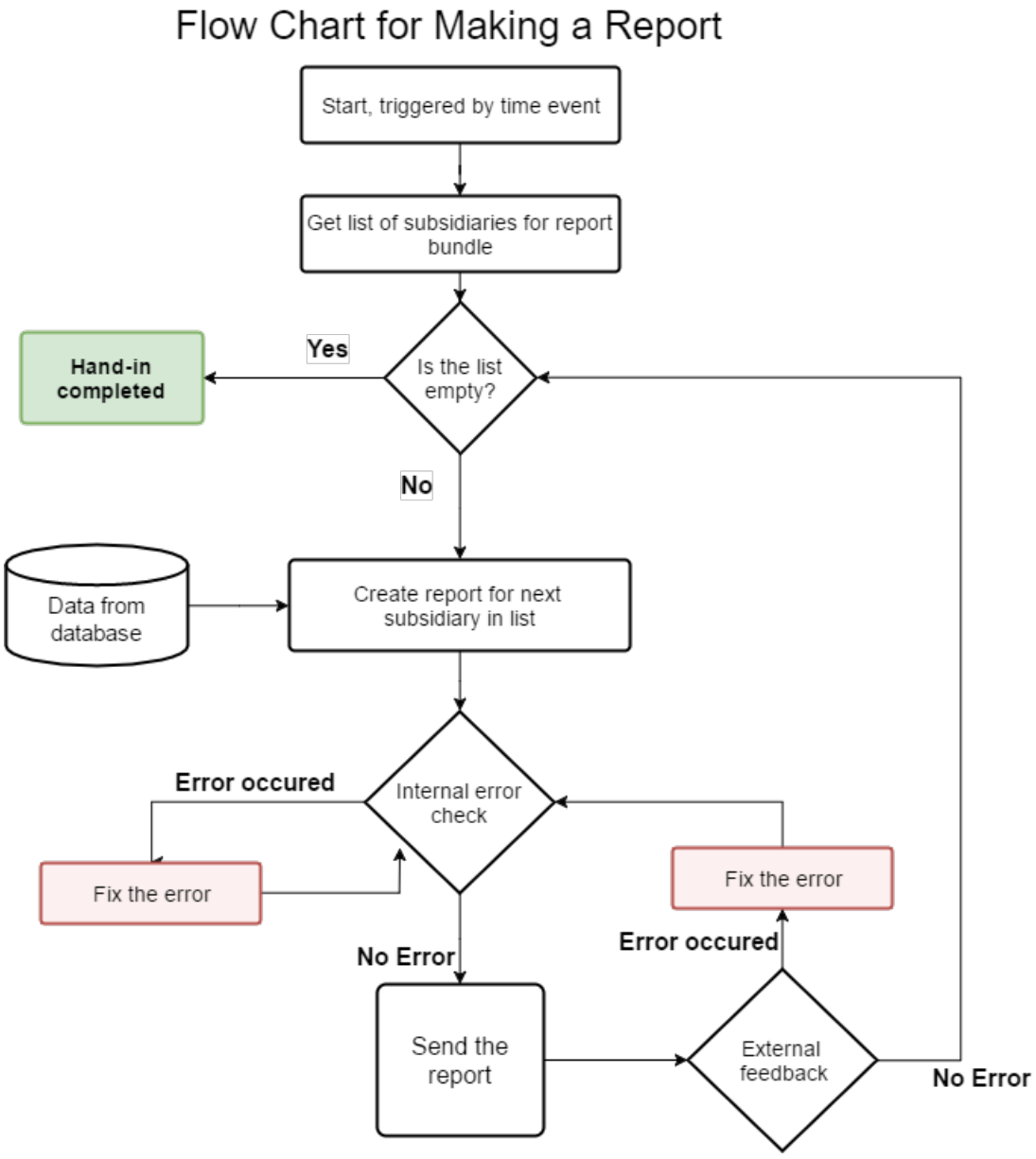


Figure 3: Reportinator flowchart

Here in Figure 4 you can see a simple system diagram.

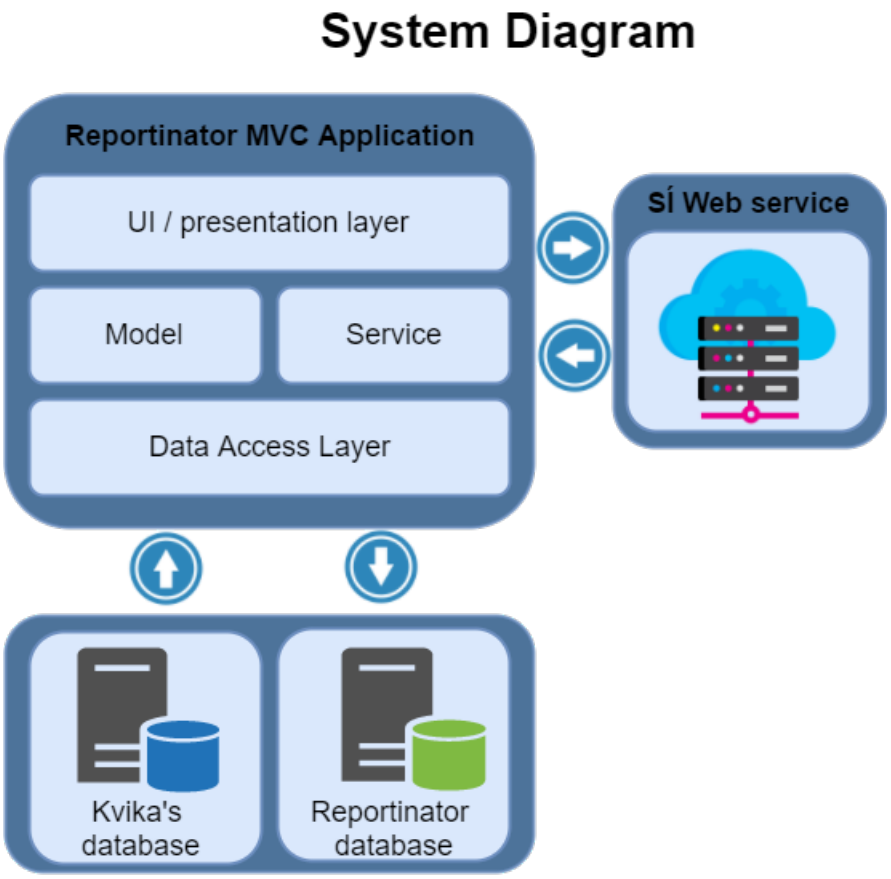


Figure 4: System Diagram flowchart

2.2 Competition Analysis

We did some research on if there are any solutions on the market that do the same or similar work. What we found out is that most of the companies that have to send in these reports create them manually with Excel spreadsheets or in some cases they have some automation where part of the report is automatically generated. We also found out that in some of the companies they have teams working on automating these reports.

3 Scenarios

The scenarios describe in a convenient way how human-system interaction takes place. They are mostly about how the system interacts and reports on errors that the system encounters. There are five different scenarios, and each one of them explains a different scenario which can arise.

1. Scenario: An employee in the risk management department gets an email from Reportinator informing him that an automatic build of a report has finished and passed all the internal checks, all work related to the report is finished. The employee goes on with his day doing what he's supposed to be doing rather than writing reports that should be automated.

2. Scenario: An employee in the financial department gets an email from Reportinator telling him that an automatic build of a report has finished but notifies him that there is an inconsistency in one of the internal checks in one report. The employee has to find out where the error is originating from so he can resolve it. He figures out that data is being marked wrong in navision and therefore data is saved incorrectly to the database. He asks IT to fix the error and the employee goes back to his desk. Then he uses Reportinator to manually build said report. He gets another email notifying him that the report was sent.

3. Scenario: An employee in the risk department at Kvika gets an email from Reportinator telling him that an automatic build of a report has finished successfully. An hour passes, and he gets another email which reads that the receiver has rejected the report due to inconsistencies, and furthermore the email reads which error it is. The employee goes to his coworker and tells him about the mistake, and they look through the data. Find out the inconsistency and fix it. Now the employee goes into the Reportinator page and presses the button to start the automatic build for the report again, and everything goes as in scenario 1.

4. Scenario: The administrator for the system gets an email telling him that the XML format at Seðlabanki Íslands (SI) has changed! He curses and begins to map out the changes. He has to perform major maintenance on the system within the source code. But when he finishes he's happy again to not have to deal with this for a few years (the rate the receivers change the templates).

4 Technology stack

In this chapter, we will list the development tools and programming languages used.

We will use Visual Studio 2015 as our integrated development environment and Microsoft SQL Server Management Studio. Our version control system is BitBucket.

Here below are the programming languages that we will be using.

1. C#
2. SQL
3. javascript
4. HTML
5. CSS

We will be using a number of development tools to assist with designing, documentation, and implementation of the project. Here below is a list of the tools we will be using:

1. Jira for documentation,
   (a) Tempo plugin,
2. Bitbucket for version control,
3. GoogleDocs, draw.io and LaTeX for documentation.
4. Wireframe for prototyping.

5 Project Schedule

The project plan is divided into a few sections, which take on a different perspective of the procedures needed to complete the set task on time. It begins with the work structure and time evaluation during different phases of the project. Following a description of a release- and milestone plan as well as a plan of deliverables. Last but not least comes a description of our take on the Scrum framework and the Agile methodology that the group will follow during the duration of the project.

All planning and decision making throughout the duration of the project was done in uniform agreement of both team members. If there were disagreements they were handled with patience and the problem debated until a solution was found which both team members agreed upon.

Due to the size and complexity of the project, specially when considering that the team members had limited understanding of the problem at hand at the beginning of the project as well as a limited experience at developing a fully functional system from scratch, the team decided to put effort into the planning of the project.

Right from the get-go it was decided to used Scrum and Agile methodologies, but not follow them blindly since the group only consists of two team members which is smaller than the suggested group size. The main alterations mostly involved documentation which had to be forfeited to focus on the programming of the project.

5.1 Work Structure

The work structure of the project is split up into three phases each one with their specific goals and deliverables.

1. Phase 1: Starts at the beginning of the project, 20th of December which is the day the team got a reply from Kvika accepting the cooperation for the project. The phase ends at 14th of February which is the point of the first project meeting (with the examiner). The deliverables are the requirement analysis, first iteration of the project backlog and design of the system. Also the work structure, detailed time evaluation and documentation for the first sprint.
2. Phase 2: Starts 15th of February and ends on the 24th of March. This period was chosen because 24th of march is the end of the first 12 weeks of the semester and this is where half of the work should have taken place (finished with 6 ECTS units of the 12 ECTS units which the full length of the project). The main deliverables are an updated work structure, documentation for the sprints during the period, updated design of the project and a full risk analysis.
3. Phase 3: Starts 25th of March and ends on 13th of May which is the date when we turn in the project. During this phase there will be two separate code freezes, first for the business

layer (all functionality should be within the system) and then a full code freeze. The main goals are get everything done so the project will be ready to turn in, both to the university and to Kvika. The deliverables are the final Project, user manual, developer manual, time schedule for the program and all scrum related documentation.

5.1.1 Phase 1: 20th of December - 14th of February

At the start of the project the team got to know Kvika, the workstation, the product owner and got familiar with future users of the system. As well as reading up on the tool-kit which was available to the team to develop the system.

The main goal was get the team members to figure out what the main functionalities which the system should be able to perform and how to best implement that functionality. This was achieved through multiple meetings with the product owner and future users of the system. Following each meeting the group performed a requirement analysis and iterated the design of the system, such as low-fidelity prototypes and intermediate prototypes. During these meetings the prototypes were presented to the future users and they asked to give their opinion on the design through a questionnaire. The requirement analysis consisted of gathering requirements, making user stories and task lists as well as drawing diagrams. This is also where the team agrees on coding rules and explains human-to-computer interaction, done through scenarios and use cases.

The last week of this phase was set out to be the first sprint, during this sprint the team should try to get an end-to-end flow through the system and getting the programming environment ready for both of the team members. The sprint is called "Sprint 0" and should be a trial sprint both to see if the precision of the time evaluation for each story point and to get a better understanding the complexity of the programming involved.

5.1.2 Phase 2: 15. February - 24. March

This is the phase where most of the programming is completed: data layer- , business layer- and presentation layer programming. The business layer has the highest priority since that is where the main functionality of the system lies. The phase is split up into 5 periods where each one corresponds to a sprint and is scheduled as follows:

1. Sprint 1: 15. February - 26. February: Programming report functionality.
2. Sprint 2: 26. February - 12. March: Programming system functionality.
3. Sprint 3: 13. March - 26. March: Programming system functionality.
4. Sprint 4: 27. March - 9. April: Programming web structure and system functionality.
5. Sprint 5: 10. April - 23. April: Programming web structure and system functionality.

These sprints all have specific set of goals that should be achieved to keep the plan set for the project. If there is an element which is not completed it is moved to the next sprint and finished when it is deemed appropriate.

Rough outline of the plan are to get the local databases for the system up and ready to use and program the model for the report to SI. When the model is ready there is to be programmed a XML generator and a XML parser to create XML strings and creating the model from the already generated XML string. Following a creation and programming of all the views and to finish the navigation through the system. Following all of this there are the problems of creating a connection to the API at "Seðlabankinn", connection to Kvikas databases, programming the data layer to populate the model correctly. Then there is the task of programming the services needed to complete all the requirements, outlined in the designing phase, for the system, as well as servicing all the models and controllers. For further breakdown of the work structure in each sprint see the Sprint Documentation chapter.

5.1.3 Phase 3: 25. March - 13. May

During this phase the group can wholly focus on the project, with no other classes to interfere with the work. This is the phase where the project should be well on the way to being finished and here is where the group will implement a business layer code freeze after week one and a full on code freeze after week two. Thus this phase is split up into 3 different sections:

1. Week 1: Sprint 6: Working on the business layer, finishing up the services needed to fulfill all requirements
2. Week 2: Get everything ready to turn in the project as well as finishing anything that is left from week 1 of the phase.
3. Week 3: Get the code ready to turn in as well as all documentation related the project.

During this phase the system should look like a full finished product and the main focus is to get everything in order that has not been completed in a very prioritized order to get the main features up and running before turning the project in.

The first two weeks are set up as sprints, they will be organized differently from the previous sprints since they are only one week long and serve as a backup, if something does not go as planned. For further information on the schedule in the sprints see Sprint Documentation chapter.

5.2 Time schedule

The schedule for the project follows the lines of the work structure and will be split up into the same three phases. A few points to be clarified, there is an exam period between 3rd of April and 21st of April, thus the group members will be reviewing for the exams some of the time during that period. During the exam period there is the Easter holiday, which will be used to work on

the project. The exact dates for the exams will be clarified in the Sprint documentation Sprint Documentation chapter. The schedule for meeting with the team's instructor was set to be on Tuesdays between 17:00 to 18:00. The work schedule for each of the three phases is as follows:

Phase 1 and 2: The team agreed with the approval of the product owner to work on Mondays and Fridays at Kvika's headquarters between 9:00 and 17:00. This is the basic structure for the first two phases but the team will adjust the structure to appropriately fit in the schedule of other classes and workload that might occur during the period. Here is a summary of the hours put into the project:

1. Phase 1: Time spent at Kvika's headquarters 180 hours (10 days) and other workload 24 hours split between the team members. This equals 204 hours spent mainly scheduling and planning ahead for the following two phases as well as writing reports. During the last week there is the first sprint which will be two full days or 36 hours
2. Phase 2: Time spent at Kvika's headquarters 300 hours (20 days) and other workload 90 hours split between the team members. This equals 390 hours spent mainly on programming the full functionality of the system. The last two sprints are during the exam weeks thus the workload there is half of the workload of the first three sprints.

Phase 3: During this sprint the team members will be working at Kvika's headquarters every weekday from 9:00 to 18:00 and from 12:00 to 18:00 on Saturdays. This equals to 112 hours each week split between the team members.

Summary: The total estimated workload for both team members equals to 930 hours over all three phases. During phase two and the first two weeks of phase three the team estimates that 30% of the time will be spent on other duties (such as writing documentation, re-designing to meet new requirements or hurdles etc.) thus the total time contributed to programming is 430 hours. The first estimate of how long it will take to finish the entirety of the product backlog is 500 hours, with each story point taking approximately 2.5 hours, this estimation is re-evaluated after each sprint.

Full documentation on the workload during each sprint can be found in the Sprint Documentation chapter.

5.3 Release-, Milestone and Deliverable Schedule

The schedule for releases, milestones and deliverables follows the three phases. The releases are set out to get reviews from the product owner and the future users of the program, milestones to get the sense of achievement when finishing a large part of the system and finishing the deliverables both to the university and Kvika. The plan is as follows:

1. 14th of February. Release: End-to-End flow through the system.
2. 14th of February. Milestone: Finishing off the requirement analysis, project design, product backlog and a rough schedule for each of the sprints.

3. 14th of February. Deliverables: Turning in the first project report, schedule for the project and the design of the system.
4. 13th of March. Deliverables: Turning in the second project report, documentation for sprints 0 through 2.
5. 24th of March. Release and milestone: Finishing off the business- and data layer programming.
6. 24th of March. Deliverables: Documentation for sprints 0 through 5.
7. 7th of April. Release and milestone: Finished with presentation layer programming.
8. 1st of May. Release: Code freeze on all business and data layer programming.
9. 8th of May. Release: Code freeze on the whole project.
10. 13th of May. Deliverables: The final system, Documentation for all sprints, user manual, developer manual and final project report.

5.4 Scrum

The team decided to use the Scrum framework and Agile methodologies for the project to help team members perform at their highest level. The product owner is Finnur Þór Erlingsson, head of IT at Kvika, and the scrum master is Emil Hjaltason. The team will program in a set sub-periods called sprints where the product backlog is split into sprint backlogs. At the beginning of each day the team will take a few minutes for a daily sprint meeting and when each sprint is finished the team will schedule two meetings, a sprint review meeting and a sprint retrospective meeting. Each meeting is summarized to display in this report in the Sprint Documentation chapter, but the daily sprint meetings will be summarized further to not get far into the details.

One problem with choosing to use the Scrum framework is the team size, but the team only consists of two programmers which is too small for a Scrum team. Thus it was agreed upon to let loose on some of the requirements of using the set framework. This is mostly concerning the documentation surrounding the product and sprint backlogs, the backlogs are to be updated as often and as thoroughly as possible but not to get stuck on the details. It was agreed upon that the programming was more important and should be prioritized above documentation, specially after finishing the time schedule and realizing the workload ahead.

5.4.1 Sprint Backlog, burnup- and burndown charts

The sprint backlog is to be created from the product backlog before each sprint, preferably the weekend before, and should only consist of as many story points as the hourly estimate per story point allows, thus scheduling the next sprint with close regards to the time schedule set for the whole duration of the project. The backlog should be updated as often as possible but not to spend to much time doing so.

