# Table of Contents

1 Table of Contents .................................................................................. 3

2 Introduction ......................................................................................... 5

2.1 AGR Dynamics .............................................................................. 5

2.2 Project Description ....................................................................... 6

3 Project Management ........................................................................... 7

3.1 Work Procedures .......................................................................... 7

3.2 Requirements ............................................................................... 8

3.3 Project Plan .................................................................................. 10

3.4 Risk Analysis ................................................................................ 11

4 Design ................................................................................................ 13

4.1 Research ...................................................................................... 13

4.2 System Design .............................................................................. 14

4.2.1 Dataservice Agent ................................................................... 15

4.2.2 Dataservice API ...................................................................... 15

4.2.3 Database ................................................................................. 16

4.2.4 Web API .................................................................................. 17

4.2.5 Website .................................................................................... 17

4.3 Technology Stack .......................................................................... 17

4.4 Website Prototype ....................................................................... 18

4.5 Website Prototype Testing ............................................................ 18

4.6 Website Sitemap ......................................................................... 19

5 Progress ............................................................................................. 20

5.1 Sprints .......................................................................................... 20

5.1.1 Sprint 0 .................................................................................... 20

5.1.2 Sprint 1 .................................................................................... 20

5.1.3 Sprint 2 .................................................................................... 20

5.1.4 Sprint 3 .................................................................................... 21

5.1.5 Sprint 4 .................................................................................... 21
2 Introduction

This report gives a summary of the development of the AGR System Monitor project, a final project in B.Sc. in computer science. The project is done for and in collaboration with AGR Dynamics, a software company specialized in supply chain management. The goal of the project was to develop a system that can monitor multiple off-site AGR installations and gather data centrally for monitoring and diagnostic purposes. The team is composed of three students in computer science; Arnar Ingi Halldórsson, Jóhann Ívar Björnsson and Sveinn Björnsson.

2.1 AGR Dynamics

AGR Dynamics offers a supply chain management software called AGR which is equipped with exception management, inventory optimization, and sales management. They specialize in consulting, implementation and maintenance of supply chain management solutions and support in logistics through the AGR software. The company provides services to many of the larger wholesalers and retailers in Iceland and has grown quite extensively in the last few years, mostly due to an increasing number of foreign customers, which now are over 150.

AGR Dynamics is a fast-growing business with customers and employees in multiple countries. Each instance of the AGR system is set up on the customers machine close to its Enterprise Resource Planning (ERP) system where a large portion of the setup process is setting up daily data transfer jobs between the ERP database and the AGR databases.
2.2 Project Description

The purpose of the AGR System Monitor is to provide AGR employees with an overview of the health and configuration of each customers AGR installation. The data needed should be gathered centrally and hosted on one of AGR’s machines.

Today, in order to see if anything went wrong in data transfers or other regular jobs, an AGR employee must be notified by the customer and can then connect remotely to their machine to see the status of their system. This is not optimal and requires that the customer monitors the solution.

The System Monitor should be a centralized environment with information on the status of various instances of AGR that each is set up on a customer’s machine, the relevant data needs to be sent from the customers machine daily to a receiving end in AGR’s network. There, the data should be processed and inserted into a database in the back-end of the System Monitor. This is to be done through either a web service call from the customers machine to a web service on AGR’s machine or via an email from the customers SQL service to a designated System Monitor email address, where the email can be scraped for information. The information sent is then to be displayed in a detailed view for each customer. The detailed view for each customer should show health indicators and other system information such as the version, when various regular jobs were last run, how long it took and if there are any recent extraordinary changes to these stats or customisations to the AGR instance. There should be a view which gives a clear overview of the system status for many customers at a time, with clear indicators if any AGR instance is reporting abnormalities. This should be displayed with green, yellow or red lights for data transfer, forecast runs, order calculations and AGR calculated fields respectively, where green means that the system is healthy, yellow means that there is reason to investigate it and red means that the job has failed.
3 Project Management

3.1 Work Procedures

The team chose to follow the Scrum methodology to organize this project except for a few adjustments that were made to Scrum that better fit to the teams working environment. The team is composed of three students who are all working full time so it was decided that there would be no formal daily meetings. This project was worked on during a spring semester that is split into two parts, the first was 12 weeks and the latter was 3 weeks and in between there were final exams and Easter break. Due to the variable course load the team chose to have dynamic sprint lengths instead of fixed length. The product owner is Jóhann since he works as a consultant at AGR Dynamics and is therefore best suited to prioritize tasks and features, Arnar was the scrum master and everyone were a part of the development team.

The team was provided with a workspace at AGR Dynamics to work on the project. The team had weekly meetings at AGR Dynamics to work on the project, team members also worked from home in between meetings. The team chose to communicate between meetings using Facebook Messenger, manage the sprint backlog with Asana and Excel, files shared using OneDrive and all code to be version controlled and hosted on Github.
3.2 Requirements

To better understand the requirements of the project a list of stories was defined by the team and stakeholders at AGR Dynamics. The stories were split into three categories: user stories (US), technical stories (TS) and project stories (PS). User stories are from the perspective of a person who interacts with the system, technical stories are from the perspective of the system and what it will need to be able to solve and the project stories are to capture other activities that must be done to finish the project such as report writing, analyses and design.

