TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Jevgeni Gavrilov 175458IDDR

Logging and Monitoring System for BBFinance Group OÜ

diploma thesis

Supervisor: Henn Sarv
Master's degree

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Jevgeni Gavrilov 175458IDDR

Logimise ja monitoorimise süsteem BBFinance Group OÜ-le

diplomitöö

Juhendaja: Henn Sarv
Magistrikraad

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Jevgeni Gavrilov

04.12.2020

Abstract

Purpose of this thesis is to make research and offer monitoring and logging solution, what will be capable to collect, process, view received information and alert when problems occur.

Second purpose is to research and analyse most popular existing solutions on market, what can be capable to solve all problems.

Third purpose is to implement and integrate monitoring and statistics view functionality into existing Pay Out from Banks process, on these data IT-team should be able to quickly detect and solve critical problems.

As result this work provides new Monitoring and Logging system structure, what is capable to amend all existing solutions.

As second result pointed out, that there is no such out of the box solution on market what can meet all required needs.

Practical part was implemented with user interface in Angular, what collects and processes statistics and problems based on typical templates (Pending too long).

This thesis is written in English and is 49 pages long, including 9 chapters, 28 figures and 11 tables.

Annotatsioon

“Logimise ja monitoorimise süsteem BBFinance Group OÜ-le”

Käesoleva diplomitöö eesmärgiks on uurida ja pakkuda lahendust, mis võimaldaks ettevõtte BB Finance OÜ-l kõikidest selle portaalidest ja teenustest koguda informatsiooni, töödelda seda ja kuvada arusaadaval viisil IT-meeskonnale. Selle süsteemi peamine eesmärk peab olema probleemide tekkimisel saada piisavalt informatsiooni selleks, et seda kiiresti lahendada. Samas see süsteem peab olema võimeline varakult hoiatama kui midagi läheb valesti või mõni 3-nda osapoole teenus ei toimi.

Töö teiseks eesmärgiks on uurida teiste olemasolevate lahenduste sobivust lahendada üleval mainitud probleeme.

Töö kolmandaks eesmargiks on implementeerida ja integreerida olemasolevasse süsteemi pankadest väljamaksete loogika monitoorimise ja statistiliste andmete kuvamise. Nende andmete põhjal IT-meeskond/fima töötajad peavad olema võimelised varakult väljamaksetega probleeme lahendada.

Töö tulemusena on pakutud monitoorimise ja logimise süsteemi struktuur, mis võimaldab asendada kõik olemasolevad lahendused, kõikide nende puudustega.

Teiste lahenduste uurimise tulemusena tõin välja et BBFinance Group OÜ süsteemide puhul puudub selline lahendus, mis kataks kõiki nõudeid.

Praktiline osa, mis korjab, töötleb ja kuvab statistilisi andmeid ning probleemide olemasolusid sai implementeeritud ja integreeritud olemasolevasse süsteemi.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 49 leheküljel, 9 peatükki, 28 joonist, 11 tabelit.

List of abbreviations and terms

API	Application programming interface
IT	Information technology
OS	Operating system
SaaS	Software as a service
UTC	Coordinated Universal Time

Table of Contents

List of figures

List of tables

1 Goal

Primary goal of this work to create architecture of logging and monitoring system for BBFinance Group.

System should be capable to collect and store all information received from existing services, store received errors, be capable to make diagnostics to third party services, show alerts in case of some action should be done, show state of all services and logging information.

Practical part will cover a part of logging and monitoring system. This will analyze the business process when loan(money) should be transferred to company customer using third party bank services, find typical patterns when something is going wrong and display it to users.

2 Existing monitoring and logging solutions

Our current monitoring and logging solution is implemented using different tools from different vendors. These are all created using different technologies have different ways of loading and reading data and don’t communicate with each other at all.

Here are how current solutions looks like when combined together:

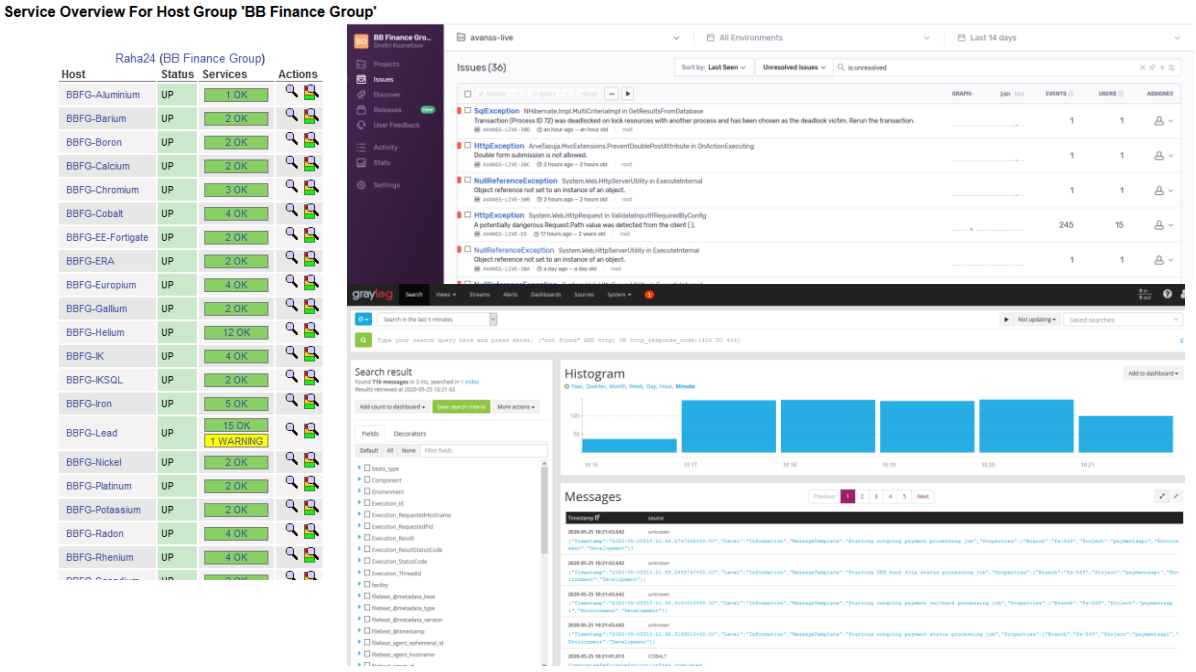


Figure 1. Existing monitoring and logging solution.

3 Problem

There are few problems that current solution has:

- No cohesive relationship between different tools. Errors found within Sentry have no clear corresponding log messages within GrayLog or it's very hard to find. Same things goes for hardware – issues reported in Nagios have no clear relationship with related errors in Sentry.
- No clear Alert functionality or it's very limited. Different tools use different approaches. Right now, only Nagios has alert functionality, when e-mail is sent out in case of errors. Sentry sends e-mails as well, but it goes to e-mail and it's not clear if it was handled or resolved. GrayLog doesn't send e-mails out, but there should be an ability to track metrics and see if something happens (that may not be an error)
- Limited filtering and search functionality. Each tool is using different approach to find and filter data. This means you need to know exactly what you want to search for (in case of GrayLog as the search query has to be correctly written for this to work properly).
- Our Application specific code is not monitored or a custom solution is implemented separately. For example, our business-critical Outgoing Payments has a diagnostics solution that was implemented with a web page that nobody is tracking. Any errors that happen there go to Sentry, but delayed payments when it happens are left unresolved until someone find an issue.
- Some errors are not tracked anywhere. For example, Windows Event Logs or SQL Server logs are not tracked at all. Any errors that happen there are not monitored.
- We don't gather statistics over time, so any changes to the code can potentially impact existing functionality that we are not aware of, for example a bug can cause e-mail sending to fail. Because we don't have a baseline of e-mails that are sent hourly, we won't be able to notice any changes.
- Different tools show different time zones. For example, GrayLog and Sentry are showing their time in UTC, this requires developer to deal with date time conversion to the correct local time zone in order to find correct error message.
- Nagios is hosted by partner company so we depend on them, if we decide to start hosting ourselves, we possibly need to find an alternative.

- Most of the time troubleshooting issues requires going through local log files or making requests in database to figure out current state of the issue.
- Right now, on one server we have 3 different environments running on the same machine and writing to the same error log, the configuration setting is different, but that's about it. It's difficult to understand through the logs which environment has triggered specific error.
- Due to mentioned above situations troubleshooting issues is difficult and time consuming. Developers and IT team don't usually know if something is broken until someone notifies us about the situation, by that time a financial damage can be already done.