Burnup and burndown charts are to be created from the sprint backlog with regards to the time tracker spreadsheet and Tempo Jira package. The charts are to be made of all of the stories within the backlog, whether or not they are completed.

5.4.2 Daily Scrum

Daily Scrum meetings should be held at the start of every working day within a sprint. The summary should contain the days where the meetings were held to better clarify which days the team was working within the sprint. The following questions are to be addressed during every meeting:

1. What did you do yesterday (last time)?
2. What will you do today?
3. Are there any impediments in your way?

Every meeting is to be documented on a shared Google Drive but in the Scrum documentation chapter there will be a summary of the daily meetings, specially concerning the impediments facing a given task.

5.4.3 Sprint Review

When a sprint finishes, there is to be scheduled a meeting to go through and consider the newly finished sprint. The main goals are to consider what could have been done better and to summarize the impediments to get rid of them for the following sprint. This is also the primary way to get the product owner's take on the project and to get his vision on how the team members are tackling the programming and problems. During the meeting there should be done a summary of what was accomplished during the sprint.

5.4.4 Spring Retrospective

This meeting is also scheduled right after a sprint finishes. The team will do a retrospective to modify anything that should be modified, specially considering the sprint backlog and to see if the project is behind or on schedule. The key questions that are to be answered during this meeting are:

1. What went well during the sprint cycle?
2. What went wrong during the sprint cycle?
3. What could we do differently to improve?

6 Risk Analysis

We assessed what could go wrong while developing and working on the project. The resulting assessment can be seen in table 1 here below. We gave the events that we thought were plausible two values on the scale from 1 to 5, a chance and effect and the product of the aforementioned values is the risk factor that we use to evaluate the impact of the risk event.

Issue	Chance	Effect	Consequence	Solution	Risk factor
Lack of knowledge of the team	4/5	4/5	Might reduce sprint capacity and/or delay the initial release of the project	The team will have to study up on the lacking knowledge	16
We might not receive access to Kvika's data mart in time	3/5	4/5	Might have to spend extra time to create another data schema	Use Kvika's old stored procedures to gather data	12
Team members might get sick	2/5	5/5	Sprints will probably fall behind on schedule	Reschedule sprints	10
Risk of Product owner/stakeholders changing requirements	3/5	3/5	We might have to change parts or even all of the project	Reschedule sprints with the new requirements and update deadlines	9
Risk of regulators changing templates	1/5	5/5	Regulators might not accept reports anymore from our system	System needs maintenance	5
Risk of Kvika's web servers being down	1/5	5/5	Not being able to communicate with e.g. databases or CI client	Work on stories that do not require db connection	5
Risk of regulators web service being down	1/5	5/5	We will not be able to hand in reports	Contact the regulators directly and find a solution together	5

Table 1: Risk Analysis

6.1 Post Mortem & Risk Analysis Retrospect

This was the largest project that either of us has worked on therefore we assessed the lack of experience of the team as the greatest risk factor. That turned out to be one of our setbacks since neither of us had implemented much of the features before. In turn, we spent a lot time reading up on material that was essential for us to continue when we hit a wall in the development. During sprint 3 it became clear that we underestimated the hours per story point so we changed the time per story point from 2,5 hours to 3 hours and during sprint 5 we again changed the time per story point to 4 hours.

In the middle of sprint 1 another issue came up. During a meeting with representatives from each department of future users we realized that some of the requirements from each department overlapped with one another but were not compatible so we had to re-design most of our back-end design and "go back to the drawing board". The requirement that was the most complicated and time consuming to fulfill was that the risk department wanted a dynamic SQL editor so they could change the data in the reports within the Reportinator web application. One month later there was another requirement change, the product owner decided to throw away the idea of dynamic SQL so we had to revert back to our old design again. So in retrospect we underestimated both the chances of a requirement change and the effect of one because of the time we spent on adapting to changes from the stakeholders. Kvika is working on creating a data mart for easy data access for various every day tasks. Our initial design assumed that Reportinator was going to use it but there were some delays on the data mart and it will not be ready before our deadline. This event was realized just a few weeks before project meeting 3 so we had very little time to change our data schema and environment. Three of our top four risk events occurred, although we hadn't planned precisely what we would do in case any of them occurred, the risk analysis still helped us realize the gravity when things do not go according to plan and at least give us some idea of what would be required of us in order to compensate for that event. We managed to complete 53 requirements out of 67. Out of the 53 that we managed to complete 37 were A requirements. As you can see in the table below we still had 10 A requirements in our product backlog. The reason why we did not finish more A requirements before starting on the B ones is that we wanted to create a functional system for one report before adding others to the system. In retrospect adding more deliverables to the system should have been categorized as B requirements. The time we spent on the project total 1.196 hours, 649 hours spent on programming and 547 hours spent on designing and other miscellaneous activities.

Requirement Summary

	A requirements	B requirements	C requirements	Total
Completed	37	12	4	53
Remaining	10	3	1	14
Total	47	15	5	67

7 User Groups

7.1 About User Groups

User Group	Background	Usage of system	Enviroment	Main goals
Administators **Importance:** High	**Age:** 18-70 **Gender:** All **Education:** Some experience/education in computer science **Abilities or Disabilities:** Nothing special **Computer skills:** Very good	**Usage:** All year around, usually when a new user starts using the system **Training:** Some training required **Attitute:** Positive **No. of Users:** 1-2	**Technical Enviroment:** Personal computer or a laptop, good internet connection **General Enviroment:** At work **Other Environment:** Nothing special	Create/remove users and modify reports, define a role for the users and general maintenance

Table 2: Admin User Group

User Group	Background	Usage of system	Enviroment	Main goals
Regular user **Importance:** High	**Age:** 18-70 **Gender:** All **Education:** No education required **Abilities or Disabilities:** Nothing special **Computer skills:** Very good	**Usage:** All year around **Training:** Some training required **Attitute:** Positive **No. of Users:** 3-5	**Technical Enviroment:** Personal computer or a laptop, good internet connection **General Enviroment:** At work **Other Environment:** Nothing special	View report log and in some cases turn in reports manually, general maintenance

Table 3: Regular User

8 Product Backlog

The product backlog is a complete (as complete as possible) description of all functionality and processes within the system. The requirement list takes on what the system needs to be capable of doing to perform as the product owner and stakeholders want it to. The user stories take on every requirement and make it a story so they can be further processed. Each story is further broken down into a task list which can be given hour estimates. These estimates are to get the general idea on how much time the implementation of the design for the whole project will take. This estimate will be re-evaluated after each sprint when the team gets a better idea of their skill as programmers. As for use cases, they were made for processes that the team thought needed better explanation. They also take on human-system interaction, to further explain how the system is supposed to be used. The diagrams further explain the processes within the system, the general functionality and how the system operates.

8.1 Requirement Analysis

The requirement list is a prioritized features list, with each entry trying to be as self-explanatory as possible. The list is split up into non-functional and functional requirements. Both the lists are ordered in a prioritized order where A is the highest priority, then B and C comes last. If the requirements have a line over the text they have been excluded from the project, either they are not relevant to the solution or they have been integrated into other requirements, thus implementation as listed here below is unnecessary.

8.1.1 Non-functional Requirements

Here below we can see a table with non-functional requirements. Requirements are ordered by priority. We were not able to finish the requirements with strike through text.

Non-functional Requirements	Priority
The system should be implemented to automatically do actions without user interference.	A
~~The system should be designed with dependency injection in mind.~~	A
~~The system should be designed with TDD principles in mind.~~	A
The system should be well documented.	A
The system should be testable, unit tests, selenium tests, etc.	A
The system should work on all major web-browsers.	B
The system should be able to be used by anyone who speaks icelandic and/or english.	C
The system should be able to run on all operating systems.	C

Table 4: Non-functional requirements

8.1.2 Functional Requirements

Here below we can see tables with functional requirements. The requirement lists are ordered by priority. The first column is a description of the requirement and the second column is the priority of each requirement.

Functional Requirements	Priority
The system should be a part of Kvika's internal web page.	A
The system should have a calendar page.	A
The system should have a report log page.	A
The system should have a report bundle page.	A
The system should have a report detail page.	A
The system should be accessible from the navigation bar on Kvika internal web page.	A
Within the navigation bar there should be a shortcut to the calendar page.	A
Within the navigation bar there should be a shortcut to the report log page	A
The system should be able to hand in reports to Seðlabanki Íslands (SÍ).	A
~~The system should be able to hand in reports to Fjármálaeftirlit (FME).~~	A
The system should notify the admin of a report when it starts making the report.	A
The system should notify the admin of a report about its progress.	A
The system should notify the admin of a report when it is finished creating a report.	A
The system should be able to create reports to send in to the regulator.	A
The calendar page should be split in two, overview at the left side and the actual calendar at the right side.	A
The Calendar should list reports in ascending order by their due dates.	A
~~The reports on the calendar should be click-able, and go to their report detail or overview page when clicked.~~	A
The Report Log should list all reports.	A

Table 5: Requirement List Part 1.

Functional Requirements	Priority
~~The Report Log page should be filterable, asc and desc on each column.~~	A
The Report Log page should be searchable, including substrings.	A
The Report Log page should have a column for 1. Error, 2. Number, 3. Receiver, 4. Number in the bundle, 5. Name, 6. Due date, 7. No sent reports, 8. Status (Sent/Not sent/Not started/Error/Accepted), 9. Date sent	A
The Report Bundle should list all individual reports within the report package.	A
At the top of the page there should be information about the receiver and due date and name of the report package.	A
The Report Bundle list should have a column for 1. Error, 2. Number, 3. Name, 4. Status, 5. Feedback, 6. Date sent, 7. Sender.	A
The report details page should contain information about 1. Number, 2. Reveiver, 3. Name, 4. Status, 5. Feedback, 6. Date sent, 7. Sender, 8. Internal Errors. .	A
At the top of the page there should be a button to manually turn in this individual report.	A
The system should be able to turn in multiple reports (report bundle) in a short period of time to SÍ.	A
~~The system should be able to turn in multiple reports (report bundle) in a short period of time to FME.~~	A
The system should be able to turn in a single report at a time to the desired receiver.	A
The report bundle to SÍ is a single XML format for all the reports within the bundle.	A
~~The report bundle to FME is split up into 3 groups, Séreignarleiðir, Fjárfestingarsjóðir/Verðbréfasjóðir and Fagfjárfestasjóðir.~~	A
The system has to error check internally for each individual report.	A
When an internal error occurs the automatic sending of the report is disabled but can be turned on when error has been rectified (automatic sending sequence).	A

Table 6: Requirement List Part 2.

Functional Requirements	Priority
The system has to handle error checks externally for each individual report	A
The system should have an error specific to “report format not compatible”	A
~~The system should have a error specific to “Web service is down” for Kvika, SÍ, FME and RSK.~~	A
~~The system has to access the data from data warehouse/data mart at Kvika.~~	A
~~When data is input into the string created from the XML format, the system has to search for a specific keyword and place the data at that location.~~	A
The system has to be able to properly create a well formatted XML report.	A
The system has to gather data for further processing.	A
The system has to populate a model created from the standardized XML template issued by the regulators and create the XML string back from the model.	A
The system has have a database table for report logs	A
The system has have a database table for report bundles	A
The system has have a database table for XML strings for version control of report creation.	A
~~The system has have a table for internal errors~~	A
The system has have a table for external feedback	A

Table 7: Requirement List Part 3.

Functional Requirements	Priority
The system has to internally validate the data, to see if assets and liabilities match.	B
~~The system has to store XML formats in a database to easily counter changes from receivers.~~	B
At the top of the page there should be a button to access grid view to see the report in spreadsheet format.	B
The Report Bundle list should be scrollable.	B
The Report Bundle should list all reports within the bundle in ascending order.	B
The Report Bundle should be filterable, asc and desc on each column.	B
~~The Report Bundle should be searchable, including substrings.~~	B
At the top there should be a button to send the whole package in manually, through the system.	B
The system should have a grid view Page to open reports.	B
The Homepage of the system should be the Calendar page.	B
~~The system should be able to hand in reports to Ríkisskattsstjóri (RSK).~~	B
The overview should have an easy to use interface to travel between periods	B
The calendar should start at todays date.	B
The Calendar should have a scroll mechanism to see previous and future dates.	B
The Calendar should differentiate between reports that have been sent in and which ones have yet to be sent in.	B
The system has to keep track of when reports are sent in to see which reports are sent in on time and which are sent in overdue.	C
The system has to keep track of who sends in the reports.	C
The system has to keep track of statistics for each individual reports.	C
The Calendar should have a 3 month period which the user can scroll through.	C
~~The system should notify administrator of a report if the report is overdue to send in.~~	C

Table 8: Requirement List Part 4

8.2 User Stories & Task Lists

User stories are created from each and every one of the requirements, some stories might include two requirements or more and some requirements might have two or more stories, etc. At least every requirement is connected in one story or more. Each user story is prioritized in ABC order, and the list is also ordered so that the important stories are in the first table and so on. This also applies to B and C stories. Each user story is split up into tasks and then each task is given an hourly estimate, but it is only a rough estimate. Every time the team finishes a sprint, the hourly estimates will be re-evaluated in light of new information gathered during the sprint. This method was chosen to get a clearer scope of the size of the project, but we emphasize that it is a rough estimate.

User story table can be found in Appendix I.

8.3 Use Cases

Use cases were created to emphasize on certain processes and functions within the system, also to explain alternative flow when navigating through the system. They mostly cover requirements that have to do with clicking around the web page, such as showing navigation to certain functions which are accessible in certain parts of the site.

Use Case 1

Name:	User checks what reports are due soon
Number:	1
Priority:	High
Precondition:	User has to have access to the system
Description:	The user goes to the Reportinator tab within the internal network at Kvika, the first page is the calendar, which automatically displays todays date and the next 30 upcoming days.
Alternative flow:	The user could look at the report log which also displays upcoming reports.
Postcondition:	The user knows about the upcoming reports.
Actors:	Regular user

Use Case 2

Name:	User checks what reports are within a report bundle
Number:	2
Priority:	High
Precondition:	User has to have access to the system
Description:	The user goes to the Reportinator tab within the internal network at Kvika, the user can click on the report from the homepage, which is the calendar page. When he has found the correct report he clicks on it and on that page there is a list of all the report within the report bundle.
Alternative flow:	The user could click on the report bundles in the report log
Postcondition:	The user knows which reports are within the report bundle
Actors:	Regular User

Use Case 3

Name:	User checks out errors generated with internal checks
Number:	3
Priority:	High
Precondition:	User has to have access to the system and there is a internal error within a report
Description:	The user goes to the Reportinator tab within the internal network at Kvika, finds the correct report on the calendar view and clicks on it. There he chooses the report with the internal error and clicks on it. Displaying the report detail view where there is a window displaying all the errors generated when automatically building the report.
Alternative flow:	The user clicks on the link in the email he got to immediately go to the report detail page, or he could go through the report log instead of the calendar.
Postcondition:	The user knows what errors are within the report
Actors:	Regular User

Use Case 4

Name:	User checks out errors generated with external feedback
Number:	4
Priority:	High
Precondition:	User has to have access to the system and there is a error received from the receiver.
Description:	The user goes to the Reportinator tab within the internal network at Kvika, finds the correct report on the calendar view and clicks on it. There he chooses the report with the external error and clicks on it. Displaying the report detail view where there is a window displaying all the errors within the report.
Alternative flow:	The user clicks on the link in the email he got to immediately go to the report detail page, or he could go through the report log instead of the calendar.
Postcondition:	The user knows what errors are within the report so he can fix them
Actors:	Regular User

Use Case 5

Name:	User stops automatic build of a report bundle
Number:	5
Priority:	High
Precondition:	User has to have access to the system
Description:	The user goes to the Reportinator tab within the internal network at Kvika, finds the correct report on the calendar view and clicks on it. Now he's in the report bundle view and clicks on the button to stop the automated build.
Alternative flow:	The user could find the correct report bundle from the report log
Postcondition:	The system wont build the report automatically
Actors:	Regular User

Use Case 6

Name:	User stops automatic build of a report
Number:	6
Priority:	High
Precondition:	User has to have access to the system
Description:	The user goes to the Reportinator tab within the internal network at Kvika, finds the correct report on the calendar view and clicks on it. Within the report bundle view he chooses the report he want to stop the automatic build and clicks on it. Now he presses the stop button to stop the build.
Alternative flow:	The user could find the correct report bundle from the report log
Postcondition:	The report wont build automatically
Actors:	Regular User

Use Case 7

Name:	User starts automatic build process again
Number:	7
Priority:	High
Precondition:	User has to have access to the system and the automatic build has been stopped
Description:	The user goes to the Reportinator tab within the internal network at Kvika, finds the correct report bundle on the calendar view and clicks on it. Within the report bundle view he chooses the report he want to start the automatic build and clicks on it. Now he presses the button to start the automatic build again.
Alternative flow:	The user could find the correct report bundle from the report log
Postcondition:	The build is now automatic
Actors:	Regular User

Use Case 8

Name:	User views a report in spreadsheet view
Number:	8
Priority:	High
Precondition:	User has to have access to the system and the report has been generated
Description:	The user goes to the Reportinator tab within the internal network at Kvika, finds the correct report bundle on the calendar view and clicks on it. Within the report bundle view he chooses the correct report and clicks on it. Now he clicks the button to ender spreadsheet view.
Alternative flow:	The user could find the correct report bundle from the report log
Postcondition:	The user can look at the report in spreadsheet view
Actors:	Regular User

8.4 Class Diagram

The SFSI Class Diagram can be found in appendix III.
Class Diagram for Sundurliður Fjárfesting Seðlabanki Íslands

(Figure 46) is a class diagram for one report (Sundurliðun Fjárfestinga til Seðlabanka Íslands). The main goal of the figure is to show the connections between each class. The report is ordered as follows:

1. Gögn
    (a) Eignir
        i. AdrarEignir
        ii. AfleidusamningarEignir
        iii. HlutabrefOgHlutdeildarskirteini
        iv. Innlan
        v. SkuldabrefOgVixlar
    (b) Skuldir
        i. AdrarSkuldir
        ii. AfleidusamningarSkuldir
        iii. BeinarLantokur
        iv. Skortstodur
        v. Verdbrefautgafa
    (c) EigendurHlutdeildarskirteinaISjodinum

These are classes in the first three lines of (Figure 46). The remaining classes are helper classes to fill in fields of the main chapters of the report which are ordered here above. (Figure 5) and **(Figure 6** explain the ordering here above with more precision and show the "Eignir" and "Skuldir" part of the data model.
The MasterTemp and Template classes are used multiple times throughout the model and we figured that this was the most efficient way to build the model. Most of the lowest tear classes have the same number of variables and each variable is List<int>. Thus we could reuse multiple classes and named them Template classes.
For further explanation of the models used in this solution see the user manual.