<table>
<thead>
<tr>
<th>No.</th>
<th>User Stories</th>
<th>Priority</th>
<th>Points</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>US-1</td>
<td>As a consultant, I want to view all customers that have the monitoring system active</td>
<td>A</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>US-2</td>
<td>As a consultant, I want to view detailed system settings for each client</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>US-3</td>
<td>As a consultant, I want to view daily job runtime in historical context for a specific customer</td>
<td>A</td>
<td>7</td>
<td>Completed</td>
</tr>
<tr>
<td>US-4</td>
<td>As a consultant, I want the interface to clearly indicate if any of the daily runs failed for each customer</td>
<td>A</td>
<td>5</td>
<td>Completed</td>
</tr>
<tr>
<td>US-5</td>
<td>As a consultant, I want to see when the forecast job was last run for a customer and that it is clearly visible when the forecasts have not been run for a certain amount of time and are therefore invalid</td>
<td>A</td>
<td>7</td>
<td>Completed</td>
</tr>
<tr>
<td>US-6</td>
<td>As a consultant, I want to be able to compare different customer nightly runtimes so I can identify factors contributing to slower running systems.</td>
<td>B</td>
<td>5</td>
<td>Cancelled</td>
</tr>
<tr>
<td>US-7</td>
<td>As a consultant, I want to see what customers have not sent data that should be sending data to be able to identify errors</td>
<td>B</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>US-8</td>
<td>As a consultant, I want to see the number of items the customer has in their inventory to better estimate how that customers system is performing.</td>
<td>B</td>
<td>3</td>
<td>Completed</td>
</tr>
<tr>
<td>US-9</td>
<td>As a consultant, I want to see a clear history of changes made to procedures and views on the AGR databases, usually recorded in the audit database</td>
<td>C</td>
<td>3</td>
<td>Cancelled</td>
</tr>
<tr>
<td>US-10</td>
<td>As a consultant, I want to see the JIRA service requests registered by each customer and a link to each request to allow me to respond quicker</td>
<td>C</td>
<td>7</td>
<td>Cancelled</td>
</tr>
<tr>
<td>US-11</td>
<td>As a consultant, I want the interface to automatically update when there is new information</td>
<td>C</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>US-12</td>
<td>As a consultant, I can see contact info for IT contact and super users for each customer</td>
<td>B</td>
<td>2</td>
<td>Completed</td>
</tr>
<tr>
<td>US-14</td>
<td>As a consultant, I want to be alerted if a report in a customer’s AGR instance has been modified incorrectly by a user so that it returns an error</td>
<td>A</td>
<td>6</td>
<td>Cancelled</td>
</tr>
<tr>
<td>US-15</td>
<td>As a consultant, I can add and remove contact information for a customer</td>
<td>C</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>US-16</td>
<td>As a consultant, I want to be able to see available storage on the customers machine</td>
<td>B</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>No.</td>
<td>Technical Stories</td>
<td>Priority</td>
<td>Points</td>
<td>Status</td>
</tr>
<tr>
<td>-----</td>
<td>-----------------------------------------------------------------------------------</td>
<td>----------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>TS-1</td>
<td>There must be a data service agent on a customer’s machine that can send structured data to an API</td>
<td>A</td>
<td>8</td>
<td>Completed</td>
</tr>
<tr>
<td>TS-2</td>
<td>The data service agent on a customer’s machine is able to retrieve log data and settings from the customer’s AGR instance</td>
<td>A</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>TS-3</td>
<td>The web service on AGR’s network can receive and manipulate structured json data sent from customers</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>TS-4</td>
<td>The data transfer between clients and the system is encrypted and only available to authenticated senders</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>TS-5</td>
<td>The receiving web service on AGR’s network can insert data into the System Monitor database</td>
<td>A</td>
<td>5</td>
<td>Completed</td>
</tr>
<tr>
<td>TS-6</td>
<td>The agent running on the customers server should know which dates should be sent and keep track of past exchanges with our data service, so it would only need to send new data</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>TS-7</td>
<td>A back end webservice for the website will be written and should adhere to the REST standard, that is be resource oriented, use HTTP verbs and follow HATEOS (Hypermedia as the Engine of State)</td>
<td>A</td>
<td>5</td>
<td>Completed</td>
</tr>
<tr>
<td>TS-8</td>
<td>The listening REST data service has its own subdomain and is accessible on the open web</td>
<td>A</td>
<td>7</td>
<td>Completed</td>
</tr>
<tr>
<td>TS-9</td>
<td>There should be set up separate development/test and production environments</td>
<td>A</td>
<td>8</td>
<td>Completed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No.</th>
<th>Project Stories</th>
<th>Priority</th>
<th>Points</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS-1</td>
<td>Complete the chapter on work procedures</td>
<td>A</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-2</td>
<td>Complete first draft of project plan</td>
<td>A</td>
<td>5</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-3</td>
<td>Complete the chapter on project plan</td>
<td>A</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-4</td>
<td>Complete the chapter on risk analysis</td>
<td>A</td>
<td>2</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-5</td>
<td>Complete first draft of design</td>
<td>A</td>
<td>3</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-6</td>
<td>Complete initial requirement gathering</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-7</td>
<td>Complete a chapter for introduction of AGR and the project</td>
<td>A</td>
<td>2</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-8</td>
<td>Decide what technology to use for the project</td>
<td>A</td>
<td>5</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-9</td>
<td>A Wireframe of the Website is presented to the AGR staff for comments and input</td>
<td>A</td>
<td>5</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-10</td>
<td>A description of each UI element is made for the final report including the wireframe and end result</td>
<td>A</td>
<td>3</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-11</td>
<td>Setup programming environment</td>
<td>A</td>
<td>4</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-12</td>
<td>Site map</td>
<td>A</td>
<td>2</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-13</td>
<td>Prepare for project status meeting 1</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-14</td>
<td>Prepare for project status meeting 2</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-15</td>
<td>Prepare for project status meeting 3</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-16</td>
<td>Complete operational manual</td>
<td>A</td>
<td>6</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-17</td>
<td>Final report</td>
<td>A</td>
<td>8</td>
<td>Completed</td>
</tr>
<tr>
<td>PS-18</td>
<td>Prepare presentation of the project</td>
<td>A</td>
<td>8</td>
<td>Completed</td>
</tr>
</tbody>
</table>
## 3.3 Project Plan

A project plan was made to better estimate what tasks need to be done and by what time.