4 Analysis of existing solutions

There are a lot of monitoring and logging tools available nowadays with rich functionality for analyzing, troubleshooting and logging.

Some most popular of them are:

- Datadog
- New relic
- Nagios

4.1 Analysis of Datadog

Datadog is paid SaaS based platform with a huge amount of functionality for monitoring, security and logging.

Key features:

- Seamlessly unites metrics, traces, and logs
- Aggregate metrics and events from 400+ technologies
- Search, analyze, and explore enriched log data
- Trace requests across distributed systems and alert on app performance
- Seamlessly pivot between correlated data for rapid troubleshooting [1]

4.2 Analysis of New Relic

Same as Datadog, New Relic is SaaS-based monitoring and logging platform.

Key features:

- 300+ agents and integrations, including OpenTelemetry, so you can ingest and store all of your operational data, including logs, in one place
- Query with lightning-fast response times
- New Relic One + Grafana Dashboards
- Real-time alerts
- Build custom apps on first class product APIs and components with built-in hosting [2]

4.3 Analysis of Nagios

Key features:

- Network Traffic Monitoring
- Network Analyzer
- Windows Server Monitoring
- Linux Server Monitoring
- Web Application Monitoring
- Application Log Monitoring
- Integration availability with many popular services [3]

4.4 Summary

	DataDog	**New relic**	**Nagios**
Windows Server monitoring	+	+	+
Linux Server monitoring	+	+	+
Application monitoring	+	+	+
Business logic monitoring based on patterns defined by business	-	-*	-
Custom dashboards	+	+	+
Custom views	-	-*	-
Integration with Sentry, GrayLog, Nagios	+	+	+
Alerts	+	+	+
Diagnostics	+	+	+
Is free	-	-	-

Table 1. Datadog, New relic and Nagios comparison.

* Have React based functionality to create custom views

All mentioned above solutions have all typical functionality for monitoring and logging, but in case of custom solution I describe in this work there are many lacks that are reason for refusal to use them:

- All these solutions do not provide functionality for creating custom views and monitor custom business data based on custom patterns (New relic offer near this

functionality with react background, but it is not acceptable for BBFinance IT team – it is .Net/Angular based)

- All above mentioned solutions do not have functionality to understand of Infrastructure of current system used in BBFinance
- All these solutions require any admin/programming skill that probably IT team does not have, my offered solution will use .Net
- There is no free solution with required capability set on market

5 Proposed solution

The proposed solution aim is to solve previously mentioned problems. Primary objectives are:

- Comprehensive solution that brings different tools together into one easy and fast to use package.
- Logs, Errors, Messages – have a relationship together, that can help developers to view entire picture and speed up troubleshooting time
- Track more places that can potentially help to prevent or troubleshoot existing issues
- If possible, provide enough information so troubleshooting issues by making direct requests to SQL database will be avoided.
- Improve alert system to be able to pro-actively detect issues or react quicker once they happen

Example of a monitoring home page:

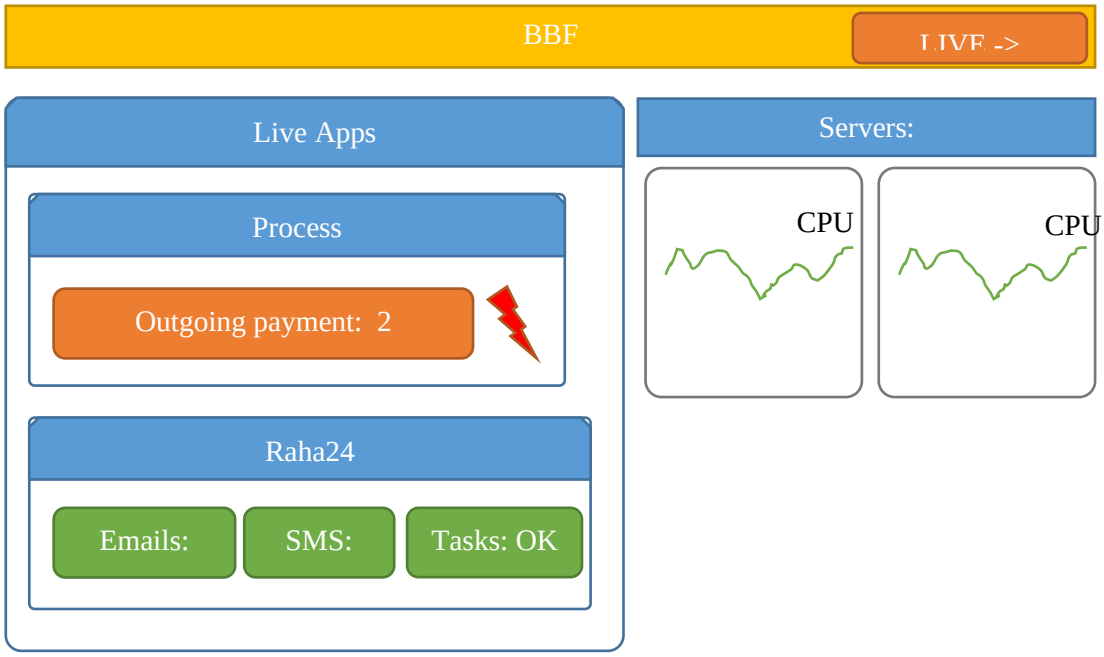


Figure 2. Monitoring home page example.

- Primary page is going to show, current state of servers, services, processes and applications, giving a quick overview over everything.
- Clicking on specific element is going to open that element with more details.

Example of monitoring Raha24 Outgoing Payments Module:

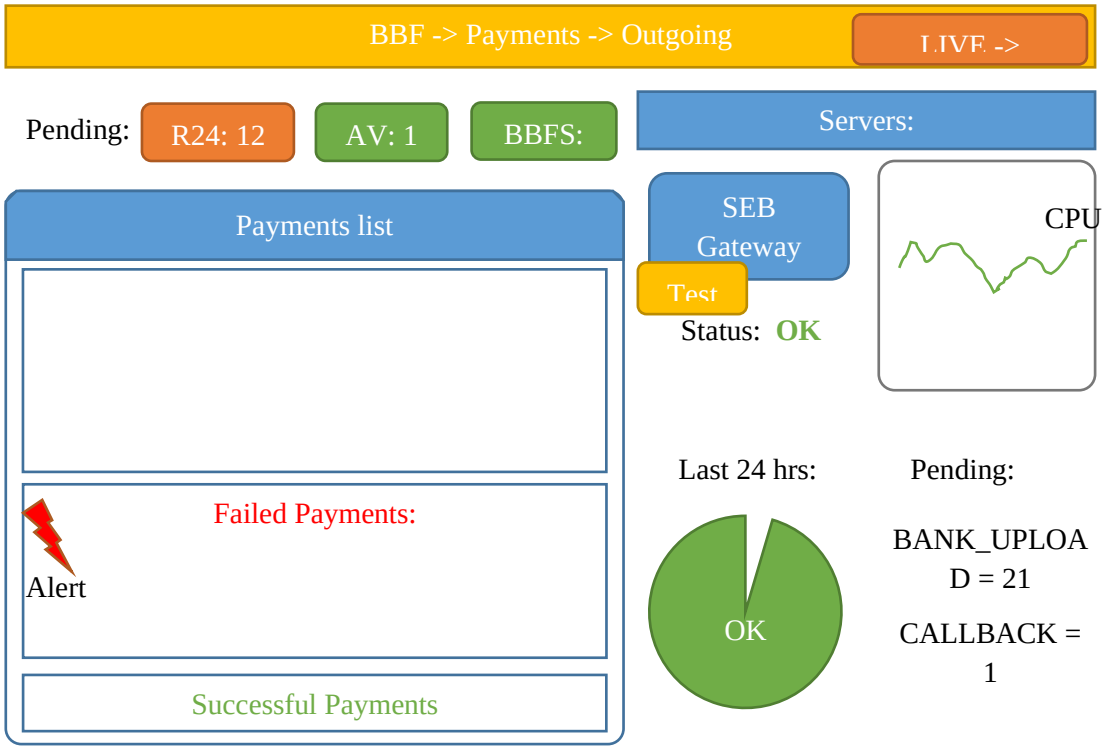


Figure 3. Example of monitoring Raha24 Outgoing Payments Module.