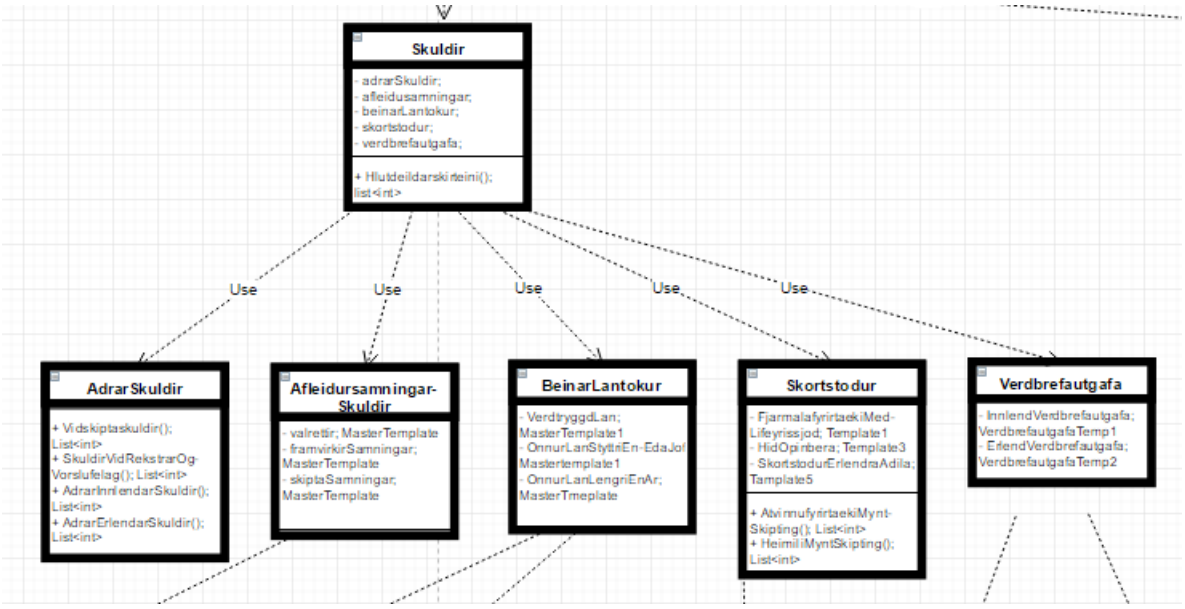


Figure 5: One part of SFSI class diagram. (Skuldir)

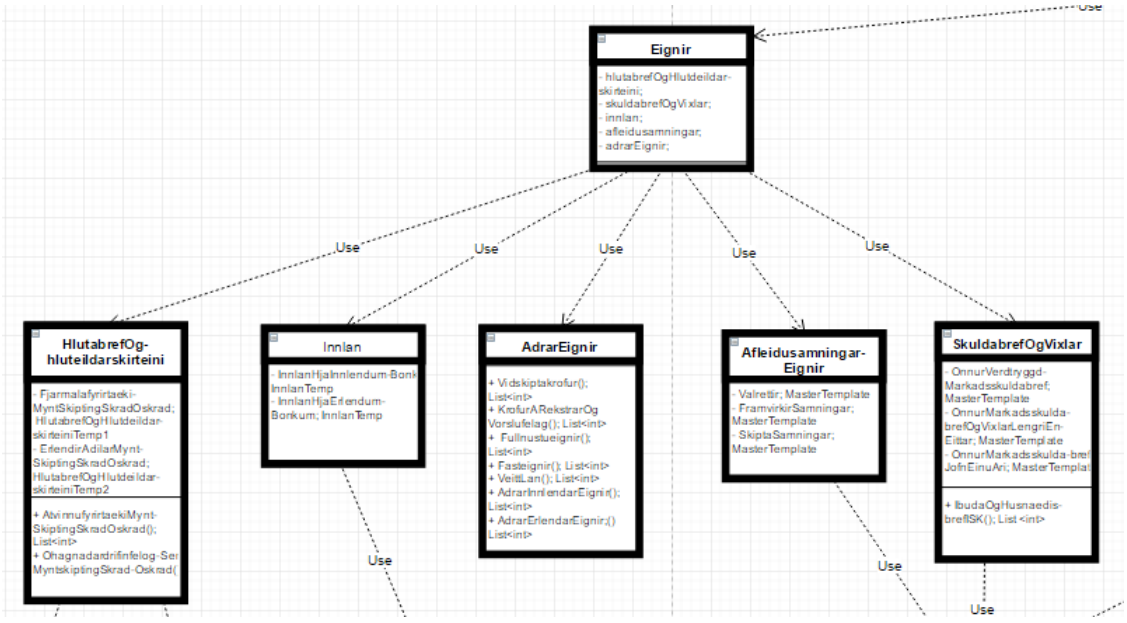


Figure 6: One part of SFSI class diagram. (Eignir

8.5 Table Schema

This table schema describes how the reports will be kept in databases. As shown in (Figure 7).

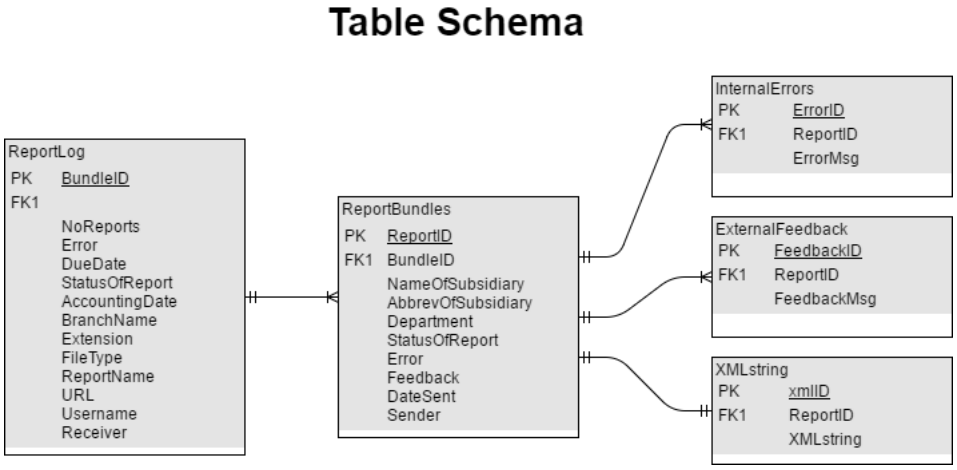


Figure 7: Table Schema

8.6 Navigation Diagram

The navigation diagram shows the flow within the system (Figure 8). The Homepage is both a calendar view and a report log and is in the dropdown menu so it can be accessed at all times. When the user clicks on a report in the report log or calendar, he goes to the report bundle view where he can access the report details page when clicking on a report. When the user is in the report details page, he has the option to view the report in grid view if the report has been built already.

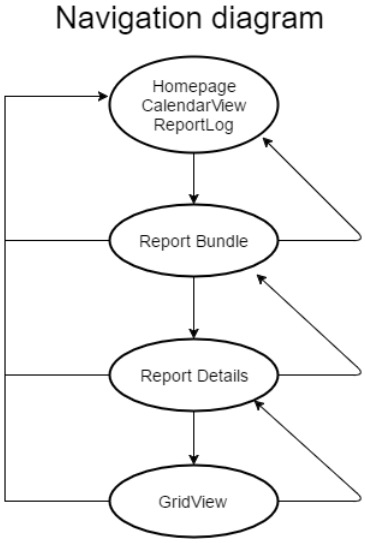


Figure 8: Navigation Diagram

8.7 Coding Rules

All coding should be in written English.

8.7.1 C# and JavaScript

All Variables/Parameters and functions should be written in Camel casing (e.g. inputString, outputInt, getDatatype, calculateTax). Classes and Structs should be written in Pascal casing (e.g. HelperClass, SendReport). Unless it is reasoned and improves the readability of the programming.

All names within the program should be descriptive of their use.

Comments should be written above each Class/Struct and above each Function. If there is any doubt that the code will not explain itself within a function, there should be a comment above that particular code (not inline comment!).

8.7.2 HTML and CSS

All HTML code will be XHTML compliant. For example, there will be a correct DOCTYPE at the start of every document and tags such as <html>, <head>, <title> and <body>.

All tags will be nested appropriately and closed (trailing slash if not needed).

All tags and attributes should be written in lowercase and quoted if needed, including id and class attributes (<table class=”striped”> </table>).

Spacing should be after each after each tag for each attribute, with no space before or after the equal sign (=). Example, (<link rel=”stylesheet” href=”style.css”>)

Do not add blank lines without reason, only if the lines if they enhance readability, for example after large separate code blocks and after main tags such as <html>, <head>, <body> and <footer>.

All CSS should be written in lowercase with meaningful class and id names.

Comments in HTML will be written in a separate line above the tag which is explained in the comment. This is the style that should be used <!- ->, comment only where it is necessary.

Comments in CSS will be above the selector which is explained, and the styling should be /* */.

8.7.3 Brackets and Spacing

Opening brackets of function, classes etc. should be in separate lines, example:

```
Void printFunction()
{
   // do something
}
```

Whitespace should be after keywords (e.g. if, while, for, etc.) and on both sides of the operator (e.g. =, +, -, ==, ||, &&).

Indentation should be one tab for each layer, which applies for all programming languages in use.

Empty lines should be between property declarations and function definitions and where the empty line improves readability.

8.7.4 Examples

```
<!DOCTYPE html>
<html>
  <title>Page Title</title>

  <body>
    <h1>This is a heading</h1>


    <h2>Tokyo</h2>
    <p>Tokyo is the capital of Japan, the center of the Greater Tokyo Area,
    and the most populous metropolitan area in the world.
    It is the seat of the Japanese government and the Imperial Palace,
    and the home of the Japanese Imperial Family.</p>

   <ol>
     <li>London</li>
     <li>Paris</li>
     <li>Tokyo</li>
   </ol>

  <table>
    <tr>
      <th>Name</th>
      <th>Description</th>
    </tr>
    <tr>
      <td>A</td>
```

```
      <td>Description of A</td>
    </tr>
    <tr>
      <td>B</td>
      <td>Description of B</td>
    </tr>
  </table>


   </body>
</html>


p {
   margin-top: 100px;
   margin-bottom: 100px;
   margin-right: 150px;
   margin-left: 80px;
}

div.container {
   border: 1px solid red;
   margin-left: 100px;
}

.floating-box {
   float: left;
   width: 150px;
   height: 75px;
   margin: 10px;
   border: 3px solid #73AD21;
}

.tooltip .tooltiptext {
   top: -5px;
   right: 105%;
}
```

9 Design

9.1 Lo-fi Prototypes

The lo-fi prototypes are a sketch up of the final product; they are done to get a better visualization of the navigation through the system. The sketches were done in iterations, starting off there were seven views, but between the first and the second iteration, the group decided to drop a list view which was not needed. Between the second and the third iteration, the group dropped a view which was the turn in view. It was concluded that the system should be fully automatic where the user can only turn on and off the automatic builds. The procedure will be fully automatic and should only store error related database for a certain period.

The final conclusion was that the system should contain five views, listed here:

1. Calendar view - Also the Homepage
2. Report log view - A table of reports bundles
3. Report bundle view - A table of reports
4. Report details view - Information about the report + errors
5. Spreadsheet view - To view the XML formatted report

The five views are shown here in figures 9 through 13, The lo-fi prototype was created in wireframe.cc, which specializes in matters like these.