<table>
<thead>
<tr>
<th>Period</th>
<th>From</th>
<th>To</th>
<th>Length</th>
<th>Points</th>
<th>Percentage</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 0</td>
<td>2019-01-09</td>
<td>2019-01-29</td>
<td>20</td>
<td>14</td>
<td>7.3%</td>
<td>Preparation.</td>
</tr>
<tr>
<td>Sprint 1</td>
<td>2019-01-29</td>
<td>2019-02-12</td>
<td>14</td>
<td>33</td>
<td>17.1%</td>
<td>Well defined project requirements. Site map and low fidelity prototype. Minimum interactions with web services.</td>
</tr>
<tr>
<td>Sprint 2</td>
<td>2019-02-12</td>
<td>2019-02-26</td>
<td>14</td>
<td>31</td>
<td>16.1%</td>
<td>Write data to database from web services. Implement connection between front end and backend for the monitoring app. Minimal views in front end of the monitoring app.</td>
</tr>
<tr>
<td>Sprint 3</td>
<td>2019-02-26</td>
<td>2019-03-12</td>
<td>14</td>
<td>24</td>
<td>12.4%</td>
<td>Well defined data model and dummy data for all entities needed for the front end. Detail views in the monitoring app with connections to the backend.</td>
</tr>
<tr>
<td>Sprint 4</td>
<td>2019-03-12</td>
<td>2019-03-26</td>
<td>14</td>
<td>17</td>
<td>8.8%</td>
<td>Working data flow from dummy customer to the central database.</td>
</tr>
<tr>
<td>Sprint 5</td>
<td>2019-03-26</td>
<td>2019-04-29</td>
<td>34</td>
<td>13</td>
<td>6.7%</td>
<td>Calculations and special views for the front end with interactive charts to display the data. Complete data transfer between customer and central monitoring system</td>
</tr>
<tr>
<td>Sprint 6</td>
<td>2019-04-29</td>
<td>2019-05-06</td>
<td>7</td>
<td>23</td>
<td>11.9%</td>
<td>Finished installable service for clients to send data.</td>
</tr>
<tr>
<td>Sprint 7</td>
<td>2019-05-06</td>
<td>2019-05-13</td>
<td>7</td>
<td>17</td>
<td>8.8%</td>
<td>Finishing touches on the front end.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>132</td>
<td>193</td>
</tr>
</tbody>
</table>
3.4 Risk Analysis

This section is intended to examine how project outcomes or objectives might change due to the impact of a risk event. To manage the risk, the probability and impact of each event were graded on the scale of 1 to 5. The two grades were then multiplied for a total risk score. A response to the risk event was decided and a team member made responsible for that response in the case of the risk event occurring.

<table>
<thead>
<tr>
<th>No.</th>
<th>Risk event</th>
<th>Response</th>
<th>Prevention</th>
<th>Probability</th>
<th>Impact</th>
<th>Risk Factor</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Client does not allow for log data to be sent out via web service</td>
<td>Let the SQL service send the data through email</td>
<td>Implement software in a way that only minimum rights are required for the service agent account to get data and post it to API</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>Jóhann Ívar</td>
</tr>
<tr>
<td>2</td>
<td>Team members get sick</td>
<td>The sick member of the team lets the other members know as soon as possible, and the team re-plans their sprint accordingly</td>
<td>Live a healthy lifestyle.</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>Sick team member</td>
</tr>
<tr>
<td>3</td>
<td>Very high workload in team’s day jobs</td>
<td>Organise time well and use evenings and weekends to catch up lost time due to work occupancy.</td>
<td>Plan workload ahead of time with our supervisor, keeping them informed on school workload.</td>
<td>5</td>
<td>2</td>
<td>10</td>
<td>Arnar Ingi</td>
</tr>
<tr>
<td>4</td>
<td>Project Scope turns out to be too small</td>
<td>Project has many different opportunities of near endless expansion of scope. The first expansion would be to increase the amount of data sent from client to the System Monitor</td>
<td>Discuss project scope with product owner continuously while working on it, allowing for the addition of features on the product backlog before running out of tasks.</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>Sveinn</td>
</tr>
<tr>
<td>5</td>
<td>Project scope turns out to be too large</td>
<td>Simplify each module of the system and focus on finishing a minimal version of the total system, from sending log data from the client to publishing it in the System Monitor front-end</td>
<td>Discuss project scope with product owner continuously while working on it, allowing for the subtraction of features on the product backlog before running out of tasks.</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>Arnar</td>
</tr>
<tr>
<td></td>
<td>Project Impact</td>
<td>Action Plan</td>
<td>Recommended Sprint Duration</td>
<td>Person</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
<td>-------------</td>
<td>-------------------------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Project falls behind schedule</td>
<td>Find time to catch-up in next sprint and re-evaluate future sprints based on how much workload the team evidently can manage in a sprint. Evaluate tasks and stories carefully while planning sprints to split the project into appropriately sized tasks for sprints.</td>
<td>4 5 12</td>
<td>Sveinn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Team loses workroom in AGR Dynamics office</td>
<td>Meet up in RU instead. Conduct ourselves in a manner that does not give AGR a reason to revoke the office access privilege.</td>
<td>1 2 2</td>
<td>Jóhann Ívar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Team members lose track of time spent on project</td>
<td>Estimate time spent and organise time tracking better in future Log time after each working session.</td>
<td>5 1 5</td>
<td>Sveinn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>It takes longer than expected to get data from customer to System Monitor database</td>
<td>Focus less on sending data from multiple log tables and instead send minimum amount of data for a minimal working version of the System Monitor. Start working on the service agent right away, and look at other possible ways to send data.</td>
<td>3 4 12</td>
<td>Jóhann Ívar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>AGR is not willing to host a web service outward to the internet</td>
<td>Let the SQL service send the data through email Get AGR’s permission before starting project.</td>
<td>3 3 9</td>
<td>Sveinn</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Hardware failure or breakdown</td>
<td>Use Github for code with regular commits, and OneDrive for other files. Be careful with our hardware and keep open containers away.</td>
<td>4 3 12</td>
<td>Arnar</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4 Design