The main idea of this page is to give comprehensive and complete overview of the outgoing payments system. Any issues that are happening can be examined, external solutions tested.

This page:

- Shows current status of payments:
  - Payments in queue and their status
- Shows history of payments:
  - Those that have failed
  - Those that have succeeded
- Shows statistics:
  - Average time to perform the payment workflow, with a breakdown for each stage
  - Sum transferred
- Shows performance of the related servers (where payments are going to be used)
- Shows external APIs that are used with Outgoing Payments and ability to test their status (Diagnostics)

- Allows to configure alerts for specific events and situations.

To troubleshoot specific issue, you can click on a specific Payment and get comprehensive information about its status:

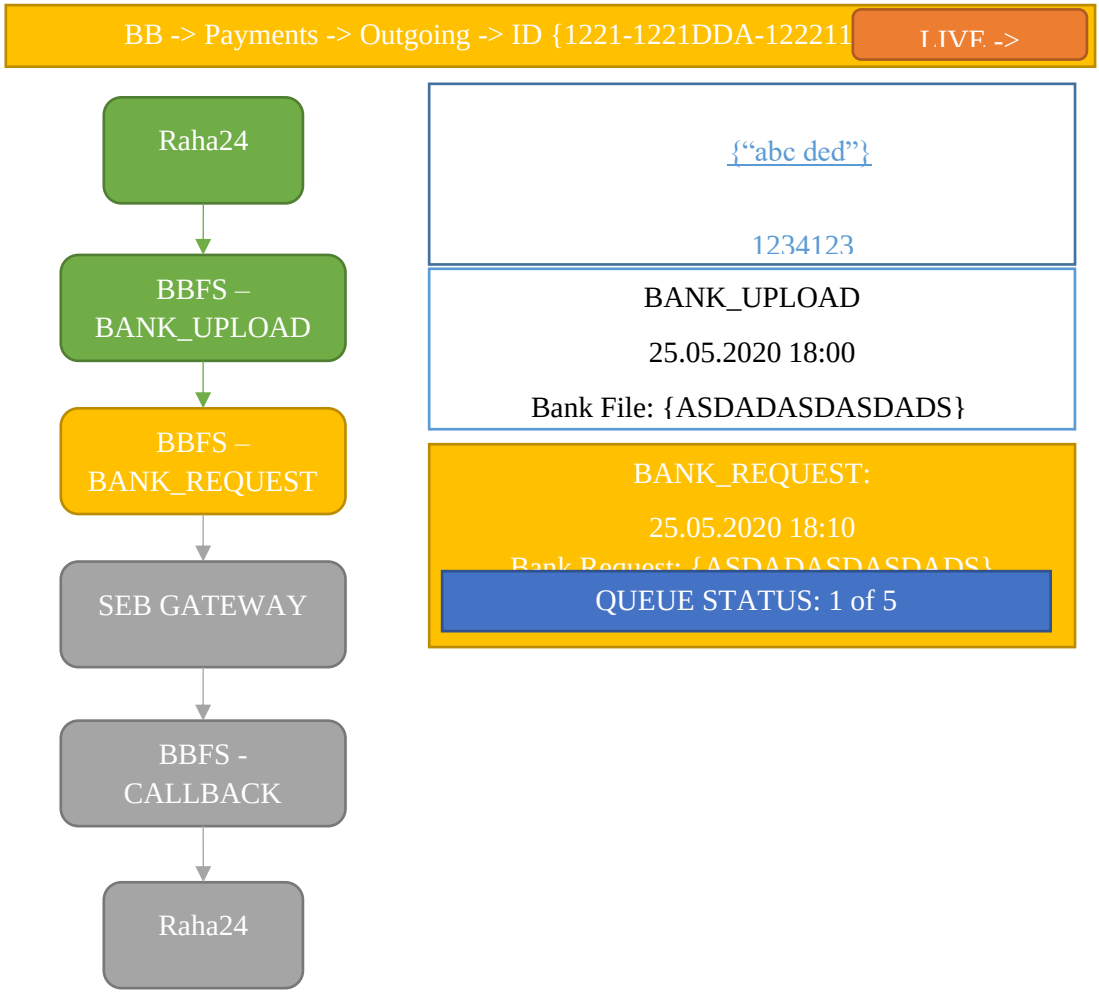


Figure 4. Example of Outgoing payment monitoring view.

The following page is going to show entire progress for the payment, starting with originator (Raha24, Avanss) and all the way back. The information for the request is retrieved and displayed.

Main features of the page:

- Shows current step where process is (yellow) and previous steps (green), where process was. Upcoming steps will be shown in gray color.
- Each step has additional information with payment details

- Some steps can contain clickable details (Person, LoanId) clicking on this link will open another window where it can be searched within logs or error messages.

6 High level design

Consists of two parts: Statistics and Diagnostics

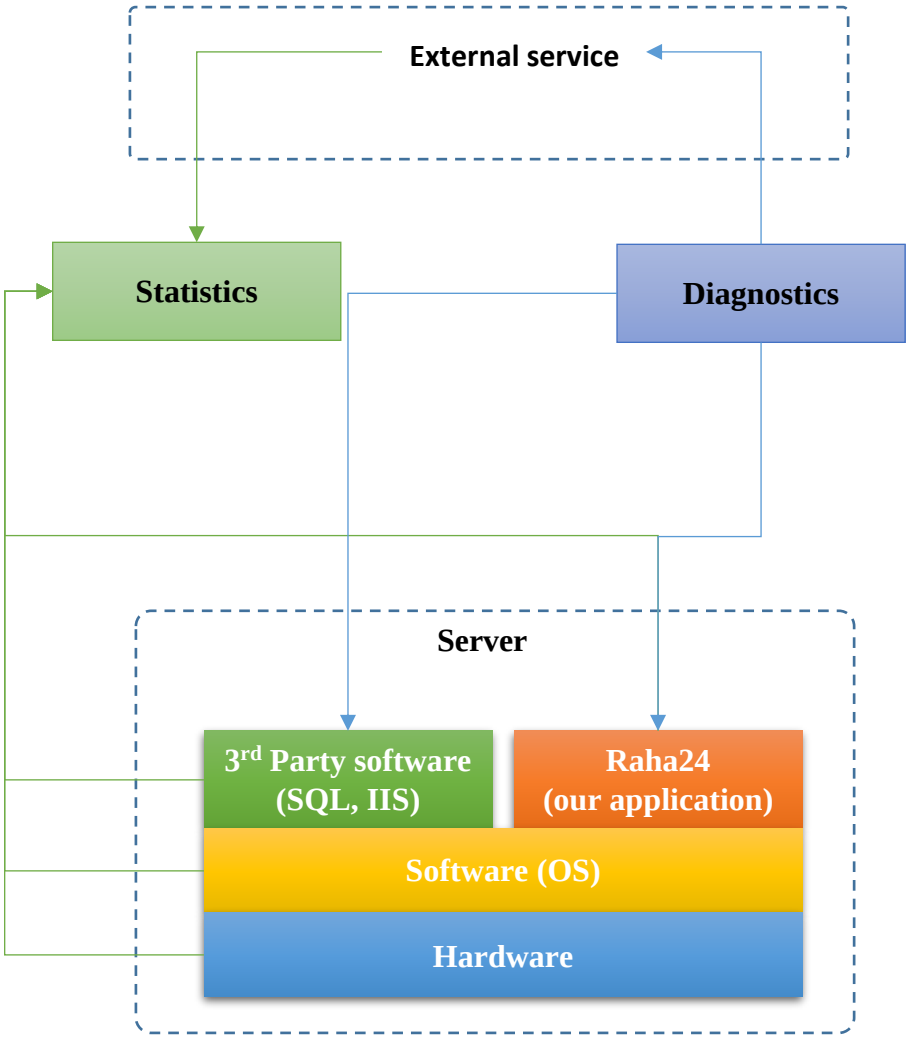


Figure 5. High level design.

6.1 Statistics

To monitor data, we need to gather statistics from Hardware and Software parts of the Server. Gathered statistics should have enough information to troubleshoot the issue, these should contain Id, ref code, exception message, input arguments, etc.

Things to gather:

Hardware

Metric	Tool	Description
Disk Space	Tool_Nagios	
CPU Usage	Tool_Nagios	
RAM Usage	Tool_Nagios	

Table 2. Existing hardware monitoring.