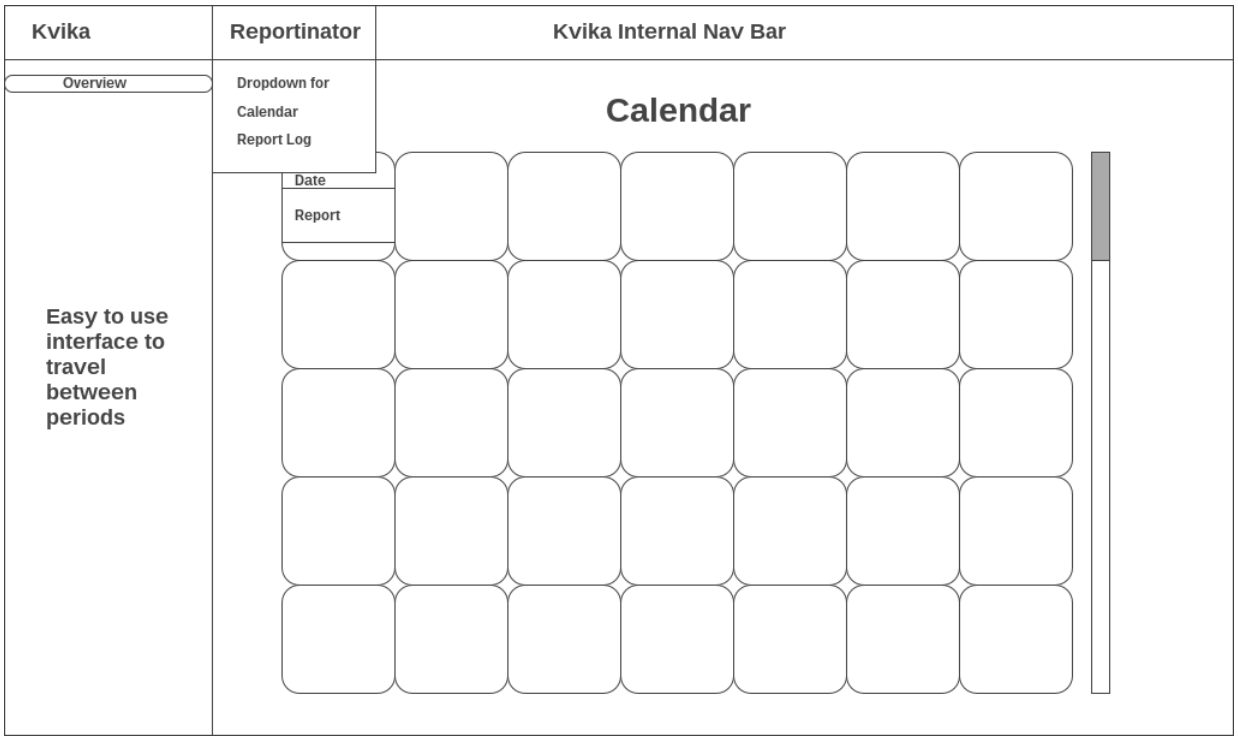


Figure 9: Lo-fi Prototype: Calendar View

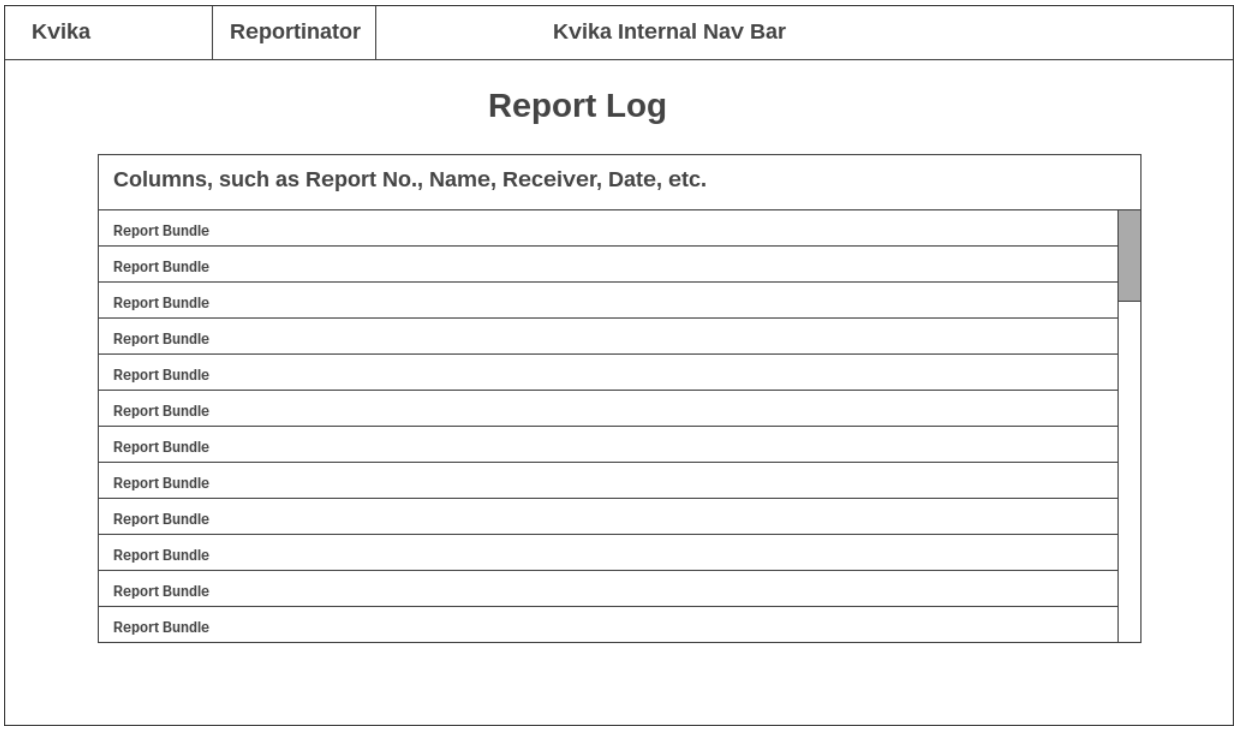


Figure 10: Lo-fi Prototype: Report Log View

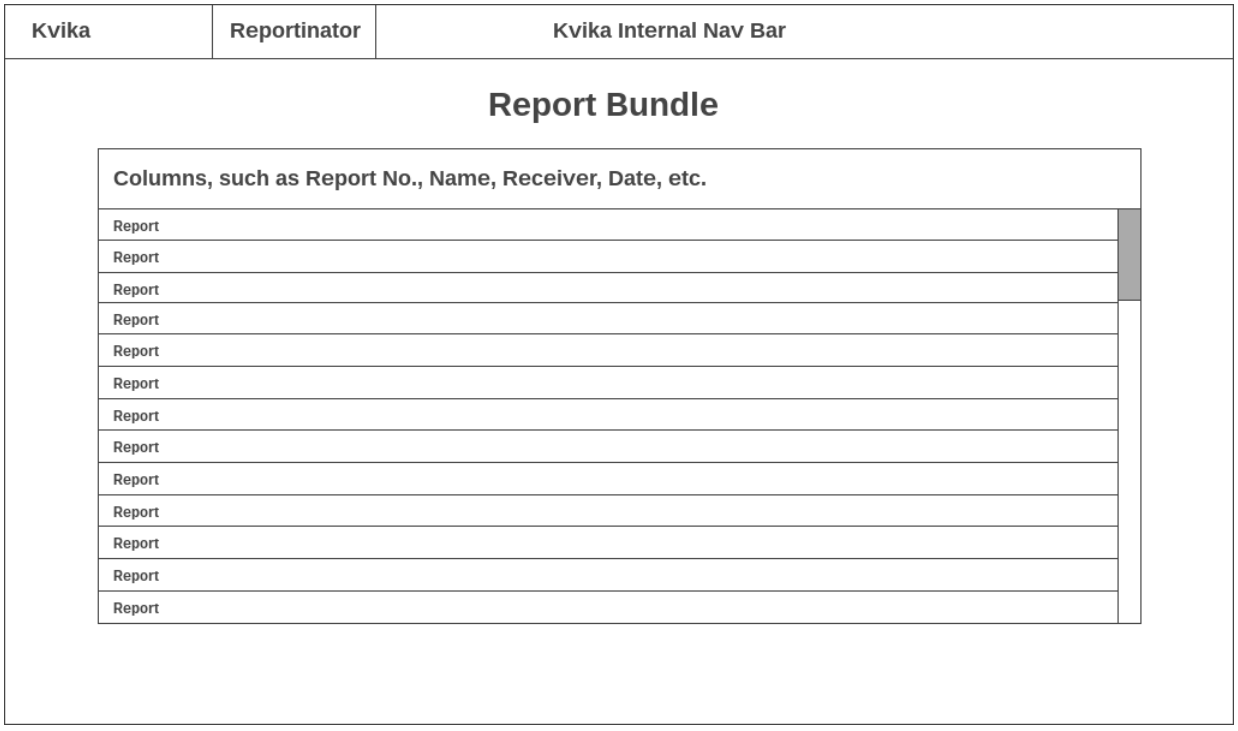


Figure 11: Lo-fi Prototype: Report Bundle View

Figure 12: Lo-fi Prototype: Report Detail View

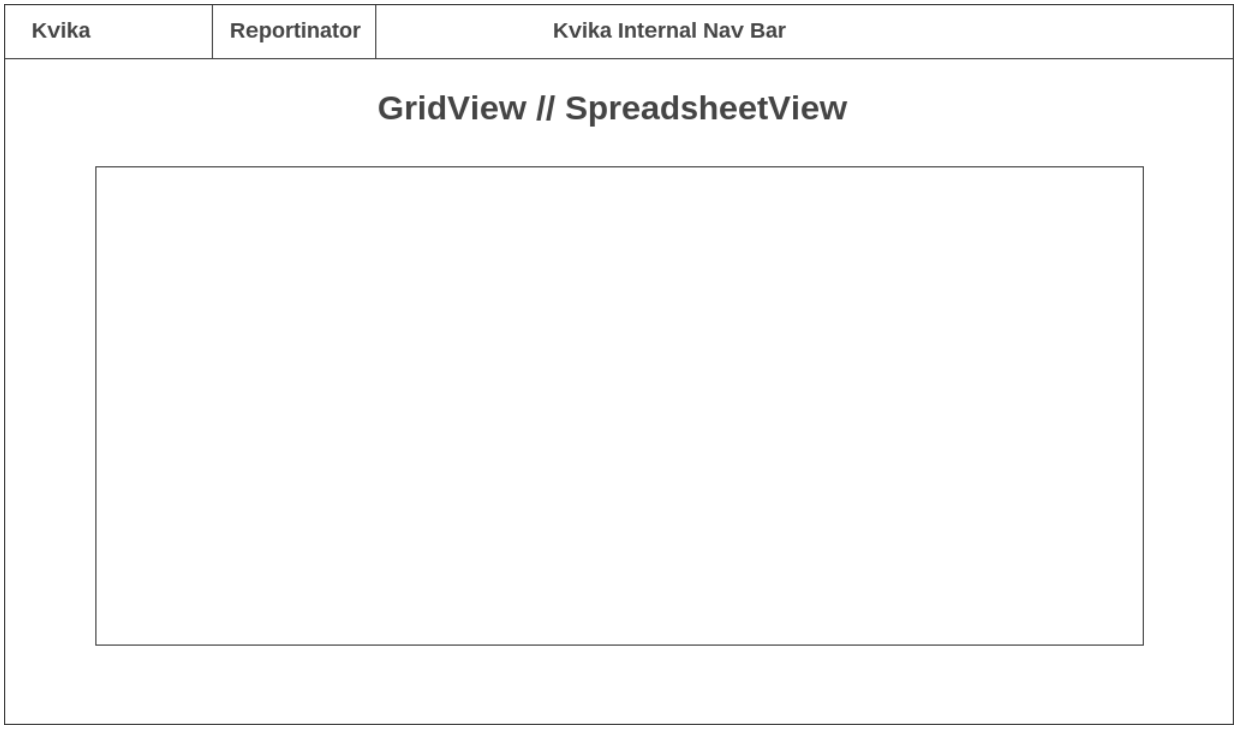


Figure 13: Lo-fi Prototype: Spreadsheet View

9.2 Intermediate Design

The intermediate design is done to visualize the flow within the system better. It's written in HTML, CSS, and JavaScript and will help us in the long run when we start coding the topmost layer which is scheduled to take place in the fourth sprint of phase 2.

The intermediate design is shown here in figures 14 through 17 here below:

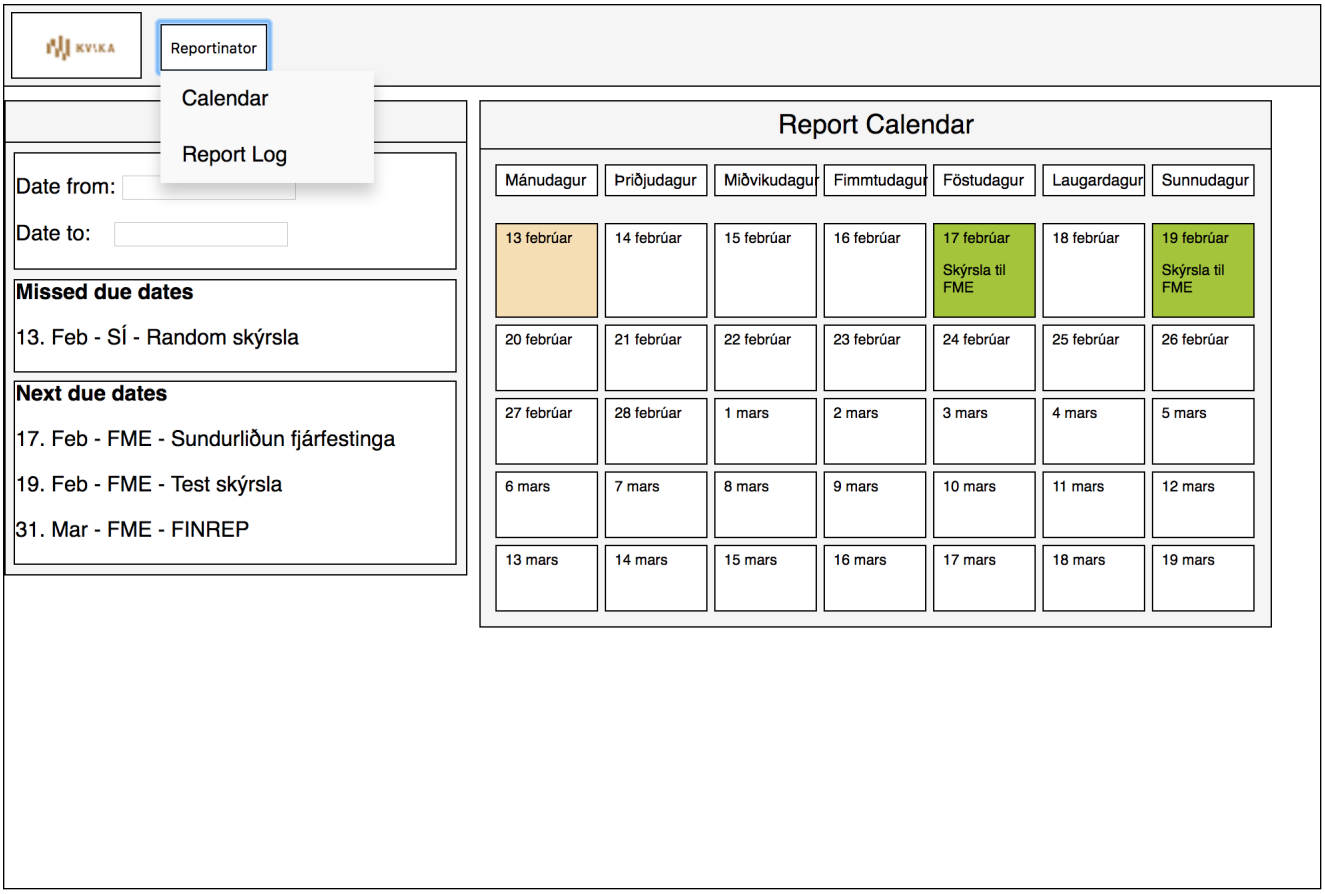
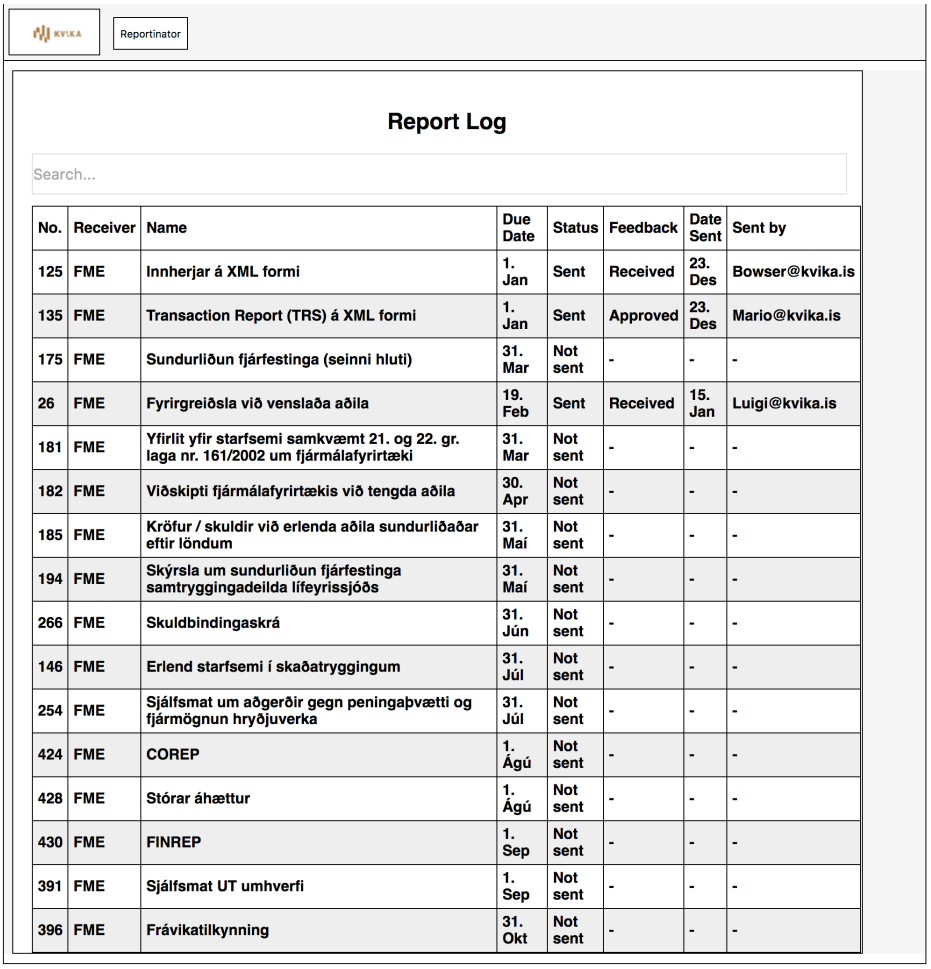
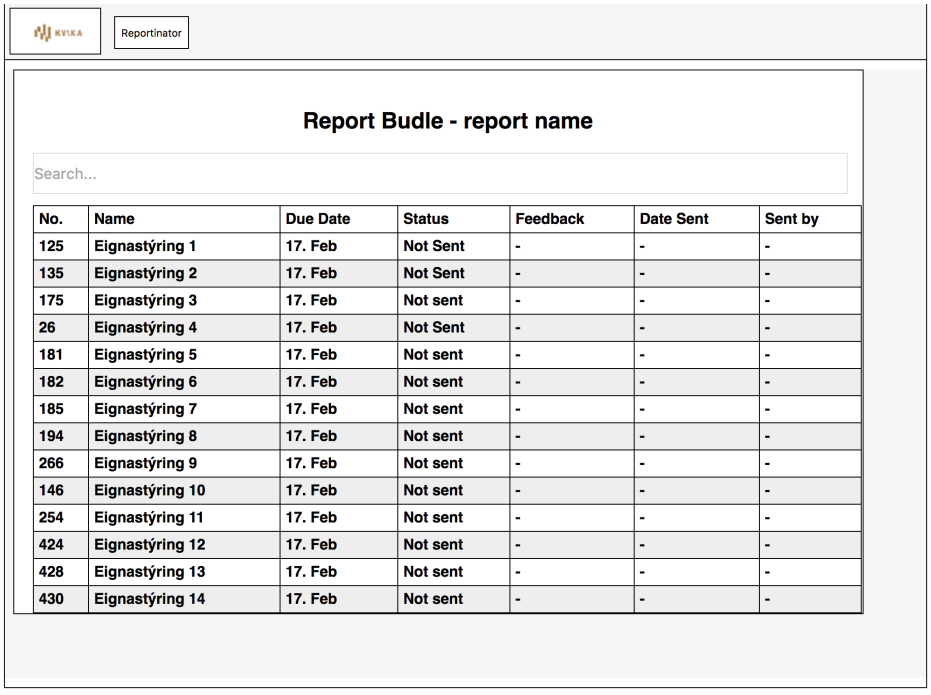


Figure 14: Intermediate Design: Calendar View



No.	Receiver	Name	Due Date	Status	Feedback	Date Sent	Sent by
125	FME	Innherjar á XML formi	1. Jan	Sent	Received	23. Des	Bowser@kvika.is
135	FME	Transaction Report (TRS) á XML formi	1. Jan	Sent	Approved	23. Des	Mario@kvika.is
175	FME	Sundurliðun fjárfestinga (seinni hluti)	31. Mar	Not sent	-	-	-
26	FME	Fyrirgreiðsla við venslaða aðila	19. Feb	Sent	Received	15. Jan	Luigi@kvika.is
181	FME	Yfirlit yfir starfsemi samkvæmt 21. og 22. gr. laga nr. 161/2002 um fjármálafyrirtæki	31. Mar	Not sent	-	-	-
182	FME	Viðskipti fjármálafyrirtækis við tengda aðila	30. Apr	Not sent	-	-	-
185	FME	Kröfur / skuldir við erlenda aðila sundurliðaðar eftir löndum	31. Maí	Not sent	-	-	-
194	FME	Skýrsla um sundurliðun fjárfestinga samtryggingadeilda lífeyrissjóðs	31. Maí	Not sent	-	-	-
266	FME	Skuldbindingaskrá	31. Jún	Not sent	-	-	-
146	FME	Erlend starfsemi í skaðatryggingum	31. Júl	Not sent	-	-	-
254	FME	Sjálfsmat um aðgerðir gegn peningaþvætti og fjármögnun hryðjuverka	31. Júl	Not sent	-	-	-
424	FME	COREP	1. Ágú	Not sent	-	-	-
428	FME	Stórar áhættur	1. Ágú	Not sent	-	-	-
430	FME	FINREP	1. Sep	Not sent	-	-	-
391	FME	Sjálfsmat UT umhverfi	1. Sep	Not sent	-	-	-
396	FME	Frávikatilkynning	31. Okt	Not sent	-	-	-

Figure 15: Intermediate Design: Report Log View



No.	Name	Due Date	Status	Feedback	Date Sent	Sent by
125	Eignastýring 1	17. Feb	Not Sent	-	-	-
135	Eignastýring 2	17. Feb	Not Sent	-	-	-
175	Eignastýring 3	17. Feb	Not sent	-	-	-
26	Eignastýring 4	17. Feb	Not Sent	-	-	-
181	Eignastýring 5	17. Feb	Not sent	-	-	-
182	Eignastýring 6	17. Feb	Not sent	-	-	-
185	Eignastýring 7	17. Feb	Not sent	-	-	-
194	Eignastýring 8	17. Feb	Not sent	-	-	-
266	Eignastýring 9	17. Feb	Not sent	-	-	-
146	Eignastýring 10	17. Feb	Not sent	-	-	-
254	Eignastýring 11	17. Feb	Not sent	-	-	-
424	Eignastýring 12	17. Feb	Not sent	-	-	-
428	Eignastýring 13	17. Feb	Not sent	-	-	-
430	Eignastýring 14	17. Feb	Not sent	-	-	-