4.1 Research

The first thing to do before designing a new system is to see if other systems exist that solve the same problem. After a bit of research, the team identified three tools that might be used to satisfy most of the requirements of this project, they are Splunk, Datadog and Prometheus. What these tools all have in common is that they are monitoring solutions that provide an agent to install where the target system is located and a central server that can process the data from each agent and display various metrics. The agent can be configured to ingest data from multiple sources such as log files, databases and web services.

Splunk is a mature and feature rich solution and used by many enterprises but is considered to be very expensive.

Datadog is a fast growing and feature rich solution for all around data monitoring but generally considered not as robust as Splunk when it comes to logging capabilities, it’s licensing model can also quickly become too costly due to its per agent licensing subscription.

Prometheus is an open source solution and community driven so it has the potential to be cost efficient but on the other hand it doesn’t seem to be a complete solution and needs some technical knowledge to configure and integrate with other open source solutions such as Grafana to deliver sophisticated dashboards and graphs.

Each of these solutions have their own qualities but in the end all of them would require extensive configuration to set up and the more mature solutions, Splunk and Datadog, would also incur increased recurring licensing cost which was not an option for this project. After going over these solutions it was decided that it would be better to create a specialised solution to solve this specific use case rather than using a generic tool.
4.2 System Design

When designing the system and the scope of the project, the team decided that it wanted to focus on delivering production ready, easy to use, high quality software using best practice, industry accepted standards.

The AGR System Monitor solution consists of 5 main components, a Windows service (Dataservice Agent) running on the customer’s AGR web server, a web service (Dataservice API) hosted by AGR Dynamics that is exposed on the internet, a database to store all the monitoring data and a website that AGR consultants have access to that consists of a back-end (web API) and a front-end (website).

The Dataservice Agent aggregates data from the customer’s AGR server and the AGR database and sends it via HTTPS to the Dataservice API which then saves the information to the database that the web API uses to provide data for the website.

It is important for the System Monitor to be updated frequently so each customer sends data on an interval that is configurable in length but was given a default value of 30-minutes.

The diagram below shows an overview of the AGR System Monitor components, in orange, and where each of them resides relative to the technical landscape of AGR and the customer.

![Diagram of AGR System Monitor components](image-url)

Figure 1: High level system diagram
4.2.1 Dataservice Agent
The role of the data service agent is to gather a substantial amount of data and send it securely over the internet to the Dataservice API. The agent is designed to require the minimum amount of rights to connect to the AGR databases and retrieve system health and configuration data, as well as relevant information about the Windows Server hosting the AGR system. The agent runs as a Windows service that is easy for AGR consultants to install with a simple user interface installer. The installer takes input variables from consultants and uses them to construct necessary connection strings and to identify the customer to the Dataservice API.

![Connection information](image)

Figure 2: Getting the textboxes into the installer UI was easy, retrieving the values was not.

4.2.2 Dataservice API
The Dataservice API is setup to accept incoming post requests from authorized senders. It is resource based with an endpoint for each resource. Every post to the API must adhere to the message standard that was set between the Agent and the API. When the API has received a post request for a given resource it validates the model, requests a load id from the database, writes the data to the database and then executes a procedure on the database for the given resource and load id. Each step of the communication with the database is also logged to an event model in the database to better identify errors if problems arise.
4.2.3 Database

The database is designed to be robust and resilient to errors in data transfers between customer and AGR. When the Dataservice API writes data to the database, it writes to a staging area. After writing to the staging area, it executes a procedure on the database that then merges the staged data into an identical table on the main schema where it inserts data that it previously did not have or updates the data if it has changed. The logic for how to use and process the data is contained in database views that target the tables on the main schema. The System Monitor web API can then read data from database views.

Figure 3: Database design
4.2.4 Web API
The web API is programmed in .NET core and adheres to the REST standard, that being resource oriented URIs, interaction with resources using HTTPS verbs (GET, POST etc.) and being HATEOAS, that is having a defined hypermedia and using that to guide the API usage with URIs pointing to other relevant data available. A JSON HATEOAS provider called AspNetCore.Hateoas was used and since there was no package available, the code is included in the project Github repository. The web API is written using the repository pattern, with separation of concern, loose coupling and dependency injection for the data context. A distinct hostname for the web API was used, secured with a certificate and served over HTTPS.