Software OS Level

Metric	Tool	Description
EventLogs	-	Event logs are generated when unexpected error happens. Right now, we don't have a tool or anything that can gather this information
Services	Tool_Nagios	List of running/not running services
Applications – Memory / CPU Usage	-	Right now we don't track any of this information, however it will be valuable for troubleshooting purposes

Table 3. Existing software OS level monitoring.

Software 3rd Party

Metric	Tool	Description
SQL Server – logs, locks, errors	-	SQL Server performance is essential, we had and still have some issues with deadlocks or timeouts.
IIS Server – logs, errors	-	IIS issues are not typical, but having good logs can help in troubleshooting IIS issues or figuring out if our server is under attack
Docker – logs, status	-	Deployment results, anything else related to that

Table 4. Existing software 3rd party monitoring.

Software – our applications

Metric	Tool	Description
App - Web status	Tool_Nagios	Status check to see if web application is offline
App - Certificate validity	Tool_Nagios	Checks validity of certificates if they have expired
App – Messages	Tool_GrayLog	Messages will go to GrayLog most of the time
App – Errors	Tool_GrayLog Tool_Sentry Local file system logs	Most errors go to Sentry, but some are still written to the local file or go to Event log. This has caused issues in the past, since we couldn't track this properly
App – Business Logic stats	-	This is something that can be used to monitor and show current activity of the application: Number of active users State of the cache – MB (elements) Currently running tasks Number of errors Number of received payments Number of items in queue Number of sent payments Auto accept process Lots more..

Table 5. Existing software our application monitoring.

6.2 Diagnostics

Diagnostics is used mostly for troubleshooting purposes, to validate if existing service is working as expected or not. This can be integrated into build process to perform automatically. The diagnostics can check our system or external systems.

Right now, the only diagnostic activity is performed by Tool_Nagios to test if our websites are offline.

The following things might need to be added to diagnostics:

Metric	Tool	Description
App - Web status	Tool_Nagios	Status check to see if web application is offline
App – Validate e-mail sending		Test to send an example e-mail
App – Validate test payment		Test to send a payment or at least test that API is replying something
App – Validate send SMS		Test that SMS sending is working
App – Validate Restful API		Test that restful API is working

Table 6. Required diagnostics.

Basically, anything that supposed to work that we can verify.

6.3 Implementation

Implementation can be modular. Whenever new development is made a diagnostics and statistics module is created that can be used later.

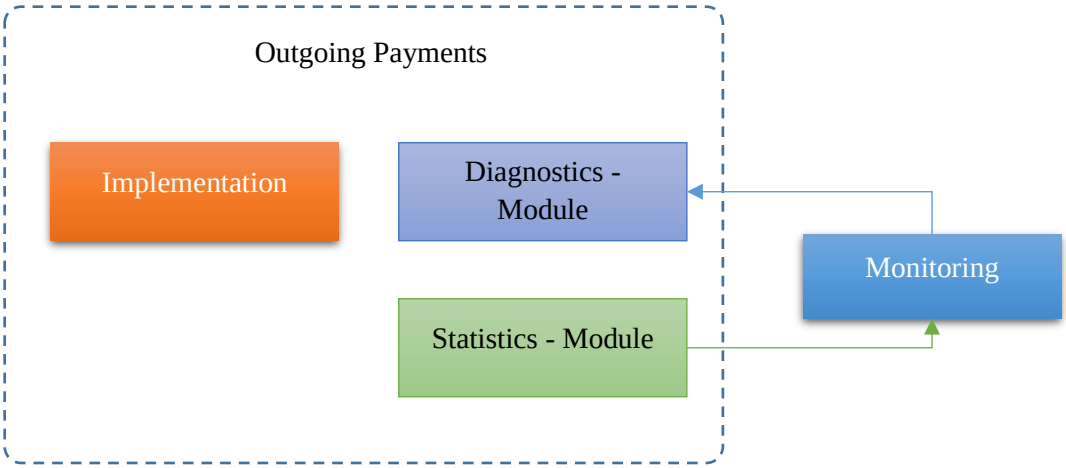


Figure 6. Outgoing payments module example.

Here, the original implementation is left unchanged, additional modules are added:

- Diagnostics module, will allow Monitoring to trigger diagnostics requests
- Statistics module, will gather statistics information and provide it to Monitoring module

6.3.1 Module representation

Each module is further broken down into parts, with additional information transferred:

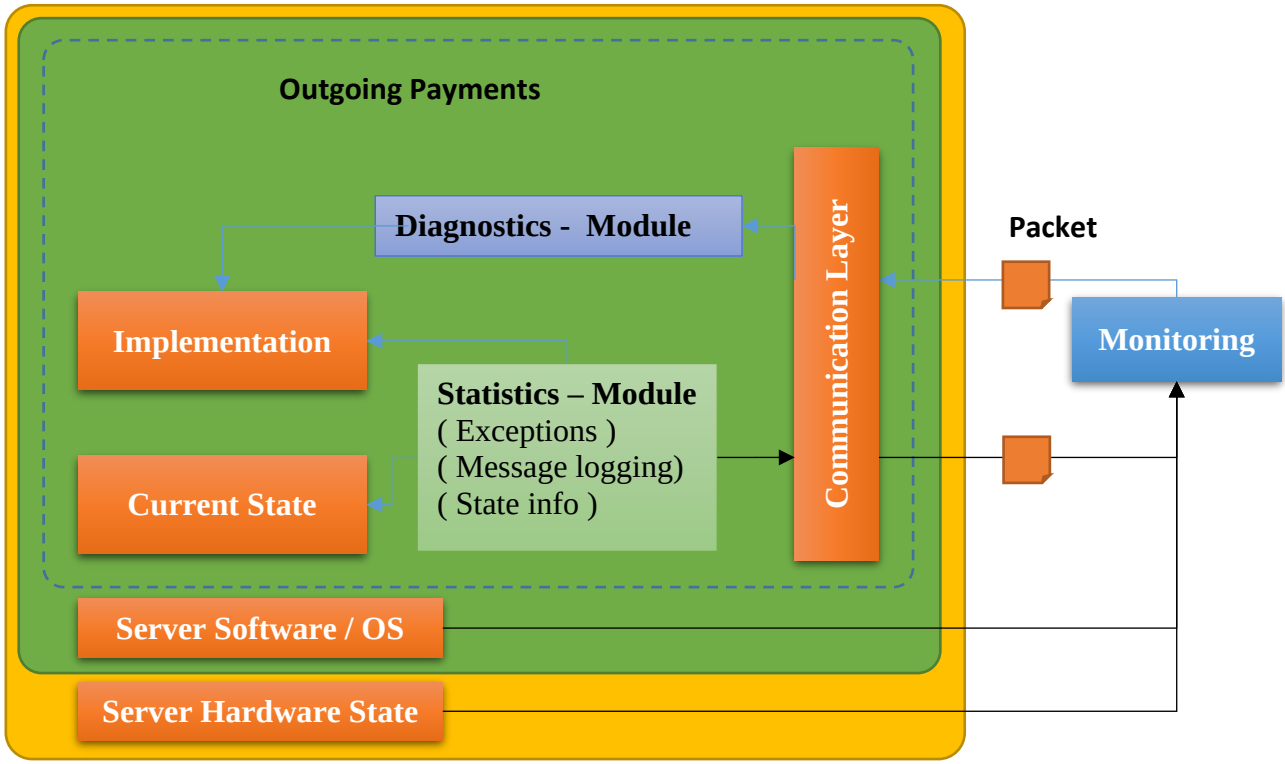


Figure 7. Module structure.

- Diagnostics is going to be called on demand to test the implementation. There are very few places where this is implemented.
- Statistics is going to be used to collect and send:
    - Exception messages or other errors. Right now, this is done with Tool_Sentry
    - Log messages. Right now, this is done with Tool_GrayLog
    - Send state information (if applicable). This is made with a custom solution in some places.

- Additionally, “OS State” and “Server Hardware State” information is going to be associated with statistics, this allows later to filter out statistics by hardware/OS software
- Communication Layer is used to send and receive packets from/to other modules:
    - When module is started, it’s going to notify the Monitoring that it’s ready to receive messages
    - When module is stopped, it’s going to notify the Monitoring that it’s no longer receiving messages
    - The communication is done through Monitoring Packets

6.3.2 Collected Module Metric Data

Each module collects data to one of the Collected Module Metric tables in Slave database. Collected data can come from different modules.