Figure 16: Intermediate Design: Report Bundle View

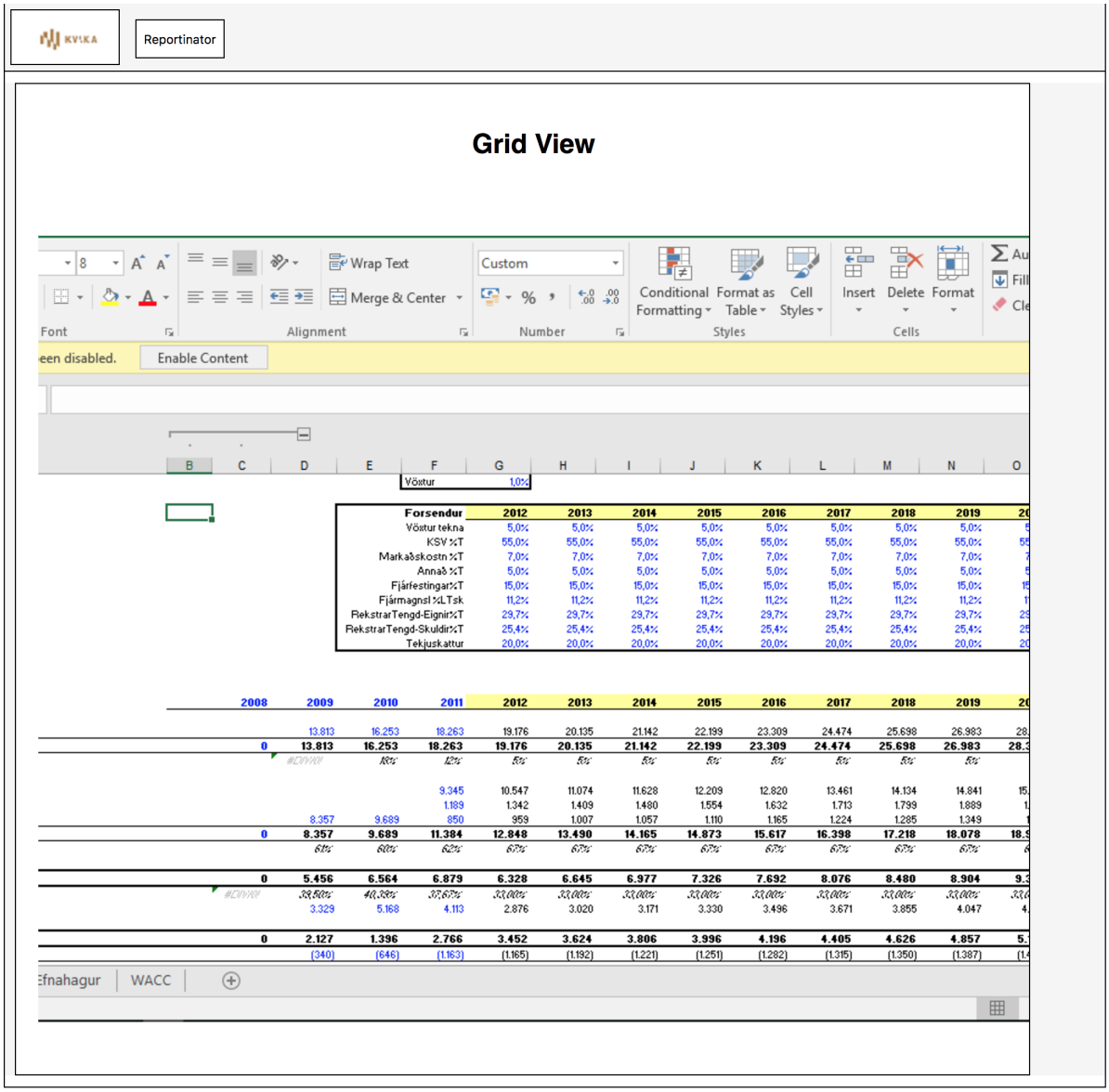


Figure 17: Intermediate Design: Spreadsheet View

9.3 Testing The Prototypes

Tests on the prototypes were done through informal interviews with a future user of Reportinator from the financial, risk management and IT departments. The tests were done on all three versions of the design:

1. Low Fidelity Prototypes - Process: Creating a questionnaire, showing the user the prototypes on a projector in a meeting room following a the questions from the questionnaire and asking their opinion on the design.
2. Intermediate Prototypes - Process: Almost identical to the Lo-Fi testing but instead of just showing the future users the pictures of the design we gave them the mouse to view the website. The intermediate design was written in javascript, html and css and was then further used in the later implementation of the views in the final product. The users thought the meeting was pretty entertaining and looked forward to see the project move on.
3. End design (almost) - Process: The end pretty much end design was tested in the same way as the intermediate prototypes were tested but the special focus of this test was to try and finalize the look of the system and to make it as user friendly as possible. We got very positive feedback and got very good points regard color schemes and such that the users wanted to see in the web application.

9.3.1 Questionnaires

Low Fidelity Prototypes

1. What do you think about the design?
2. What do you think of the overall design and base functionality?
3. Anything you want to add?

Intermediate Prototypes

1. What do you think about the design?
2. Do you think the flow within the system is good?
3. Anything you want to add?

End Design (almost)

1. What do you think about the design?
2. How would you like to have the color scheme and effects within the web application?
3. Anything you want to add?

10 Sprint Documentation

10.1 Sprint 0

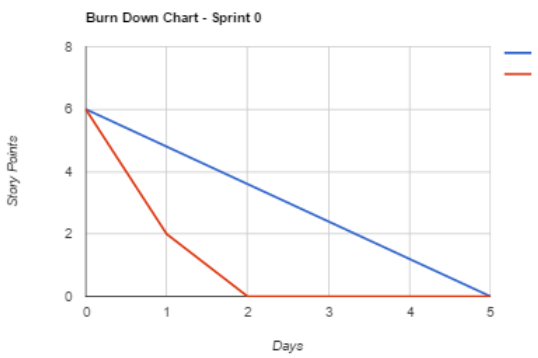


Figure 18: Burndown Chart

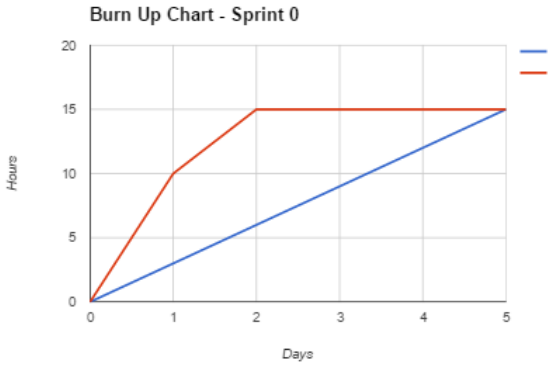


Figure 19: Burnup Chart

Remarks: This sprint was a set out as a trial sprint, the main goal was to setup the programming environment and get an end-to-end flow for the system. The sprint unfortunately ended up being on the same week as the team was to deliver the first project report therefore much of the time which was set out as time spent programming went to writing the project report and getting all the deliverables in order to turn them in. The end-to-end flow through the system was a big bite to take, vastly underestimated by team members and that milestone was pushed back until it became relevant. The backlog for the sprint was wholly rewritten to meet these new discoveries and the focus was set on the local database of Reportinator. The local database for Reportinator is explained in the Table Schema. The stories include creating data and collecting data from and to the Reportinator local database, thus both service and data layer programming, for use in the system where it was appropriate.

The hourly estimate for each story point during the sprint was 2.5 hours for each point, calculated from the product backlog. This estimate was put to the test during the sprint.

Product Backlog

This is the rewritten product backlog after realizing that the schedule would not hold for the sprint. A lot of stories were moved to future sprints and the backlog (Figure XXX) consists of the stories completed during the sprint. Unfortunately Jira did not keep track of the changes, mostly because of lack of knowledge of how Jira is usually used. Thus the backlog is not as informative as it should be, but the stories removed/uncompleted are mostly concerning data collection and services related to the generator of the reports. The burn up and burn down chart here below are created from the updated product backlog and are representative of the finished stories within the sprint, but burn up chart is the more informative since the team tried to include all the time spent on different stories in the dataset.

Completed Issues

Key	Summary
RPN-438 *	"As a user, I want the system to have a database table for report logs to keep track of which reports are in the system. "
RPN-439 *	As a user, I want the system to have a database table for report bundle to keep track of which reports are within each report bundle.
RPN-440 *	"As a user, I want the system to have a database table for reports to keep track of what information is within each report. "
RPN-441 *	"As a user, I want the system to have a database table for XML templates to automatically fetch each template from database. "
RPN-442 *	"As a user, I want the system to have a database table for Internal Errors generated when creating a report to list them up for manual fixing. "
RPN-443 *	"As a user, I want the system to have a database table for External Feedback when turning in a report to list them up for manual fixing. "

Figure 20: Project backlog for sprint 0

Daily Scrum summary

The daily scrum meetings were held on the 10th and the 11th of February. During the former meeting the team member set out to configure their work environments and to begin with the large task ahead beginning with the local data layer programming of Reportinator. The latter meeting the full scope of the complexity had struck in and the team decided to lower their ambitions for this sprint and move stories to the future sprints.

Sprint review

Programming the local database for Reportinator was a larger task than expected but was the total time it took was lower than the hourly estimate of 2.5 hours for each story point, thus the team reduced the hourly estimate to 2 hours for each story point for the following sprint. During this sprint the team finished 6 story points over 12 hours of programming over two days, the time it took to configure the work environment and updating the product log and scrum backlog was not take in the account.

The burn up and burn down chart are very steep because the team only worked on the sprint for two days (in the beginning of the sprint), but the sprint it self was 5 days.

The product owner was content with the progress of the sprint and gave his opinion in the table schema and the database implementation.

Sprint retrospective

Since the main goal of the sprint was to be a trial run we considered it a success, but the realization that we were starting to fall behind on the schedule was hard. The schedules for future sprints needed to be more realistic in order for us not to fall further behind on the schedule. The programming went well when considering the renewed backlog and the time estimate per story point was lessened from 2.5 hours to 2 hours.

10.2 Sprint 1: 15. February - 26. February

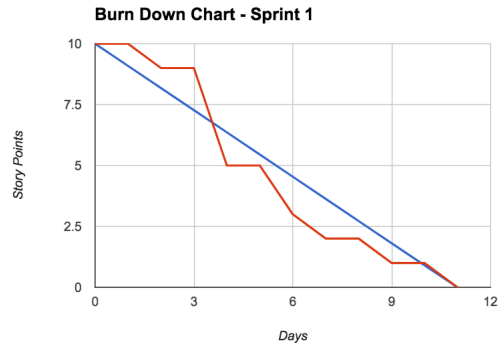


Figure 21: Burndown Chart

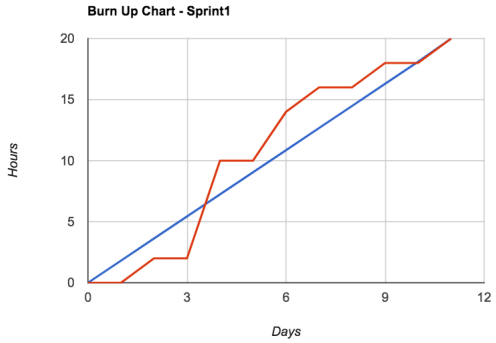


Figure 22: Burnup Chart

Remarks: This is the first sprint of phase 2, the team set out to start on the navigation through the system. That involves populating the tables starting with the ReportLog table, though during the sprint the team decided to make a table called ReportOverview which would handle populating the ReportLog table. The ReportOverview table was added to keep track of all the report types within the system "Sunduliðun fjáfestinga" would then be one item in that table, which would correspond to 12 items of that report in the ReportLog table since it is turned in monthly. The hope was to be able to start on the ReportBundle table, which holds the individual reports within the ReportLog table (one for each investment fund operating under Kvika). Another object on schedule was the calendar view, but that was done using the DevExpress plugin which Kvika has access to. The idea for the sprint was that one of the developers would take on the ReportLog and the other tackle getting to know DevExpress.

During this sprint we held a meeting with the heads of IT, risk management and financial departments of Kvika where they explained a concern that the system would not be not that useful unless the users of the system could alter the data flow to every part of the report. In other words that the system had to be implemented with dynamic SQL in mind, this meant that the team had to go back to the drawing board and reconfiguring the design for the whole system. the meeting took place at 21. February. Thus much of the last days of the sprint were spent entirely on re-designing the system, not programming.

The hourly estimate for each story point during this sprint was 2 hours per point and the product backlog was created with that in mind.

Product Backlog

Completed Issues

Key	Summary
RPN-12	As a user, I want to have the calendar page as the homepage of the system to have overview over upcoming reports when entering the system.
RPN-36	"As a user, I want the report log to have an ""error with any reports within the report bundle"" column to make it easier to sort through errors with turning in a report. "
RPN-37	"As a user, I want the report log to have a ""number of the report"" column to see the number of the report. "
RPN-38	"As a user, I want the report log to have a ""receiver of the report"" column to see who we're sending the report to. "
RPN-39	"As a user, I want the report log to have a ""how many reports are in the bundle"" column to see the size of the report bundle. "
RPN-41	"As a user, I want the report log to have a ""due date"" column of the report to see if we're turning in the correct report. "
RPN-42	"As a user, I want the report log to have a ""how many report have been sent in"" column to see how many reports in the bundle went through without an error. "
RPN-43	"As a user, I want the report log to have a ""status of the report bundle"" column to make sure that the report has been sent in or not. "

Figure 23: Project backlog for sprint 1

Daily Scrum summary

The daily scrum meetings were held on 16, 17, 19, 21, 24 and 26 of February. During the first daily meeting it was decided that Emil would take on DeVExpress and Magnus would work on the ReportLog functionality. The following meetings went on smoothly where each of the developers would explain what they had done the previous workday and what the plan was for the day to come, mostly keeping the plan set out before the sprint explained in the remarks here above. Not much got in the way of programming, mostly just bad documentation on the DevExpress website.

Sprint review

The sprint went quite well with team members completing the set task but the upset of having to implement dynamic SQL hit the group pretty hard, thus not being able to add more stories to the sprint. But the work in the beginning of the sprint seems to have been underrated since the time burnup and burndown charts were not impacted, though having to spend much of the last days re-designing the system. The DevExpress programming was slow at first but picked up as the sprint went along and the calendar was almost mostly finished and populated identically to the ReportLog.

Sprint retrospective

When looking back on the sprint the work structure was good, not to make the sprint backlog to big before getting into the sprint and it was decided to keep that mentality going forwards. The sprint would have been a big success would it not have been for having to re-design the whole system.

10.3 Sprint 2: 26. February - 12. March

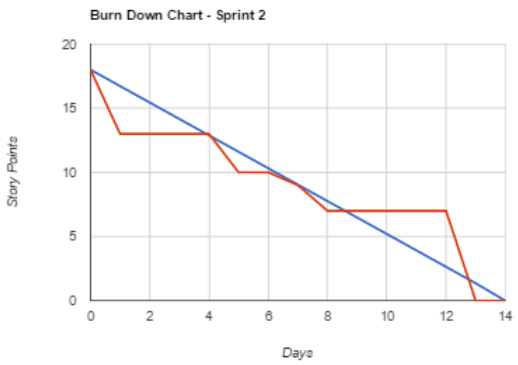


Figure 24: Burndown Chart

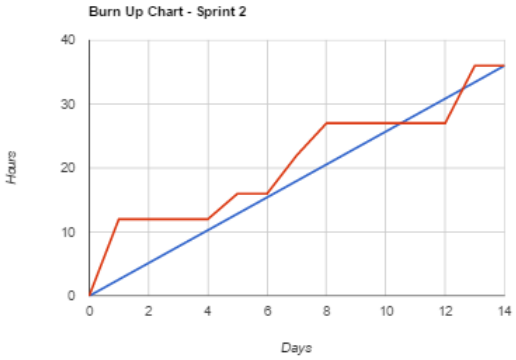


Figure 25: Burnup Chart

Remarks: The team set out to keep working on the navigation, or in other words to populate the tables which lead to the following view. The hope for this sprint was to get to the final page or the ReportDetails page, thus completing the functionality on every page within the system excluding the spreadsheet view. A few requirements related to the calendar did not make it into sprint 1 because of the upset, and completing them was a priority during this sprint.

The hourly estimate did not change between sprints one and two since the estimate was pretty spot on. Thus the product backlog for this sprint was created with the estimate of 2 hours per point same as for sprint 2.

Product Backlog

Completed Issues

Key	Summary
RPN-8	As a user, I want to have a Report Detail page to show more details on the report.
RPN-9	As a user, I want to access The Calendar page from the navigation bar to make navigation within the system easy.
RPN-10	As a user, I want to access The Report Log page from the navigation bar to make navigation within the system easy.
RPN-23	As a user, I want the calendar to start at today's date to see the next upcoming report.
RPN-24	As a user, I want the calendar to have a scroll mechanism to travel to previous or future dates.
RPN-51	As a user, I want the report bundle to have a field at the top which shows the receiver of the report bundle to have fewer columns in the list.
RPN-52	"As a user, I want the report bundle to to have a field at the top which shows the due date of the report to have fewer columns in the list. "
RPN-53	"As a user, I want the report bundle to have a ""error within the report"" column to see if any internal error was found while making the report. "
RPN-54	"As a user, I want the report bundle to have a ""number of the report"" column to assure the user that the user is working on the correct report. "
RPN-55	"As a user, I want the report bundle to have a ""name of the individual report"" column to make sure that the user is working on the correct report. "
RPN-56	"As a user, I want the report bundle to have a ""status of the report"" column to see if the report has been sent in or not. "
RPN-57	"As a user, I want the report bundle to have a ""feedback from the receiver"" column to make sure that the receiver has accepted the report. "
RPN-58	"As a user, I want the report bundle to have a ""date sent"" column to keep track of when the report was sent in. "
RPN-59	"As a user, I want the report bundle to have a ""sender of the report"" column to be able to trace the report back to the user who created the report. "

Figure 26: Project backlog for sprint 2

Daily Scrum summary

The daily scrum meetings were held on 27. February, 3, 5, 6, 7 and 11 of March. The meetings went well mostly just team member explaining their work structure as explained here in the remarks above. The main impediments were that the connection to Kvika's DataMart was not yet there, since they had not yet implemented the database which the system was to connect to.

Sprint review

The sprint went well, the team finished all the set stories and added stories related to the ReportDetails view and finished them as well. Thus the main three views and the functionality related to them was mostly finished, but still some work left in the last view, ReportDetails. The final look of the views was not to be a high priority until the functionality in the system was well on the way. The calendar was pretty much wrapped up and the two views, ReportLog and Calendar, were combined into a single view since they pretty much showed the same thing. Thus the navigation through the system was mostly finished at the end of the sprint. The product owner liked the idea of combing the two views and gave his input on the progress, which was positive.

Sprint retrospective

The sprint was a success, the system taking on a form corresponding to our design. The time each story point was rather higher during this sprint than the first two sprints, around 3 hours per point so the team decided to up the time each point took for the next sprint.

10.4 Sprint 3: 13. March - 26. March

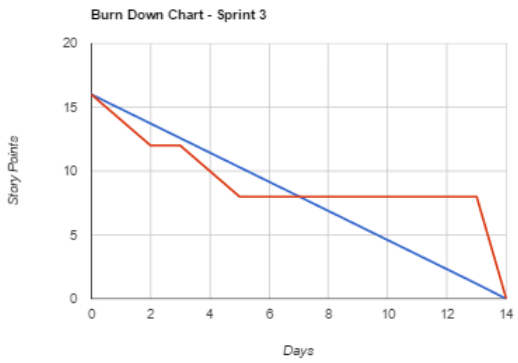


Figure 27: Burndown Chart

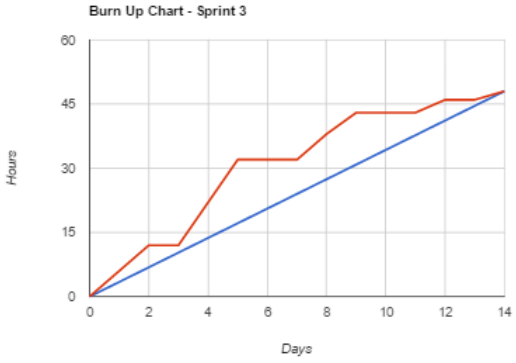


Figure 28: Burnup Chart

Remarks: The team set out to program the model which would make the "Sundurliðun Fjárfestinga" report which is turned in to Seðlabanki Íslands as well as to start getting the navigation within the system. The model programming involves programming a XML parser and a XML generator. Both to make a XML string from the model (XML generator) and to populate the model back from the XML string (XML parser). The parser was to view reports that have already been generated by the system, because the system only stores XML strings from older reports (and data related to the making of the report, but not each variable in the model it self). The stories which were left for the ReportDetails page were also finished during this sprint.