4.2.5 Website
The website is programmed in React JS which runs in the browser of the user. The aim was to have the code as modular and reusable as possible and also to be easily read and understood. Each component has it’s own CSS definitions in a file in the same folder that is loaded through CSS Modules. This project uses Redux to manage state, with a few exceptions like opening and closing modals and other UI related things. React-Router is used to navigate the UI and it also accepts explicit URLs typed into the browser to navigate straight to the location of a specified customer or his configuration page. The website, like the web API, is served over HTTPS with a certificate and it also has its own hostname. The final user interface design can be seen in Appendix 7.2.

4.3 Technology Stack
When deciding what technology to use it is not only important to pick the best technology for each solution but also look to the technical knowledge of those who will implement the given solution and those who will use and maintain it. The current technologies used at AGR Dynamics include SQL Server, C#, .Net, dotnet Core and Angular 6. The team was familiar with most of these technologies except for Angular but two members of the team had some knowledge of React so it was decided to build the front-end website for the system using React and keep the web API in C# running on the dotnet core framework. The Dataservice Agent was written in C# running on the .Net 4.6 framework and not the dotnet core framework since some research indicated that the older framework had better support and knowledge base to create windows service agents. The Dataservice API was written in C# running on the dotnet core framework and utilizing Entity Framework to get the data to the database. The data is stored in a SQL server database since all of AGR Dynamics’s software runs using that technology although the team had more experience with Oracle SQL.
4.4 Website Prototype
A prototype was first made on paper for the team to get a general idea of how the final product should look like. After deciding on a rough draft of the UI a high-fidelity prototype was made using Adobe XD to be tested by consultants at AGR. The design of this prototype can be seen in the Appendix 7.1.

4.5 Website Prototype Testing
During early stages of the project, an interactive prototype was tested with three consultants at AGR. Each was asked to complete two task scenarios using the prototype and encouraged to think out loud while doing so.

The three participants are among the group of people that will eventually become the users of the system. The participants are 25, 31 and 50 years old respectively, with 2-4 years of experience working for AGR. Each test was conducted in the System Monitor development team’s workroom in AGR’s offices. To encourage the users to comment on the prototype while testing, it was emphasized that they were only being shown a mock up with no functionality.

The first task given to participants was checking which version of AGR a certain client had set up and the second task was to see if any of the daily runs had run into problems. After completing the tasks, the following questions were asked about their experience:

1. Was there anything you found counter-intuitive or bothered you in some way about the system?
2. Was there anything in the system that you found unnecessary?
3. What features of the system do you think will be the most useful in your job?
4. Were there any features missing that you would have like to see?
5. How likely are you to use the product?

The full interview with answers can be found in the Appendix 7.3.

Each of the participants managed to complete their tasks somewhat easily. Two participants specifically mentioned that they liked the daily job graph under customer details, and one participant stated that he was happy to see the status of forecast runs on the system, as she has had problems with the sales forecasts for some of her clients being outdated without any alerts raised to her or the client. One participant mentioned that the JIRA Service desk plug-in would probably not be useful because soon, AGR consultants will get a designated Service Desk monitor up on a screen in the office to monitor open service requests. Therefore, a JIRA service desk plug-in to the AGR System Monitor would be obsolete.

All three consultants agreed that the system was intuitive and simple to use. One consultant stated that more specifications on the customers SQL Server would be helpful with explaining some cases of performance issues. The consultants were all certain that the AGR System Monitor would serve as an important tool for AGR Dynamics to pro-actively deal with errors in the daily runs.
4.6 Website Sitemap

The user interface navigation is relatively simple and consists of a frontpage where all customers are listed. From the frontpage a user can navigate to an overview for a specified user and from there the user can navigate to more detailed information for the user like the System Configuration and a list view of recent critical errors.
5 Progress

5.1 Sprints

5.1.1 Sprint 0
This sprint was mostly focused choosing a project, discussing the scope of the project, deciding on what technology to use, starting to familiarize with the chosen programming languages and starting on project planning.

Stories finished: PS-1, PS-2, PS-8.

5.1.2 Sprint 1
This sprint was focused on continued system design, requirements gathering and preparation for the first project status meeting. A high-fidelity prototype for the web application was made and presented to a few employees at AGR who had a few comments on the design.

Stories finished: PS-3, PS-4, PS-5, PS-6, PS-7, PS-9, PS-10, PS-12, PS-13

5.1.3 Sprint 2
This sprint was focused on getting a proof of concept by getting basic data from a test customer to our central system and displaying the results. The tasks assigned for this sprint were mostly finished except for a few minor tasks that were moved to the next sprint.

5.1.4 Sprint 3
This sprint was focused on improving the pipeline for dataflow from customer to the central database, improvement of how the database receives and works with the data and viewing the sent data in the user interface. Customer info such as operation system, SQL server specifications and other useful information and sending of log data was added to the data pipeline. Work for preparing for the next status meeting that was scheduled in this sprint was moved to the next sprint due to scheduling of meetings.


5.1.5 Sprint 4
This sprint was focused working on a configuration file for the Dataservice agent that can store customer specific information such as database connection string and more, installer for the service but without success. The sprint was also focused on preparing for status meeting 2, updating the interim report and preparing the presentation.