6.3.3 Fault classification

Whenever a failure occurs inside a module, we need to classify it and log it.

This is very important, as depending on the classification appropriate action can be taken. Correct classification notifies others about severity of situation and urgency of the issue, thus reducing impact of the fault on the systems.

Here is an example of a Fault structure:

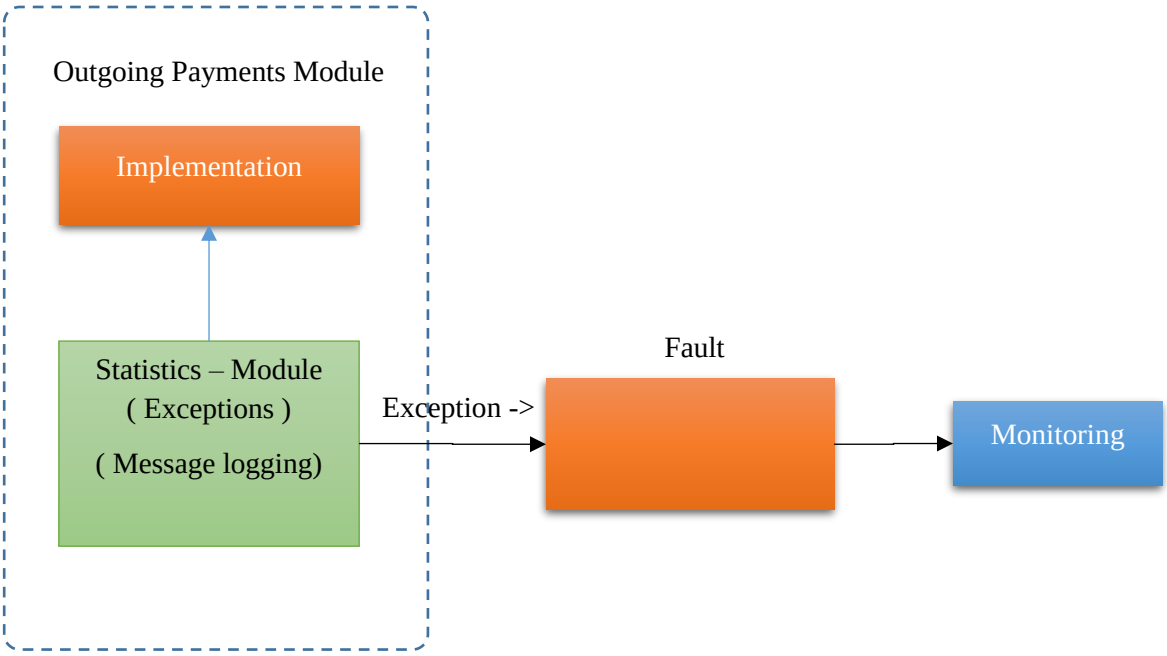


Figure 8. Fault structure.

Fault can contain the following details (it's going to be automatically associated with correct module Id when sent to Monitoring):

- Message – Error message
- Resolution – Indicates a possible resolution for this specific problem. For example, if there is missing App.config setting somewhere, explain what needs to be added and where. This is going to be very helpful when troubleshooting this issue in the future.
- Severity – (1..5) – Indicates importance of this issue. Anything above 3 should be taken into work and resolved, anything below may be ignored (for example temporary deployment errors, etc.)
- Exception Data – Additional exception data (Optional). It's recommended to include additional information that can be used to troubleshoot an issue. For example, if failure happened during a Loan application process, PersonID in the system can be included as well as loan application details.
- Stack Trace – Stack trace where error has happened (Optional)

Severity table:

Fault Severity	Action	Action
1	Ignore	Issue that is known to be a temporary situation and should be ignored
2	Ignore / Fix	Issue that can happen in some situations but may not be a problem that needs to be fixed
3	Fix	Issue that impacts a system and should be fixed.
4	Fix	High priority issue, impacts existing system and should be taken into priority.
5	Fix	Critical issue, should be solved ASAP

Table 7. Severity table.

Advantages:

- Having a fixed severity defined for every possible fault situation, can indicate exactly how important specific issue is.
- This can also be used to automatically create a Work Ticket or Task when fault happens, so that someone can start troubleshooting it right away.
- Having a clear and specific resolution text included with fault can cut time in the future when this issue happens again, especially if this issue is unfamiliar for the developers.

6.3.4 Hierarchical project structure

Applications and projects are created with hierarchical structure, the relationship is defined to have a clear association between elements

6.3.4.1 Projects

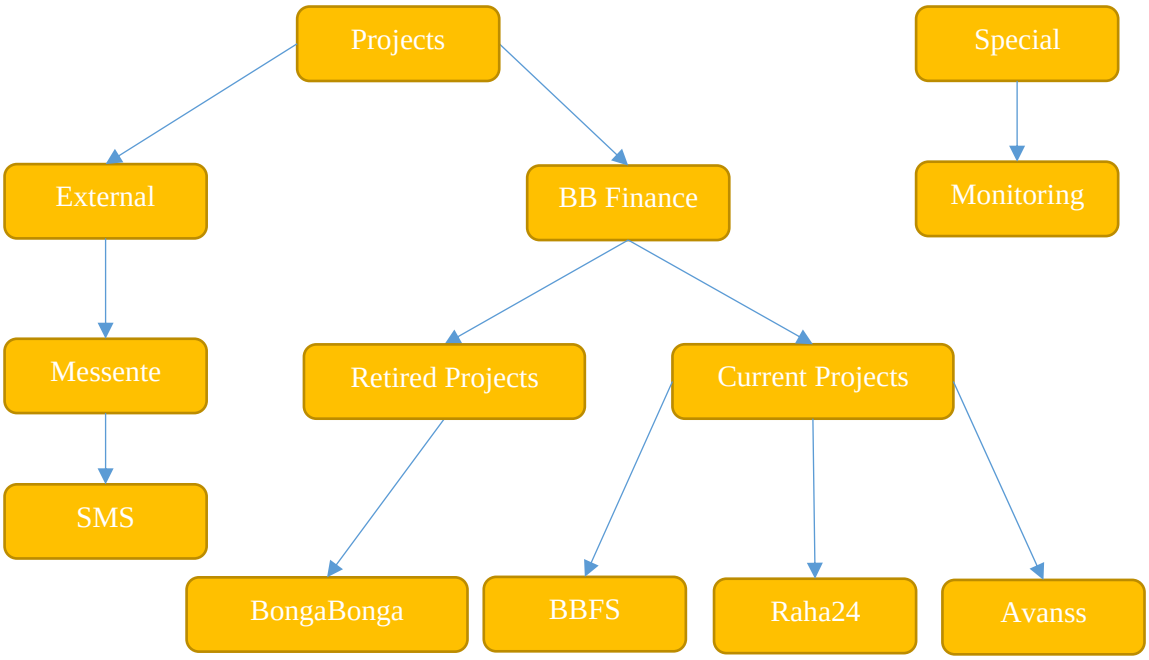


Figure 9. Projects structure.

The structure is divided into:

- Projects – Project that will be monitored and can be added by developers
- Special – Special project used by monitoring itself. It is divided also into Master and Slave

6.3.4.2 Project Unique identifier

Each project is uniquely defined by Id, Id of Module is created dynamically based on the hierarchical structure. It's possible to define Id manually for each module or use a random number. Module Id must be of 2 symbols.