The hourly estimate for this sprint was 3 hours per story point, decided after the previous sprint retrospective. The product backlog was populated accordingly.

Product Backlog

Completed Issues

Key	Summary
RPN-61	"As a user, I want the report details page to list the number of the report to make sure the user is working on the correct report. "
RPN-62	"As a user, I want the report details page to list the receiver of the report to make sure that the report is being sent to the correct reciever. "
RPN-63	"As a user, I want the report details page to list the name of the report to make sure the user is working on the correct report. "
RPN-64	"As a user, I want the report details page to list the due date of the report to make sure that the user is working on the correct report. "
RPN-65	"As a user, I want the report details page to list the status of the report to let the user know if the report has been sent in or not. "
RPN-66	"As a user, I want the report details page to list the feedback on the report from the receiver to let the user know if the receiver has accepted the report. "
RPN-67	As a user, I want the report details page to list the date when the report was sent to keep track of when reports are sent in.
RPN-68	As a user, I want the report details page to list the sender of the report to be able to trace back to the author if there is any trouble.
RPN-76	As a user, I want the system to make a single XML format for the Sundurliðun Fjárfestingar to Seðlabanki Íslands (SÍ) report to make the system easy to use.

Figure 29: Project backlog for sprint 3

Daily Scrum summary

The daily scrum meetings were held on 13, 14, 16, 17, 20, 21, 24 and 25 of March. The meetings went smoothly and team members just telling each other what the plan was and what they did last work day according to the remarks here above. The main issues were that the access to DataMart was still not accessible, thus impeding any population of the model which was in the making.

Sprint review

The sprint went quite well the programming for the ReportDetails page went very well but the main focus was on the model and the XML related programming. The model went alright and the implementation is explained in the SFSI Class Diagram, the XML programming was the tricky part since the XML string is very complicated and our take on the problem was that the string was not designed to be parsed like this. But in the end the team finished the set task. The product owner was pleased and thought the implementation was quite good and not overly complicated.

Sprint retrospective

The sprint went well, but when considering the original plan the team was behind schedule. The system was slowly coming together but according to the original schedule it should have be a lot further. Not only that but we sunk far more time into the model than we had expected, which seemed to be the trend for this and the previous sprint. The team was taking on more complicated stories and they were taking more time than anticipated. It was agreed upon that the only way to complete the system in the given time frame was to put in more hours. When calculating the total time each point took during this sprint the estimate was closer to 4.

10.5 Sprint 4: 27. March - 9. April

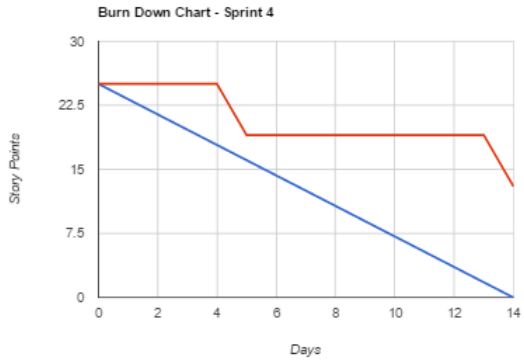


Figure 30: Burndown Chart

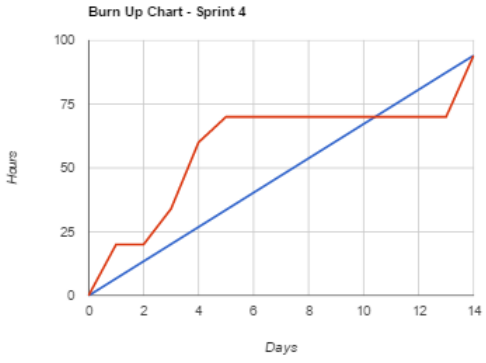


Figure 31: Burnup Chart

Remarks: One notable event had a huge impact on this sprint. The final exams on 6th and 7th of April which the team members had to take. This took much of the time away for programming during this sprint. Since the hourly estimate of each story point was increased to 4 there were not many stories in this sprint. The main ones were to automatically fil in the ReportLog table from the Seðlabanki Íslands servers and to generate the ReportBundle table from Kvika's database. Also to create the internal checks for the Sundurliðun Fjárfestinga report.

The product backlog was created with a 4 hours per story point estimate. The effect of the exams can be seem on the burnup and burndown charts.

Product Backlog

Completed Issues

Key	Summary
RPN-6	As a user, I want to have a Report Log page to show upcoming reports that need to be sent in and previously sent reports.
RPN-7	As a user, I want to have a Report Bundle page to show which reports are in the bundle.
RPN-29	"As a user, I want the report log to have a list of all reports to access report easily. "
RPN-46	"As a user, I want the report bundle to list all reports within the bundle to see what is being sent away. "

Issues Not Completed

Key	Summary
RPN-87	As a user, I want the system to validate the data before sending the reports away to lessen the risks of sending away reports that aren't correct.

Figure 32: Project backlog for sprint 4

Daily Scrum summary

The daily scrum meetings were held on 27, 29, 30 and 31 of March as well as 9. April. Where Emil started to work on the internal checks and Magnus went on to automatically create the tables. The main issue, still, was that there was no connection to DataMart.

Sprint review

Since there still was not a connection to Kvika's databases the team decided to forgo the database connection but rather to create dummy tables in the internal database of Reportinator which held data for 33 reports, corresponding to 33 investment funds. These tables were populated from the stored procedures which collected the data previously, but Reportinator was not supposed to use them riginally. The data was then multiplied by a constant and all names removed from the data to make the data untraceable.

The internal checks took time, but got finished in the end and all the data that is sent away is validated before sending away to the receiver. The project was taking a better picture but still very much behind schedule but that is understandable since the sprint was very effected by exams.

Sprint retrospective

The internal checks like most other things in this project too more time than we expected and we were not able to finish them. The team also realized that the model was not as good as it could have been. If the model would have been created differently then working on the internal checks would have taken much less time, but configuring the model would take time which was really sparse, so we had a tradeoff between the two. We decided to stick with our current model and continue the work on the internal checks in the next sprint. A note: the next model implemented in the system should be designed in another way to make the checks easier.

10.6 Sprint 5 : 10. April - 23. April

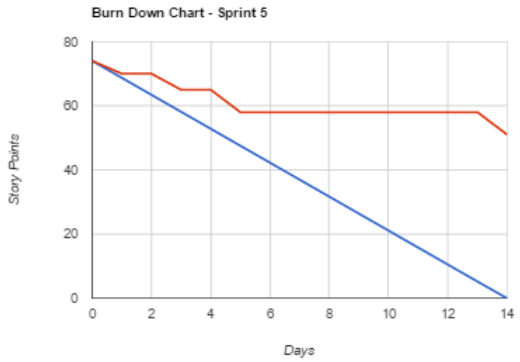


Figure 33: Burndown Chart

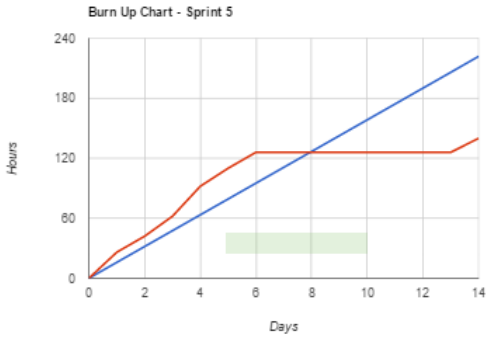


Figure 34: Burnup Chart

Remarks: Again the sprint is highly effected by an exam, but both of the developers had an exam on the 21. April. Which left little time for programming. During this sprint the team decided to put in a lot of stories and not following the guideline set in sprint 1. This was done to try and get the extra incentive to push through more stories in the sprint, which went as you can see in the burn up and burn down graphs here below. The main goal for this sprint was to implement a spreadsheet view and delve down into the documentation of DevExpress once again.

The hope was to implement another model during this sprint, but that thought was quickly put aside and will not be finished in this project. The system will only contain one report to "Seðlabanki Íslands". If we had to put the blame on something for not having addition time to implement another report e.g. one report to FME as well we would go with; lack of organization within the team, the time spent on redesigning for the dynamic sql (and then have to scrap that idea and revert back to our older design again) and the constant delay of the data mart from Kvika.

The product backlog was created with way too many stories, a flaw the team knew beforehand but we had hoped to finish as many of them as possible. The estimate for this sprint was 4 hours per story point, but the hope was to reduce that number at the end of the sprint.

Product Backlog

Completed Issues

Key	Summary
RPN-11	As a user, I want to access a grid view/spreadsheet to have the option to look over a report.
RPN-28	"As a user, I want the calendar to differentiate between reports that have been sent and reports that have yet to be sent in to make visualization of reports due easier. "
RPN-30	"As a user, I want the report log to start the list at the first unsent report to easily see upcoming reports. "
RPN-31	"As a user, I want the report log to list initially a few upcoming reports"
RPN-33	As a user, I want the report log to differentiate between reports sent in and reports yet to sent in to make readability better.
RPN-69	"As a user, I want the report details page to contain a section to list internal errors generated when writing the report. "
RPN-70	"As a user, I want the report details page to contain a section to list external feedback generated when sending the report in. "

Issues Not Completed

Key	Summary
RPN-16	As a user, I want the system to notify when it starts making a report to make sure that the system is doing it's job correctly.
RPN-27	"As a user, I want the calendar to have clickable links on every report directing the user to the reports overview page to make navigating through the system easier. "
RPN-77	As a user, I want the system to make one XML format report for Séreignarleiðir for the Sundurliður Fjárdestingar to Fjármálaeftirlitið (FME) to make the system more efficient.
RPN-78	As a user, I want the system to make one XML format report for Fjárfestingarsjóðir/Verðbréfasjóðir for the Sundurliður Fjárdestingar to Fjármálaeftirlitið (FME) to make the system more efficient.
RPN-79	"As a user, I want the system to make one XML format report for Fagfjárfestasjóðir for the Sundurliður Fjárdestingar to Fjármálaeftirlitið (FME) to make the system more efficient. "
RPN-80	As a user, I want the system to access data from the data warehouse/data mart at kvika to be able to automatically build reports to turn in.
RPN-84	As a user, I want the system to gather data from databases to save time.
RPN-86	As a user, I want the system to inject data at the correct places within a string created from the XML format to make the system as automatic as possible.
RPN-87	As a user, I want the system to validate the data before sending the reports away to lessen the risks of sending away reports that aren't correct.

Figure 35: Project backlog for sprint 5

Daily Scrum summary

The daily scrum meetings were held on 10, 11, 12, 13, 14, 15, 16 and 23 of April. The daily meetings went again mostly as described in the remarks here above, each team member describing what they were going to tackle and had tackled the previous day.

Sprint review

With the database issue resolved the team could tackle the data layer related stories but the status of the project was not that much better since the team had to create dummy data for the presentation which took time away from developing the system further. This sprint went altight, the team finished 23 story points as can be seen on the charts above. Of course a lot of stories were pushed into the next sprint, but this was a test to see if the team would perform better under more pressure. Which kind of helped but the hope was to wrap up more stories.

The spreadsheet programming took a lot of time but was successful, the spreadsheet is implemented to give the user a identical view of the excel sheet they previously made the reports in. The spreadsheet is also has the functionality to let the user make changes to the XML string before turning the report in. But Reportinator has a read only access to databases thus if there are major errors with any of the reports the user has to figure out the underlying problem and fix it at a database level.

Sprint retrospective

The idea to put in a lot of stories went well. This resulted in the most story points finished within one sprint, but has to improve if the project is to be finished with all features on the set release date. The stories took less than 4 hours to complete on average or around 3 hours, thus the estimate for sprint 6 will be reduced.

10.7 Sprint 6

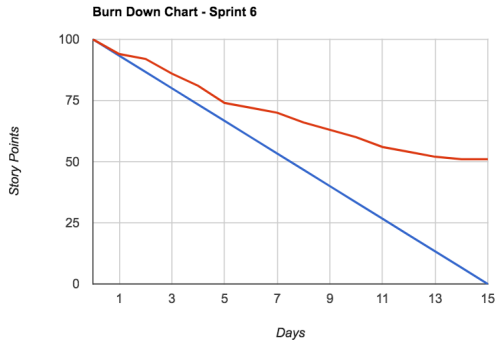


Figure 36: Burndown Chart

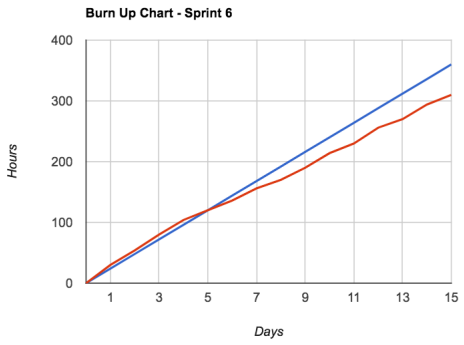


Figure 37: Burnup Chart

Remarks: This was the final sprint the team worked on the project. There was a unanimous decision to prolong this sprint until the end of week 2 of phase 3. This was done to try and finish as much as possible before the finish of the project. Overall the sprint went alright, team members worked hard and prioritized the work very much to get the main features of the system running and having a product ready at the due date of the project. Which was accomplished with some minor sacrifices of functionality which will be listed under future vision of the project. The team worked almost relentlessly and got in incredible hours during the final sprint which resulted in a finished product at the end of the sprint. The system and service works, both automatically creates reports and all functionality can be overwritten in the web application. The team tried to document as much as possible, but the product backlog got left behind when the focus was mainly on refining the existing functionality and finalizing everything related to the programming of the system.

Daily Scrum summary

Daily scrum meetings were held every day during the sprint, following the regular format. Team members talking about their goals for the day as well as listing impediments that were in the way to make sure that everything would go as smoothly as possible.

Sprint review

Review for the sprint would be summarized as follows: Everything went incredibly well, team members were very efficient, productive and found good solutions to most everything that was in their way, if anything took to much time it was set aside and put under future vision for the project.

Sprint retrospective

Main retrospective would be summarized in the question: "Why didn't all the sprint go as well as this sprint?". The retrospective for the sprint would be tense and productive, but never the less fun. We learned a lot during the sprint and got help from the two programmers at Kvika when there were problems team member were not familiar with.

11 Appendix I - User Manual

T-404-LOKA, Lokaverkefni

Reportinator - User Manual

Emil Hjaltason
Magnús Þorsteinsson

12th May 2017

Instructor:
Gunnar Sigurðsson

Examiner:
Stefán Freyr Stefánsson

Introduction

This manual is a document describing the general functionality and usage of Reportinator, the processes, and procedures that are relevant to the user. Reportinator is designed to automate report delivery to financial regulators in Iceland, and can be divided into two different sections, that is a service and a system (web application). As a service Reportinator creates, validates and sends in reports automatically to government regulators and as a system, it keeps track of both sent and upcoming deliverables.

Detailed description of Reportinator is provided in chapter Using the System. The chapter is divided into two sections, one for the service and the other for the web application. The service is explained in a manner to give the user all the information needed to interact appropriately to the different responses issued from Reportinator. Following the service chapter, there comes a chapter describing the web application. Each page is thoroughly explained with mention of all functionality relevant to the user, as well as how to react to different messages displayed in various fields of the pages.

Using the System

This chapter lists all relevant information regarding the usage of Reportinator for a user which is using the system to turn in a report or resolving errors related to a report. In addition, there is a detailed description on how Reportinator interacts with users and how to interpret various messages issued while the system is working.

Reportinator is implemented with the following deliverables:

1. Sundurliðun Fjárfestinga to Seðlabanki Íslands.
   (a) Frequency: Once a month for each investment fund operated

Users can access Reportinator on Kvika's internal web page; it is listed there under a tab in the main navigation bar on the top of the page.

Reportinator Service

The service which Reportinator provides is directly related to the automatic creation, validation, and sending of deliverables. The service will be referred to as the worker. The worker is connected to a scheduler which activates him once a day. When he is activated he checks two things:

1. Is there a report due in 7 days or less?
   If there is a report bundle due in less than seven days and the bundle has not been built the worker will create all the reports which make up the bundle. When the worker finishes creating all the reports in the report bundle, he wraps up the creation process by sending an email to the user. This email contains information about how well the creation procedure went, e.g. whether or not the reports passed all the data validation or in other words the internal checks for the report. The email can go two ways:

   - None of the reports failed their internal checks
     The user receives an email as can be seen in Figure 38. The user now knows that all the reports have been created and all of them passed the internal checks. Since no report within the report bundle contain any errors the package is ready to be sent off to the receiver. The users role is hereby finished as there are no errors to resolve, but if he wants to look further into the report see chapter: Reportinator Web Application.
   - One or more reports failed their internal checks
     The user receives an email as can be seen in Figure 39. The email will contain two tables, the first one displaying all the reports which did not pass the internal check and the latter one showing all the reports which passed the tests. The tables have two columns, the first one displaying the abbreviation of the investment fund that the report is for, and the latter one displaying a link to the report details page for the report (see Report Details Page for information about the report details page). The

user can now open the links to the reports which failed the internal checks and resolve the issues (more information about that topic in Report Details Page).

2. Is there a report due today?
   If there is a report bundle due today, the worker makes sure that the reports in the bundle have already been created and sends them off to the receiver (i.e., SI). When all the reports have been sent away there begins the same process as in check number one. The worker sends the user an email containing either the message that the reports have been sent to the receiver successfully or an email containing two tables. The first one with reports that were not sent successfully away (see Figure 38) and the other containing the successful ones (see Figure 39).

The user can overwrite all of these automatic features. Both the report log (see Report Log Page) and the report bundle page (see Report Bundle Page) have buttons which force the creation and sending of report bundles and individual reports. The main thing to have in mind is that manually forcing the functionalities will stop the worker from creating the reports seven days before the due date since he only creates the reports if they do not exist. To make sure that the user does not accidentally push buttons which he does not intend to, most of the buttons have pop-up windows with warnings about the effects of the action in question (more on this topic in Report Log Page and Report Bundle Page).