Stories finished: TS-7, PS-14

5.1.6 Sprint 5
This sprint was focused on tying up loose ends and leftovers from previous sprints, improving database jobs, connecting runtime data to an interactive chart in the web interface and finishing the configuration settings for the Dataservice agent.

5.1.7 Sprint 6
This sprint was focused on setting up servers and services on the AGR network and configuring the web service to serve dotnet core applications and react web client. Changes were made to the programming environment to support development, testing and production environments. Improvements were made to the data pipeline to support detection of outdated forecasts in the AGR system. Front page was made more responsive, and redux implementation was changed to improve user experience.


5.1.8 Sprint 7
This sprint was focused on preparing for the final status meeting, adding features to the UI like automatic refresh, changing the chart from Google Charts to Highcharts and configuring the chart to disable empty data series. Currently running procedure and recent critical errors were added to the UI and implementation of filters on the customer list overview was started.


5.1.9 Sprint 8
This sprint was focused improving user experience, by adding more detail to some views and offering a tooltip to further explain some subjects. Some bugs were fixed in parts of the database logic, bugs related to the react router and user cookies. Disk space information on the customers machine was added to the data pipeline and to the UI.

Stories finished: US-16, PS-16, PS-17, PS-18
5.2 Overall Progress

The status of the project was mostly on track, it strayed a bit off track when the final exams started up until the three-week course started. The general requirements of the project mostly remained unchanged but some of the features requested turned out to be not needed like integration with the Jira service desk (US-10) and detailed comparisons of two separate customers (US-6) while other requirements such as monitoring if custom user reports were failing (US-14) turned out to be infeasible at this point and would need to be addressed by the AGR software first before being able to include in the System Monitoring solution. Other requirements were added in the final sprints after consulting with AGR staff to compensate for the stories that would not be included.

5.2.1 Project Burn-Down
5.2.2 Time Tracking

The work hour plan was good to have since it provided a clear goal how much time should be spent on the project. The planned hour goal was set by estimating 13.5 hours per week during the first 12 weeks, 5 hours per week during the exams and over Easter break and then 46 hours per week during the last 3 weeks so that the total hours worked for each person would be around 300 hours.

<table>
<thead>
<tr>
<th>Period</th>
<th>From</th>
<th>To</th>
<th>Length</th>
<th>Planned hours</th>
<th>Worked hours</th>
<th>Arnar</th>
<th>Jóhann</th>
<th>Sveinn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sprint 0</td>
<td>2019-01-09</td>
<td>2019-01-29</td>
<td>20</td>
<td>116</td>
<td>70.5</td>
<td>26.5</td>
<td>18</td>
<td>26</td>
</tr>
<tr>
<td>Sprint 1</td>
<td>2019-01-29</td>
<td>2019-02-12</td>
<td>14</td>
<td>81</td>
<td>109.5</td>
<td>30.5</td>
<td>30</td>
<td>49</td>
</tr>
<tr>
<td>Sprint 2</td>
<td>2019-02-12</td>
<td>2019-02-26</td>
<td>14</td>
<td>81</td>
<td>89</td>
<td>27</td>
<td>20</td>
<td>42</td>
</tr>
<tr>
<td>Sprint 3</td>
<td>2019-02-26</td>
<td>2019-03-12</td>
<td>14</td>
<td>81</td>
<td>72.5</td>
<td>22.5</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>Sprint 4</td>
<td>2019-03-12</td>
<td>2019-03-26</td>
<td>14</td>
<td>81</td>
<td>79</td>
<td>29</td>
<td>29</td>
<td>21</td>
</tr>
<tr>
<td>Sprint 5</td>
<td>2019-03-26</td>
<td>2019-04-29</td>
<td>34</td>
<td>73</td>
<td>98.5</td>
<td>28.5</td>
<td>28</td>
<td>42</td>
</tr>
<tr>
<td>Sprint 6</td>
<td>2019-04-29</td>
<td>2019-05-06</td>
<td>7</td>
<td>138</td>
<td>156</td>
<td>47</td>
<td>66</td>
<td>43</td>
</tr>
<tr>
<td>Sprint 7</td>
<td>2019-05-06</td>
<td>2019-05-13</td>
<td>7</td>
<td>138</td>
<td>136</td>
<td>49</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>Sprint 8</td>
<td>2019-05-13</td>
<td>2019-05-21</td>
<td>6</td>
<td>118</td>
<td>133.5</td>
<td>46.5</td>
<td>44</td>
<td>43</td>
</tr>
</tbody>
</table>

Total Hours Worked

- **Arnar**: Total 907 hours
- **Jóhann**: Total 944.5 hours
- **Sveinn**: Total 306.5 hours
- **Total**: Total 335 hours
5.2.3 Division of labour

The division of labour when the project started was that each member would focus on separate aspects of the project, Jóhann focused on the Dataservice Agent, Arnar focused on the Dataservice API and database, Sveinn focused on the web service API and the website. When the project progressed, the team focused more on the user interface together.
6 Conclusion

6.1 The Project

The project as a whole provided good and useful experience for all team members. The system included many different components, and each had its challenges and opportunities to learn new things and the following chapters go into detail on the lessons learned for each component.

6.1.1 Dataservice Agent

The biggest challenge of developing the Data Service Agent turned out to be the task that seemed relatively trivial in the beginning, namely the installer for the service rather than the service code. The installer needed to have a simple user interface for AGR consultants to input information about the customer and their server, such as the account key, SQL server name and database names, and then construct application configuration from that user input. The custom input fields were relatively easy to add to the user interface, but having the installer retrieve the values from the user interface and then set the application config accordingly was a bigger challenge.