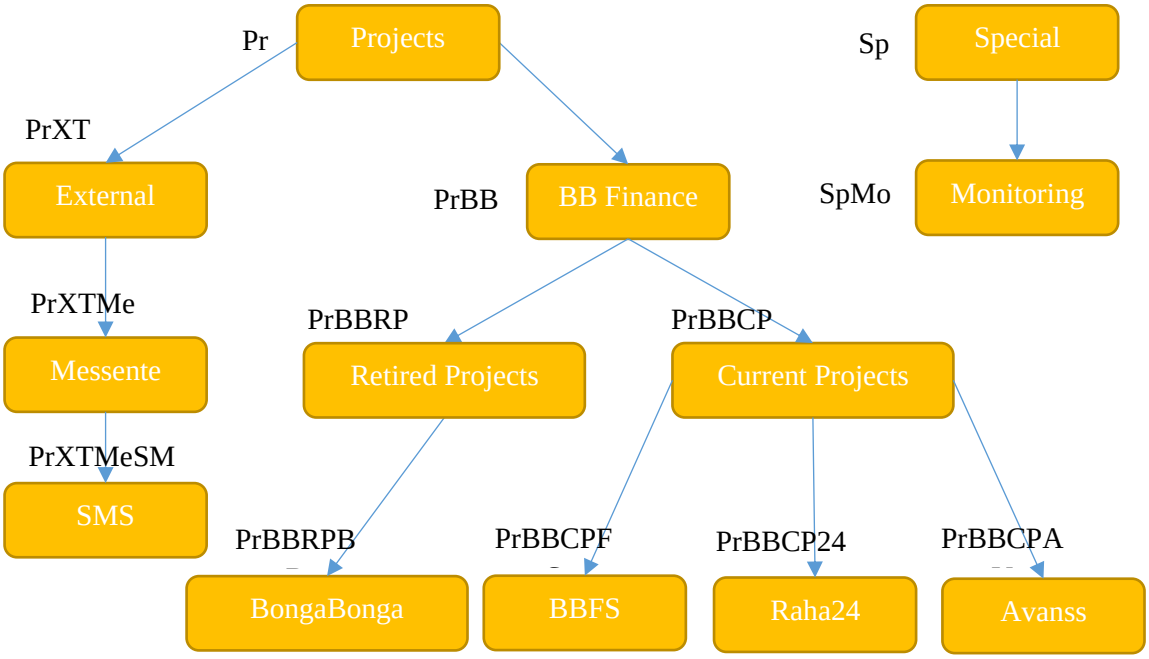


Figure 10. Project unique identifiers.

For example Raha24 project will have a unique key of 'PrBBCP24' if we want to filter out all metrics collected for all Current Projects we can use a SQL filter statement WHERE like 'PrBBCP%' which should be fairly efficient to do if there is an appropriate index created.

6.3.4.3 Project Module Containers

Project module containers is used to define how project modules are hosted (i.e. where are they running on OS Level) and how they communicate with each other on the project level. These need to be defined first before any modules are defined.

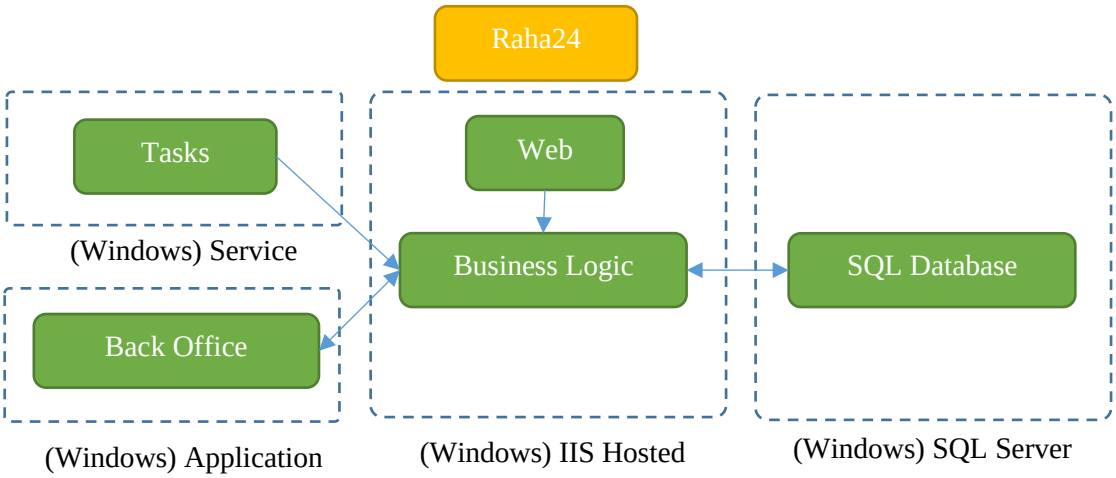


Figure 11. Project module containers.

6.3.4.4 Project Modules

Modules are defined as hierarchical structure to distinguish unique functionality that we want to monitor.

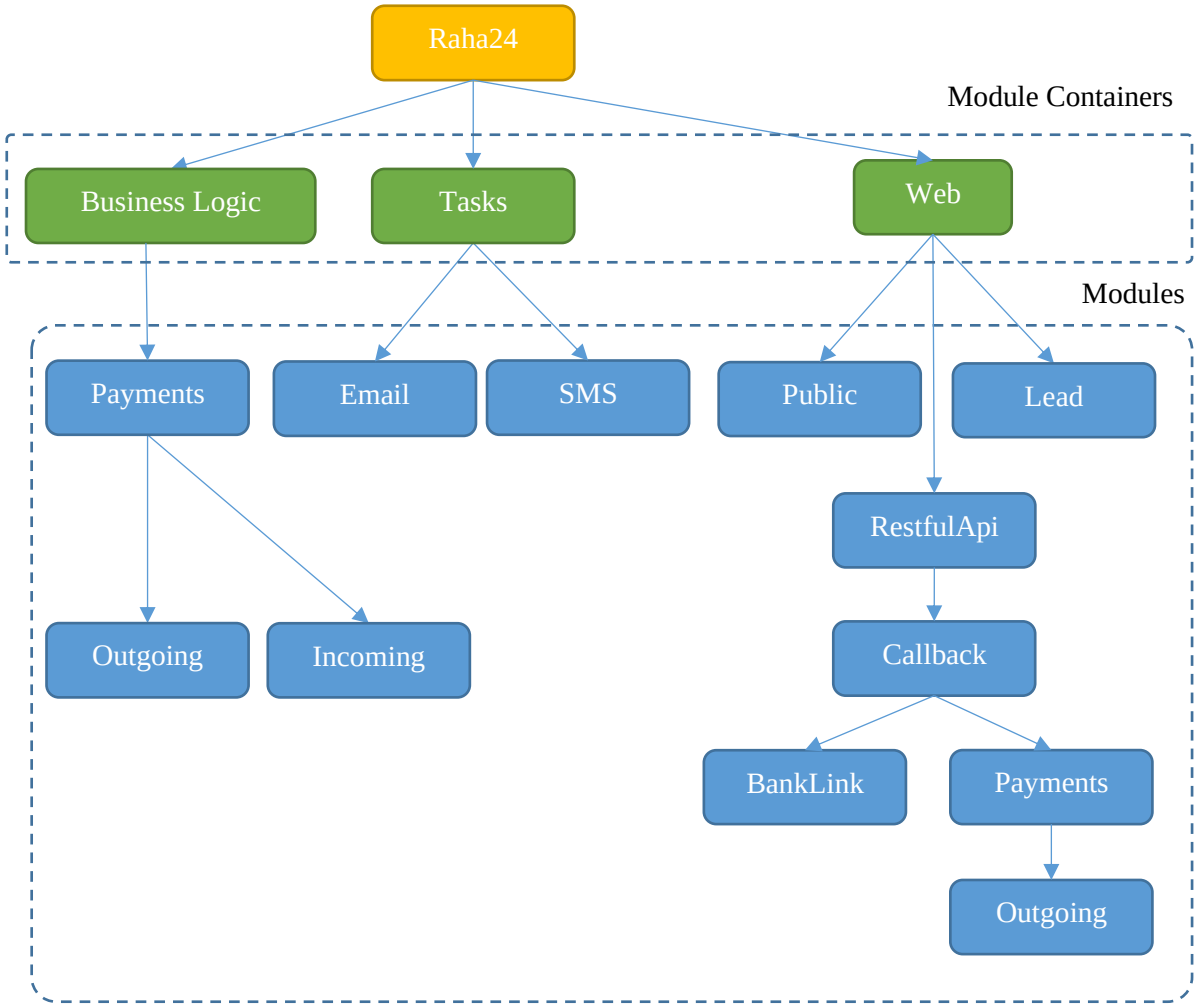


Figure 12. Project modules.

6.3.4.5 Project Module Unique identifier

Each Module need to define a unique identifier that will be used. This Id will be unique and should not be shared with others. The Id starts with project name and is delimited, the remaining parts are max 2 chars long.