Figure 38: Email (Sending) - Containing no errors

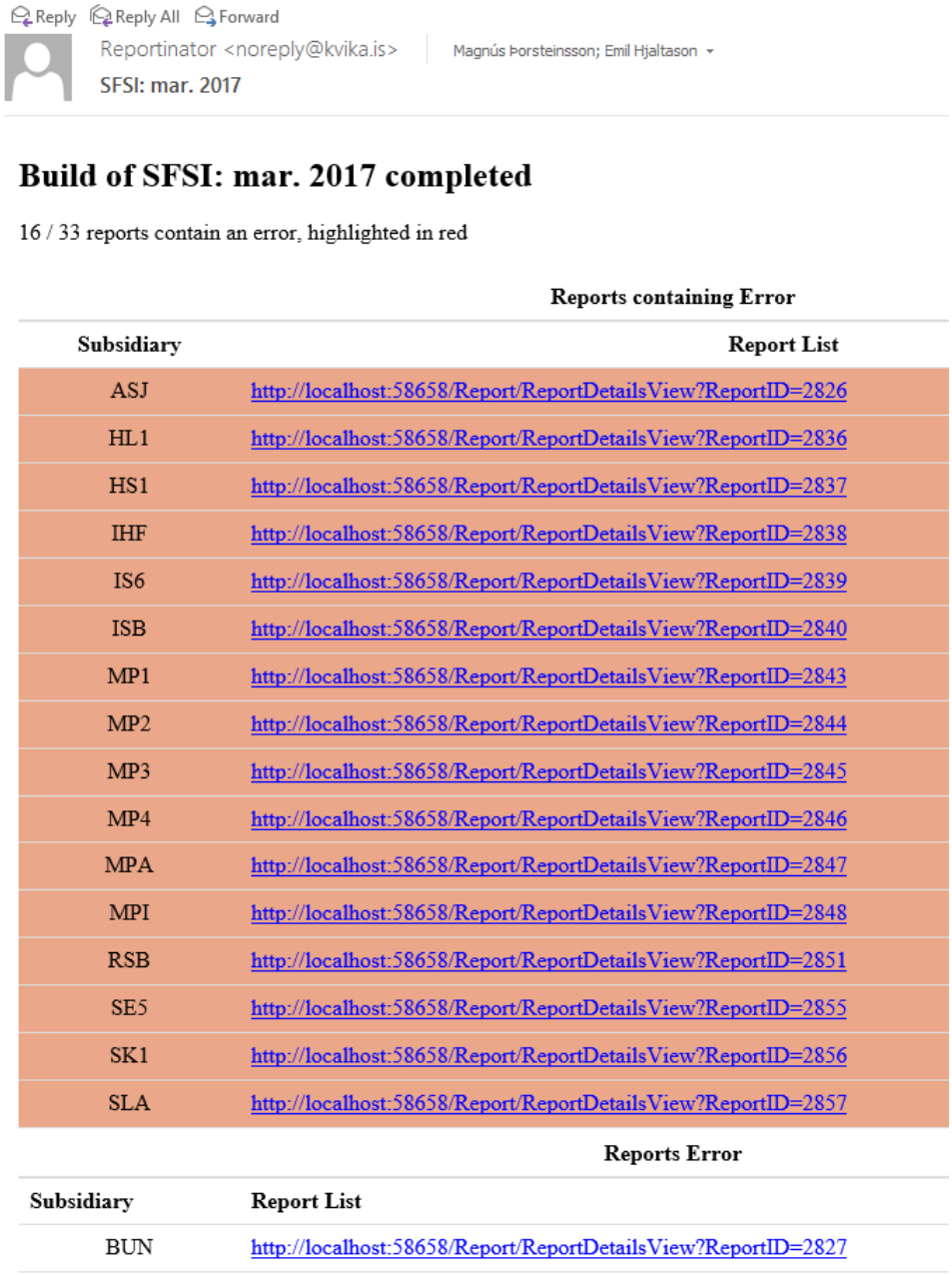


Figure 39: Email (Building) - Containing no errors

Reportinator Web Application

The web application is designed to give the user the opportunity to take over the automatic procedures which Reportinator provides, or in other words to resolve any issues related to the creation, validation, and sending of the reports. It also serves as a platform to keep track of deliverables that have been implemented in the system, such as already submitted reports, reports in the process of sending, and future reports.

The web application is designed to be straightforward and easy to use. The whole system consists of the following pages:

1. The Report Log page - Homepage
2. The Report Bundle page
3. The Report Details page
4. The Spreadsheet Editor page

Through these pages, the user can resolve most issues related to the process of creating and sending reports. One limitation of the system is that it only gathers data from Kvika's databases, but it never updates, deletes or inserts data. Thus, if an internal check fails and can not be explained it is most likely an error in the underlying database and will arise in other sections of the company (e.g. other reports).
All functionality on each of the four pages is listed and explained in the following four sub-chapters. A few features included in all pages of the system are:

1. Navigation bar - Access to Report Log (Homepage)
2. Breadcrumbs trail - Top left corner
3. Hovering over an item within the web application displays a tool-tip to explain the purpose of the item

Report Log Page

The report log page is the front page or homepage of the system. This page allows the user to access all report bundles that the system is currently managing. The page has three main features, which are explained here below, as well as in Figure 40 and Figure 41:

1. Upcoming Report Log
   The upcoming report log is the table on the left in Figure 40. It lists all reports that have not been sent away by the system. If a report is overdue, it will still be listed in the upcoming report log until it has been sent away. The columns in the table are:
   - Name of the report package
   - Error, can be:
     - Error, if there is error in any of the reports in the package
     - No error, if none of the reports contain any errors
     - N/A, if the report has not been created already
   - Due date of the report
   - Status of the report, which can be:
     - Not started, the report package has not been created
     - Not sent, the report package has been created but not sent
     - Sent, the report package has been sent away
     - Error, there is an error in one or more reports in the package
   - Build the report package column, button to force the automatic building sequence (builds all the reports that make up the report package, labeled 1 in Figure 40).
   - Send the report package column, a button in each to manually overwrite the automatic sending sequence (sends each and every report in the report package, labeled 2 in Figure 41). The buttons in this column are not activated until the report package has been built.

   By clicking on the line in the report log table, marked with a blue square labeled number 4 in Figure 40. The user is redirected to the report bundle page for the report package in the line that was clicked on. When the user clicks the button to create or build the report that triggers a pop-up window to make sure that he is not accidentally forcing an action.

2. Calendar
   The calendar is a more visual representation of the upcoming- and the old report tables, the only functionality is the visual perspective.

3. Old Report Log
   To access the old report log, the user has to click the button on the top right section of the

page, marked with a blue circle labeled 3. Clicking the button triggers a pop-up window as shown in Figure 41. The contents of the window are mainly the table. This table is just like the upcoming report log table explained here above. The columns are the same and clicking the lines will take you to the corresponding report bundle page for the report.

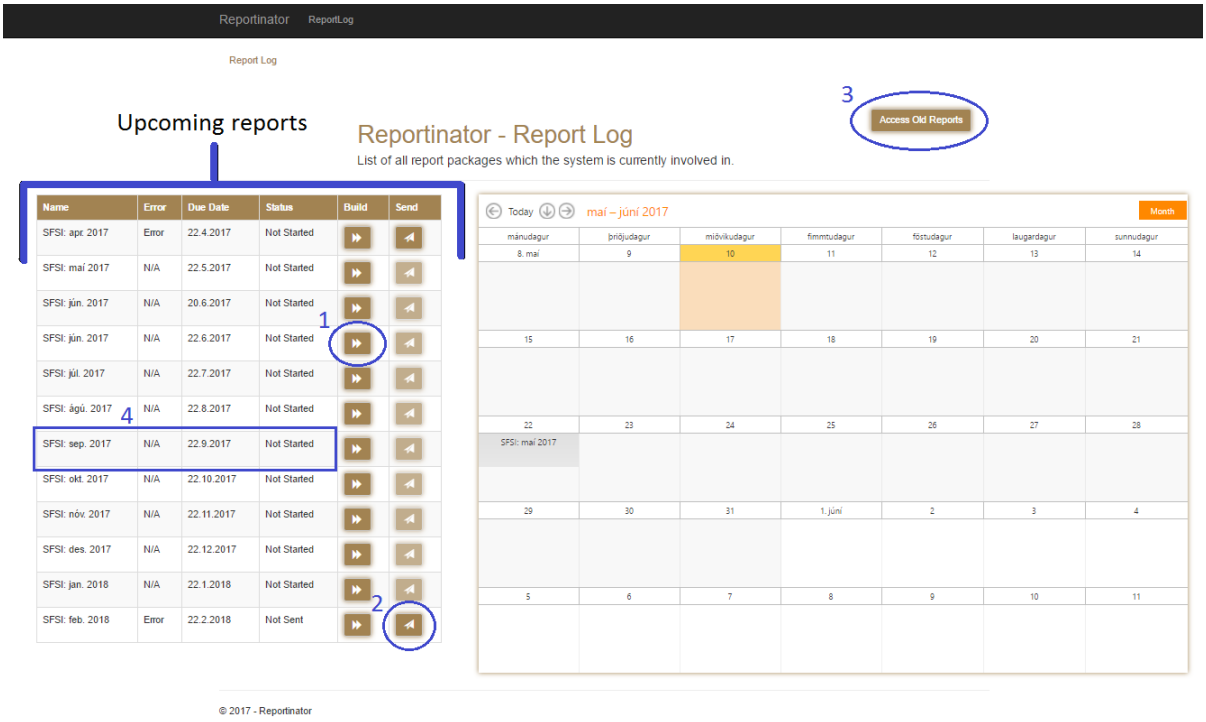


Figure 40: Report Log Page - Homepage

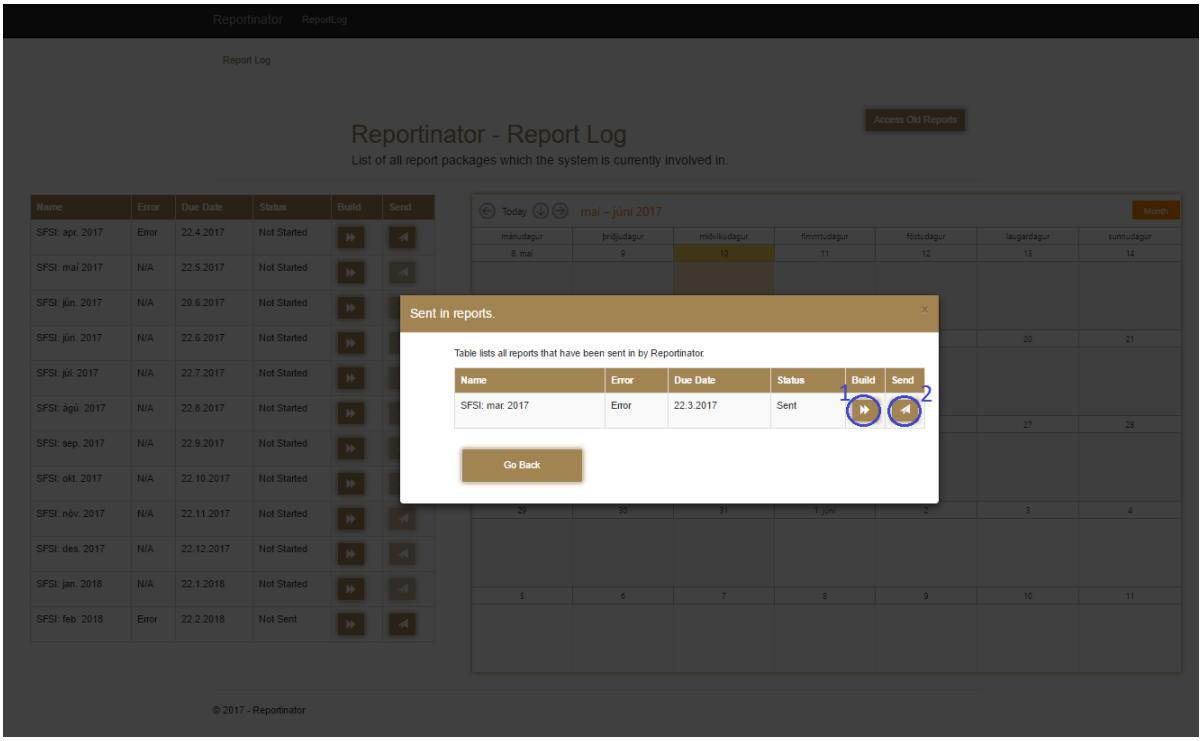
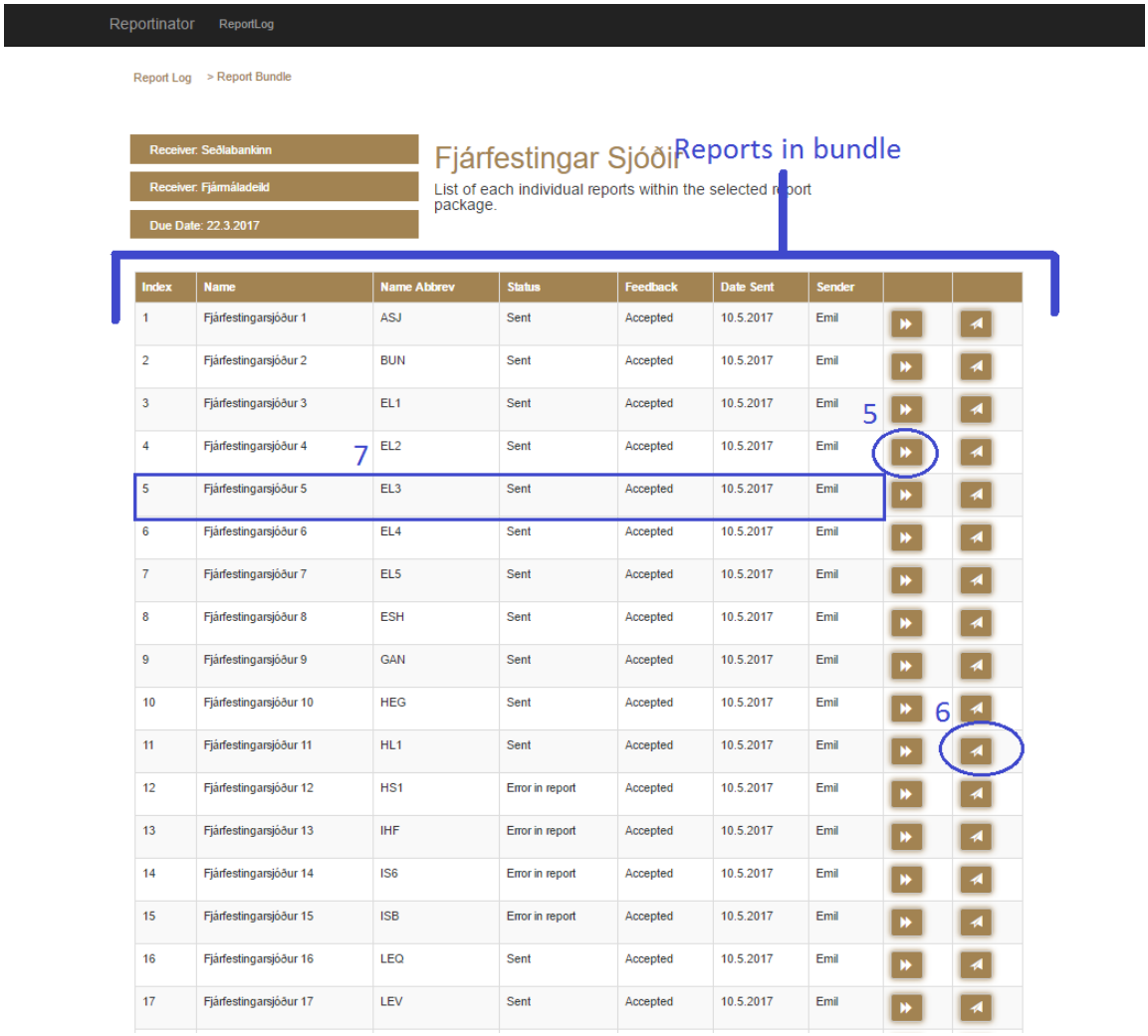


Figure 41: Report Log Page - Access to sent in reports

Report Bundle Page



Index	Name	Name Abbrev	Status	Feedback	Date Sent	Sender
1	Fjárfestingarsjóður 1	ASJ	Sent	Accepted	10.5.2017	Emil
2	Fjárfestingarsjóður 2	BUN	Sent	Accepted	10.5.2017	Emil
3	Fjárfestingarsjóður 3	EL1	Sent	Accepted	10.5.2017	Emil
4	Fjárfestingarsjóður 4	EL2	Sent	Accepted	10.5.2017	Emil
5	Fjárfestingarsjóður 5	EL3	Sent	Accepted	10.5.2017	Emil
6	Fjárfestingarsjóður 6	EL4	Sent	Accepted	10.5.2017	Emil
7	Fjárfestingarsjóður 7	EL5	Sent	Accepted	10.5.2017	Emil
8	Fjárfestingarsjóður 8	ESH	Sent	Accepted	10.5.2017	Emil
9	Fjárfestingarsjóður 9	GAN	Sent	Accepted	10.5.2017	Emil
10	Fjárfestingarsjóður 10	HEG	Sent	Accepted	10.5.2017	Emil
11	Fjárfestingarsjóður 11	HL1	Sent	Accepted	10.5.2017	Emil
12	Fjárfestingarsjóður 12	HS1	Error in report	Accepted	10.5.2017	Emil
13	Fjárfestingarsjóður 13	IHF	Error in report	Accepted	10.5.2017	Emil
14	Fjárfestingarsjóður 14	IS6	Error in report	Accepted	10.5.2017	Emil
15	Fjárfestingarsjóður 15	ISB	Error in report	Accepted	10.5.2017	Emil
16	Fjárfestingarsjóður 16	LEQ	Sent	Accepted	10.5.2017	Emil
17	Fjárfestingarsjóður 17	LEV	Sent	Accepted	10.5.2017	Emil

Figure 42: Report Bundle Page

The report bundle page is a page dedicated to showing all the reports which made up the report packages listed in the report log page. The primary purpose of this page is the report bundle table displayed in the middle of the page. The function of this table is the same as in the upcoming and old report log tables in the previous chapter. If you click a line in the table (labeled 7 in Figure 42) you are sent to the next page, the report details page. The columns of the table are as follows (skipping the obvious ones):

1. Status, see Report Log Page
2. Feedback, from the receiver either Accepted or error (for unsent reports it is N/A)
3. Date sent, blank by default filled when the report is sent
4. Sender, blank by default filled when the report is sent
5. Build, create this individual report again with data from the database, labeled 5 in Figure 42
6. Send, send the report to the receiver, labeled 6 in Figure 42

Report Details Page

Figure 43: Report Details Page

The report details page lists related to the report. The main features of the page are:

1. Internal feedback list
   This is a list, positioned on the left side, of all the errors related to the data validation i.e. the internal checks. If the list is empty, it means that the report passed all the test and is ready to send out to the receiver. If the report contains an error, it is recommended to see what the error is in the spreadsheet view, accessed in the middle of the page. The error messages start by informing what line the error is in, the error is calculated in column "H" in the spreadsheet, where you can see how large it is. It is advised to find out what is causing the error and fixing it but the user can change the value in the editor and send in the report.