After a fair share of reading and trying out different tools recommended on online forums, such as the Windows XML Installer, the end-result was a Visual Studio Setup Project that finally managed to get the installer to do what was intended with the user input variables.

6.1.2 Dataservice API

The challenges the team faced with the Dataservice API was that the Entity Framework used to communicate with the database is not designed to handle large data sets so it uses a lot of memory when a customer is sending an initial load which is usually around 360 times larger than a normal load. To improve this it would be good to look into using SQLBulkCopy which is a relatively new feature of dotnet core.

6.1.3 Database

The database architecture was efficient in receiving data in bulk from the Dataservice API and providing it to the web API in the development and test stages of the project. When the system was deployed to a production environment and started receiving a large amount of data on regular intervals from actual customers and was being used by several AGR consultants a time, the consultants started experiencing notable performance issues. As the servers provided by AGR Dynamics to host the database, web API and website were only allocated 2 Gigabytes RAM, the importance of well optimized database tables, views and procedures became evident.

After consulting with SQL experts and researching SQL performance tuning, opportunities were found for improvement by adding indexes and primary keys to a few important tables in the database for faster reading of the tables with a small trade-off in the speed of inserts.
Another room for improvement was that by default, tables are locked from reading while receiving data to preserve data integrity. This caused views providing data to the web API to wait for inserts to a table to finish before reading. By using the WITH NOLOCK hint in views, the locks are ignored and data is read from the table immediately giving better response time in the website. The trade-off in data integrity is acceptable because out-of-sync data is preferred over slow response time in report system applications like the AGR System Monitor.

A valuable lesson was drawn from this to perform load testing on the test environment. That way, the performance issues would have been discovered earlier and dealt with before deploying to production. Fortunately, this was discovered in time to fix before the end of the development period.

6.1.4 Web API
The most important lesson in developing the web API was to study the different ways the Entity framework queries data, since minor adjustments made late in the project contributed to less load on the SQL server. The web API itself is lightweight when running on the webserver and can serve many users at once.

6.1.5 Website
While undertaking this project, two of the team members also took the advanced web programming class at RU and were learning about React, Redux and JavaScript programming in general. This was very useful for the project and the assignments from that class helped shape the work done in the front end. In the beginning, the website was set to refresh data from the server on a ten second interval. This has been modified to 30 seconds on the All Customers overview, which is the frontpage, and the most vital view since it will be up on display and then a minute for the Customer Overview. These adjustments made a great difference in load on the SQL server.

6.2 Final Product
The purpose of the project was to create a monitoring solution so AGR consultants can monitor the status of their customers setup of the AGR software. We believe we have met the expectations and delivered a complete product that fulfils the basic requirements that were outlined in the beginning of the project. Our initial design of the system has proved viable and has not changed much from the start. All priority A requirements apart from those that were cancelled were finished and most of the priority B requirements were finished.

In addition to the software, an installation manual for the Dataservice Agent and an operation manual for the System Monitor have been provided to AGR Dynamics.

The AGR System Monitoring solution is now in production and receiving data from ten customers at the time of this writing and multiple other customers that have agreed to install the Dataservice Agent. The system is performing well under the increased load.
6.3 Further Work

The final product is fully functioning and fulfils all major requested features but there are still some features and structure that would be good to develop further to make this a better product. Here are the features we would recommend implementing and suggested strategies to implement them.

**Notifications**

A running C# service that polls the database for customers with problems and sends an email to the person/persons marked as responsible for the given customer. We would implement this as a separate service on the internal network and provide users with customizations and subscription options on the website.

**User accounts for AGR consultants**

Having user accounts for every consultant would benefit if notifications were to be implemented since that would give the option for consultants to subscribe to notification for customers of their choosing. It would also be used to give management better information of who is using the system. We would recommend looking to the Identity model provided by dotnet core to implement this since a lot of the features come ready out of the box. We would use Active Directory for authentication.

**Analytics Dashboard**

The information provided by the current system is mostly on a customer detail level while it could also prove beneficial to have another view of that could display common trends of the overall system and comparisons between versions etc.

**Load Balancing**

Implementing load balancing for the Dataservice API would allow for easily allocating more resources to receive data from customers if the current setup does not handle increased traffic from more customer being added to the system and it would also allow for easier deployment of new versions of the Dataservice API without any downtime. Since our product is based on stateless sessions the load balancing could be implemented with ease using any hardware or software load balancer.

**Continuous Deployment**

Having a continuous integration/deployment system would greatly improve how the development process if further work is to be done. We would recommend using Jenkins since it is open source and free to use.
7 Appendix

7.1 High Fidelity Prototype

All Customers Overview

Elko
- Data Transfer
- Orders
- Reports
- Forecast
- Region: AGR Iceland
- AGR Version: 5.2.0

Aðföng
- Data Transfer
- Orders
- Reports
- Forecast
- Region: AGR Iceland
- AGR Version: 5.5.3

Byko
- Data Transfer
- Orders
- Reports
- Forecast
- Region: AGR Iceland
- AGR Version: 5.9.1

Costco
- Data Transfer
- Orders
- Reports
- Forecast
- Region: AGR Nordic
- AGR Version: 5.5.3

Dorma
- Data Transfer
- Orders
- Reports
- Forecast
- Region: AGR UK
- AGR Version: 5.5.1