For example:

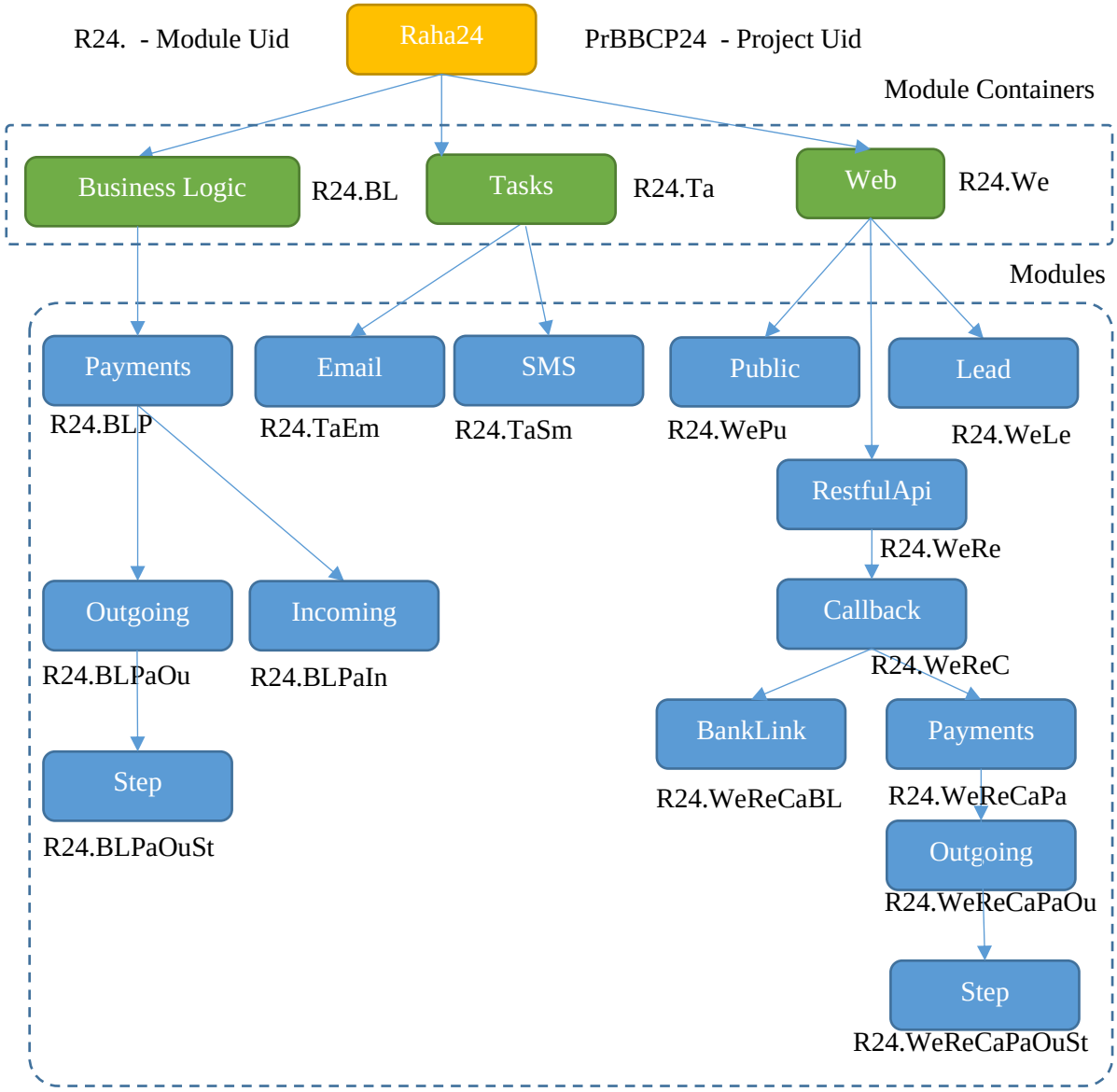


Figure 13. Project module unique identifier.

6.3.4.6 Project Module associations

Each module can be configured to show relationship with other modules. In this example, R24 OutgoingPayments sends a request to BBFS that is going to send a callback to R24 RestfulAPI Callback

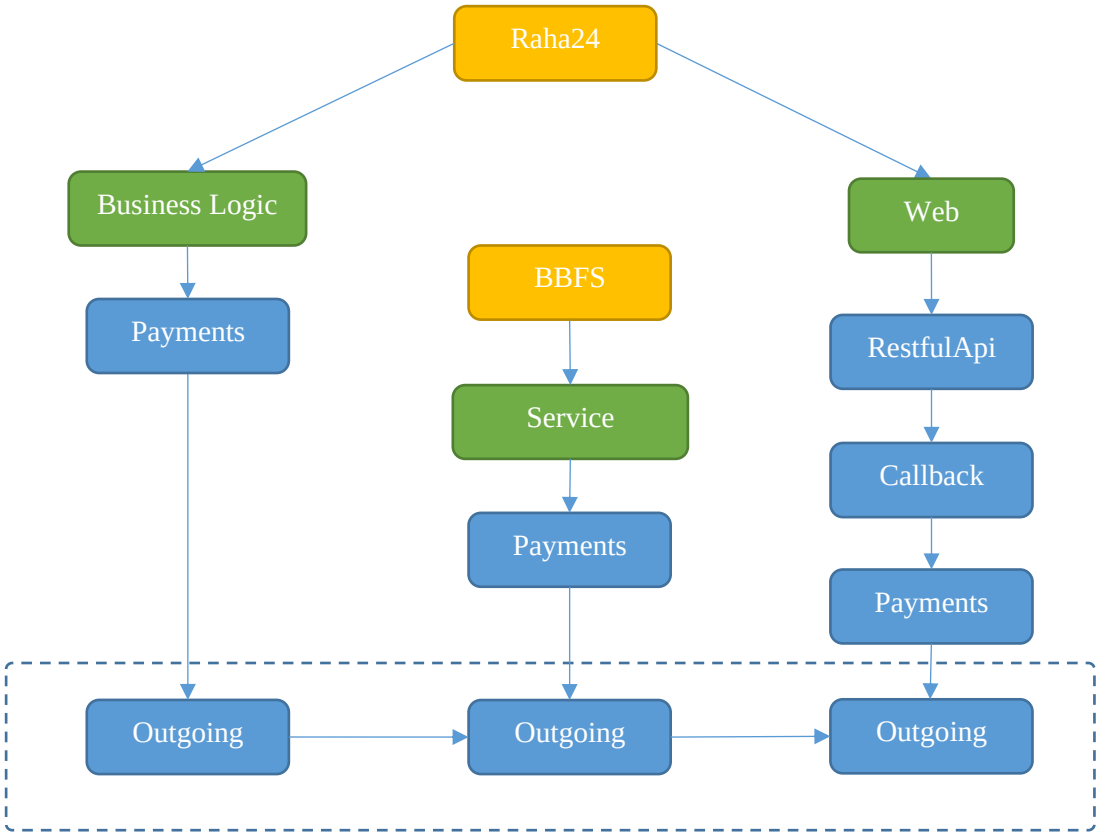


Figure 14. Project module associations.

6.3.4.7 Processes

Processes are used to combine or group multiple modules from different projects together into one.

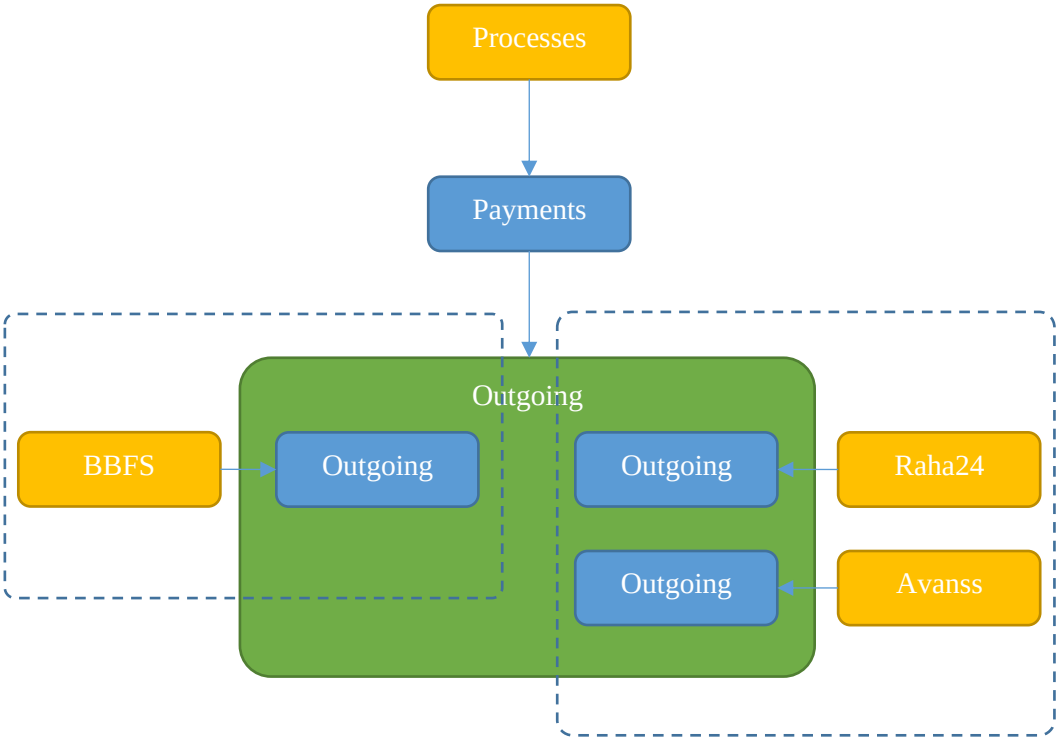


Figure 15. Processes.