2. External feedback list
   This is another list which displays error messages generated when sending in the report to the receiver. The message displays where the error is located, such as "kennitala" or due date.

3. Spreadsheet view
   The button (labeled 8 in Figure 43) in the middle of the page gives access to the spreadsheet view explained here: Spreadsheet Editor Page.

4. XML version control
   To access the XML version control, the user has to push the button (labeled 9 in Figure 43) in the top right corner which enables a pop-up window. The window displays a table containing the date when the report was created. If the user clicks on a line in the table (labeled 10 in Figure 44), he is redirected to the spreadsheet page with that version of the report.

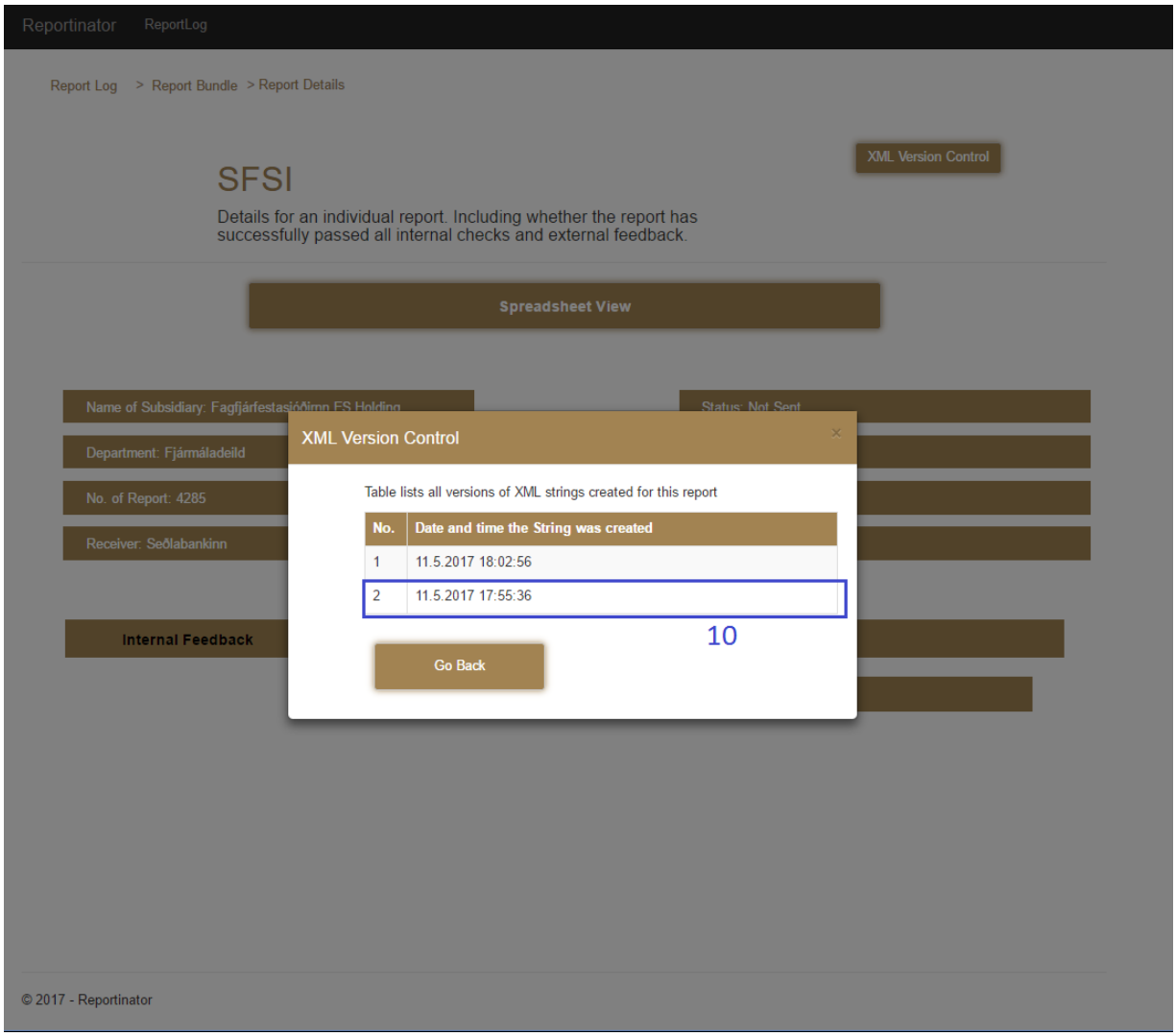


Figure 44: Report Details Page - XML Version Control

Spreadsheet Editor Page

The spreadsheet is an editor that allows the user to manipulate the data which is to be sent off to the receivers. The main features of the spreadsheet are:

1. Internal checks - The location of internal checks which are calculated for all SFSI reports are in column "H", if there is an error in a report, it is advised to go and look at the nature of the error in the spreadsheet editor. The spreadsheet editor displays all information about the report. Nearly all of the data that is used to create a report appears in the spreadsheet as well as the calculations for the internal checks and some additional information to better visualize the data which is being sent off.

2. Save - The save button is in the top left corner (labeled 11 in Figure 45). The user must press that button if he wants to save the changes to the report and thus updating the XML string, which is being sent off. If the user tries to leave the page, he will get a warning asking him whether he has already saved or not.

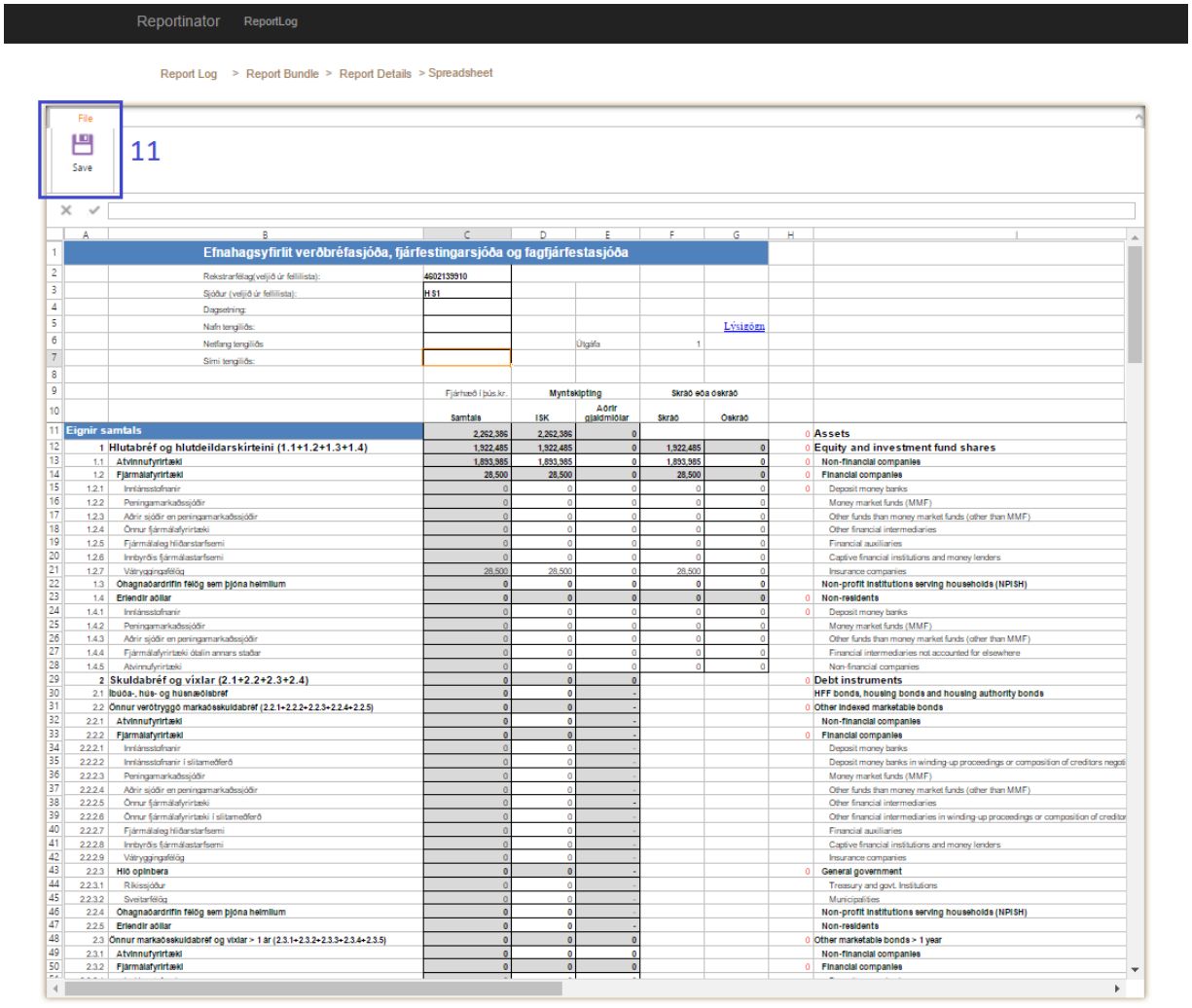


Figure 45: Spreadsheet Editor Page

12 Appendix II - Developer Manual

T-404-LOKA, Lokaverkefni

Reportinator - Operation Manual

Emil Hjaltason
Magnús Þorsteinsson

12th May 2017

Instructor:
Gunnar Sigurðsson

Examiner:
Stefán Freyr Stefánsson

Introduction

This document is an operations manual for the system and service Reportinator. Reportinator as a service generates reports in a standardized format that are to be turned in to government regulators. At the time being, the system only incorporates reports that are handed into The Central Bank of Iceland (referred to as SI hereafter) called "Sundurliðun Fjárfestinga" (referred to as SF hereafter). This manual describes the process off installing or launching the system as a standalone web application, but the service can also be integrated into another website (such as an intra-network of a company).

This manual is also an introduction to developing the system even further since the design of the system is about managing all report deliveries to government regulators. This is explained in detail in the chapters: For Developers/Programmers, and Adding New Deliverables.

This manual is written with the notion in mind that the reader has already read the user manual for Reportinator and has the general idea of what functionality is available to the user. Therefore, if the reader has yet to read the user manual it is recommended to do so before reading this manual.

For Developers/Programmers

This chapter is an introduction to the project for new developers. The structure of the project was designed to have the solution organized and relatively easy to develop further. Thus, scalability was a fundamental goal during the entirety of the design process, and a decision was taken to follow the structure shown here below. The project is divided into five sub-projects with each of them outlined as follows:

1. Reportinator

    - Views
      - Report Log View (Homepage)
            - Scheduler Partial View (Calendar)
      - Report Bundle View
      - Report Details View
      - Spreadsheet View
            - Spreadsheet Partial View

      Functionality of each view is explained in detail in Reportinator - User Manual.

    - Controllers
      - Report Controller
      - Spreadsheet Controller

2. Reportinator - DAL
   - The data access layer is divided into two sections:

    - Kvika's Database
    - Reportinator's database.

3. Reportinator - Models

    - Models - Models for Report Log
      - Models for Report Bundle
      - Models for Report Detail
      - Deliverable Models

            - SFSI Model
            - Add new report models

    - ViewModels (one for each view)

4. Reportinator - Services

    - Controller Services
      - Email Service
      - Worker Service
      - Report Log Service
      - Report Bundle Service
      - Report Detail Service
    - Deliverable Services - SFSI Service
      - Add new deliverable services
    - Global Constants
    - Scheduler/Calendar Service
    - Spreadsheet Services

5. Reportinator - Tests
   - Unit test directory.

A few things that ought to be noted; The worker is the automatic service class of the solution. One of the main features of the worker are the functions that allow for the automatic creation and sending of reports. In addition, the worker has, among other procedures, two functions which are called MonthlyCheck() and DailyCheck(). Both of these functions have to be connected to a scheduler which activates them once a day for the daily one and once a month for the monthly one. The monthly one automatically gets expected deliveries from SI to populate the report log; The daily one is to see if there are any report bundles due soon. The daily check initiates an automatic build of a report bundle if the deadline of the bundle is seven days or less from the date he is activated, only creating the reports once. When the worker has created all the reports that make up the report bundle he sends out an email. The email is to inform the user whether or not the report build was successful regarding the data validation (internal checks). If any of the reports contain errors, the user can resolve it in the corresponding report details- and spreadsheet page for the report.

Another functionality which was labeled as a future vision for the project: The same day as the report is to be turned in the worker could create all the reports again and compare the newly created XML strings with the first one the worker produced. This would be done to see if any data was changed during the period since the automatic creation, seven days earlier. If there were a difference, the worker would inform the user who could look into the change.

Adding New Deliverables

Adding a new deliverable requires creating and adjusting a few classes in the projects above. Here are the main features that need to be created:

1. Spreadsheet View & Controller & Service
   The spreadsheet editor is connected to each deliverable as the export and import is directly related to the deliverable model (SFSI model for instance) and the corresponding service.
2. Deliverable Model & Service
   The new deliverable would need a brand new model and a corresponding service to manage the model.
3. Worker and Email Service
   The worker needs to be updated for the new deliverable, but all functionality is designed to handle multiple deliverables.

The spreadsheet view might not be as usable for the new deliverable as it is for the SFSI report. Therefore, excluding the spreadsheet might be an option for any of the future deliverables included in the system.

Third Party Dependencies

The following items are required to both setup, host, and develop Reportinator further:

1. DevExpress license
2. Event handler/scheduler system (e.g. ExMon)
3. Optional but recommended: Version control system (e.g. Bitbucket license)

Installation of System

Operating Environment

The machine running Reportinator requires:

- Windows 7 or above
- .NET framework 4.6.2
- IIS 7.0 or above
- SQL server 2012 R2 or above
- Windows server 2012 R2 or above

To deploy Reportinator on IIS, perform the following steps:

1. step: Publish Reportinator from Visual Studio.
2. step: Copy the published application to the deployment destination. (usually the default directory for Windows server and IIS is "drive_name:\ inetpub \ wwwroot")
3. step: Run the IIS manager. From there you select the "Application Pools" from the left hand side and select ".NET v4.5"
4. step: Setup the database with the script supplied in the Reportinator folder "Reportinator/SQLScripts"

Now you can run Reportinator.

13 Appendix III - SFSI Class Diagram

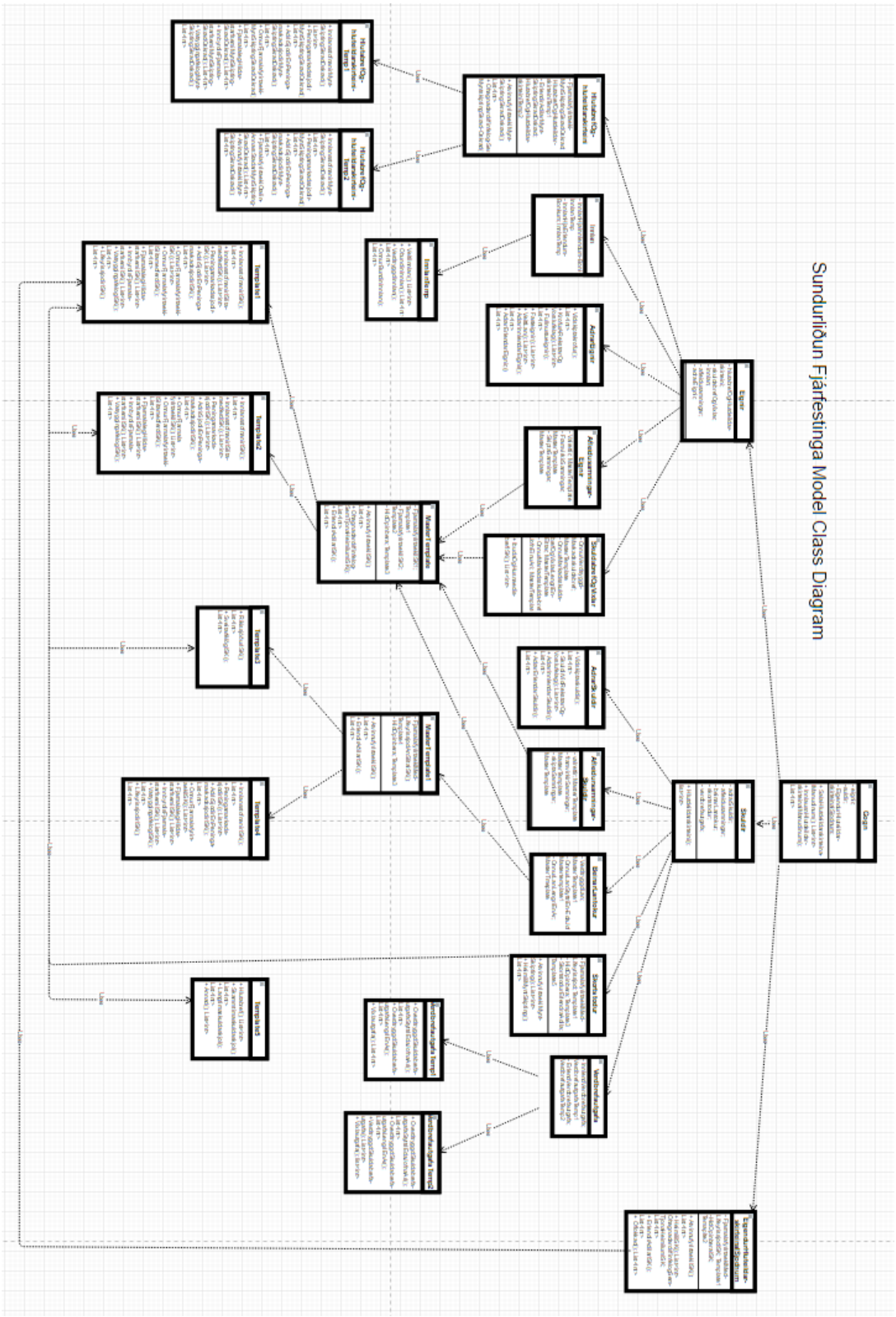


Figure 46: Class Diagram for Sundurliður Fjárfesting Seðlabanki Íslands