Jysk
- Data Transfer
- Orders
- Reports
- Forecast
- Region: AGR USA
- AGR Version: 5.5.3

Vinbúðin
- Data Transfer
- Orders
- Reports
- Forecast
- Region: AGR Iceland
- AGR Version: 5.9.1

Ólgerðin
- Data Transfer
- Orders
- Reports
- Forecast
- Region: AGR Iceland
- AGR Version: 5.5.1
Customer Overview: Elko

Customer Info
- Number of items in inventory: 15,423
- Number of items in shipping: 3,002
- Number of sales locations: 14

Data Transfer Job Duration

System Configuration
- APS Version: 5.5
- APS Build Number: 2018.03.12
- MySQL Version: 2014.11
- Windows Server Version: 2012 R2
- Inventory System: SAP

Health Indicators
- Data Transfer
- Orders
- Reports
- Forecast

Open JIRA Service Requests

Customer Overview: Elko > System Configuration

Settings
- Some settings: True
- Some setting 2: False
- Some setting 3: False
- Some setting 4: False
- Some setting 5: False
- Some setting 6: False
- Some setting 7: False
- Some setting 8: False
- Some setting 9: False
- Some setting 10: False
- Some setting 11: False

Setting Detail
- Name: Some Setting
- Table Id: some_setting
- Created: 19/07/2014 16:32:13
- Updated: 19/09/2014 13:22:21
- Value: True
Customer Overview: Elko > Recent Critical Errors

Recent Critical Errors

- The service has encountered an error processing your request. Please try again.
- Cannot process create or update request. Too many create or update operations in.
- The service has encountered an error processing your request. Please try again.
- The service has encountered an error processing your request. Please try again.
- The service has encountered an error processing your request. Please try again.
- The service has encountered an error processing your request. Please try again.

Error Details

Timestamp: 24/07/2018
Error Text: The service has encountered an error processing your request. Please try again. [Error Code: 4197]

You receive this error when the service is down due to software or hardware failures, or any other failure problems. The error code (4197) embedded within the message of error 4197 provides additional information about the kind of failure or failure that occurred. Some examples of the error codes are embedded within the message of error 4197 are 40002, 40143, 41566, and 40540.

Reconnecting to your SQL Database server automatically connects you to a healthy copy of your database. Your application must catch error 4197, log the embedded error code (4197) within the message for troubleshooting, and try reconnecting to SQL Database until the resources are available, and your connection is established again.

Customer Overview: Elko > System Configuration

Settings

- Some setting 1: True
- Some setting 2: False
- Some setting 3: False
- Some setting 4: False
- Some setting 5: False
- Some setting 6: False
- Some setting 7: False
- Some setting 8: False
- Some setting 9: True
- Some setting 10: False
- Some setting 11: 10

Setting Detail

Name: Some Setting 11

Table Id: some_setting_11

Created: 23/02/2016 10:12:56

Updated: 21/08/2017 14:44:52

Value: 10
7.2 Website User Interface
7.3 User testing

The following interview was conducted with each consultant that participated in the interactive prototype testing

7.3.1 Participant 1 (25-year-old woman)

1. **Was there anything you found counter-intuitive or bothered you in some way about the system?**

   No, I think the system was intuitive and easy to navigate.

2. **Was there anything in the system that you found unnecessary?**

   The JIRA Service Desk plug-in is a cool feature but with the monitor screen from JIRA where all open service requests will be seen on that screen, I suspect that it will become unnecessary to see that information in the AGR System Monitor too.

3. **What features of the system do you think will be the most useful in your job?**

   The forecast monitoring is going to be very useful. I have had quite a few occurrences lately where some items had way too old forecasts and the order logic was proposing weird orders.

4. **Were there any features missing that you would have liked to see?**

   There are probably endless possibilities with adding information to the system, but I can’t think of anything important missing at the moment.

5. **How likely are you to use the product?**

   I will definitely be using the system, especially for the forecast monitoring. Also, if the customer overview will be up on a screen in the office, we will all effortlessly be using the system.
7.3.2 Participant 2 (31-year-old man)

1. **Was there anything you found counter-intuitive or bothered you in some way about the system?**
   
   For a second, I thought the daily job graph had something to do with the system health lights.

2. **Was there anything in the system that you found unnecessary?**
   
   No

3. **What features of the system do you think will be the most useful in your job?**
   
   The ability to have an overview of our customers setups in one place without having to log in at each customer. It makes it easy to see if it’s time to upgrade some customer to the latest version, or to estimate how long the daily job is going to take for a potential customer based on how long it takes for a current customer with a similar number of item numbers and locations.

4. **Were there any features missing that you would have liked to see?**
   
   No

5. **How likely are you to use the product?**
   
   Very likely.

7.3.3 Participant 3 (50-year-old man)

1. **Was there anything you found counter-intuitive or bothered you in some way about the system?**
   
   No, the user interface is just like you would expect from a monitoring system.

2. **Was there anything in the system that you found unnecessary?**
   
   No

3. **What features of the system do you think will be the most useful in your job?**
   
   Being alerted when something goes wrong in the daily runs, definitely.

4. **Were there any features missing that you would have liked to see?**
   
   Seeing the specifications of the SQL server machine, like the version of the operating system and available space on the disk.

5. **How likely are you to use the product?**
   
   It depends on how easy it will be to install the data service agent on the customers machine. If that’s simple, I don’t see why I wouldn’t install it at all my customers machines to be able to have overview of my customers system health, all in one place.