6.3.4.8 Environments

Environments can be created to group Projects or other modules together but separate them depending on requirements. Environment is the source that will receive monitoring data.

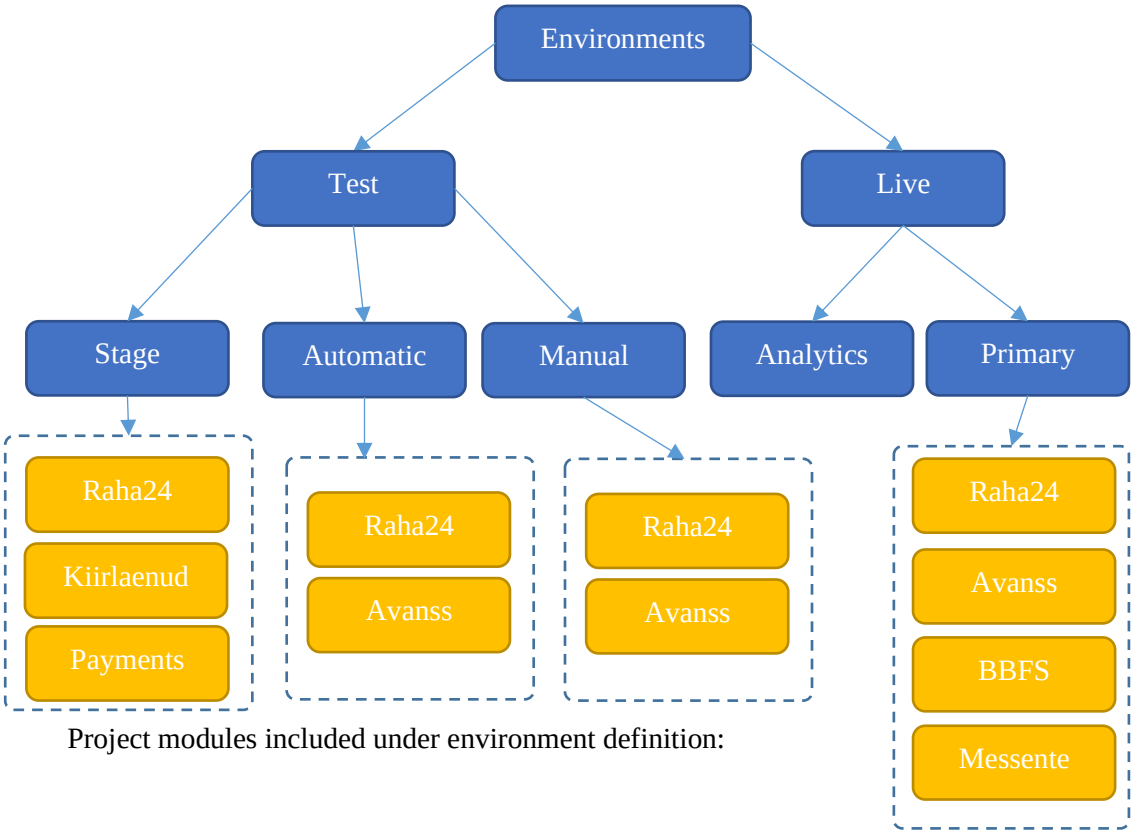


Figure 16. Environments.

Each environment should be independent of another: different databases, preferably different servers, but technically there is no restriction and multiple environments can be running from the same machine and share same SQL Server.

This is very similar how current test server has 3 test environments, but it's defined very loosely.

Because environments are separated for Monitoring and Diagnostics, all errors and messages are going to be already filtered by the environment, so it is easier to understand when troubleshooting.

Environments can be potentially controlled in the future, such as mass actions can be done for all the projects within that environment:

- Restore database
- Start/Stop all services
- Deploy latest version

6.3.4.9 Environment topography

Once Project Module associations is configured for each project and associations are defined, the entire environment topography can be displayed along with information gathered about current location where software is running and it's state:

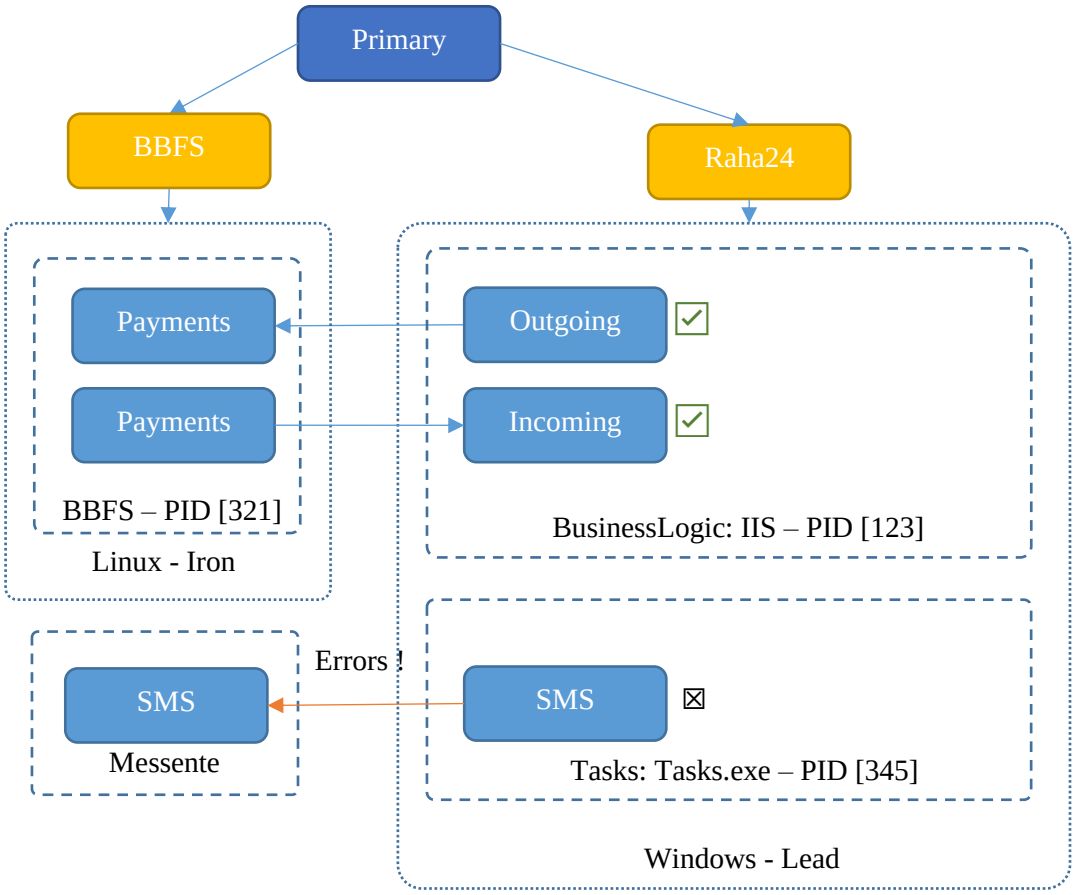


Figure 17. Environment topography.

6.4 Settings configuration

To improve and simplify configuration of Environments or Projects, it will be possible to set specific settings depending on the requirement:

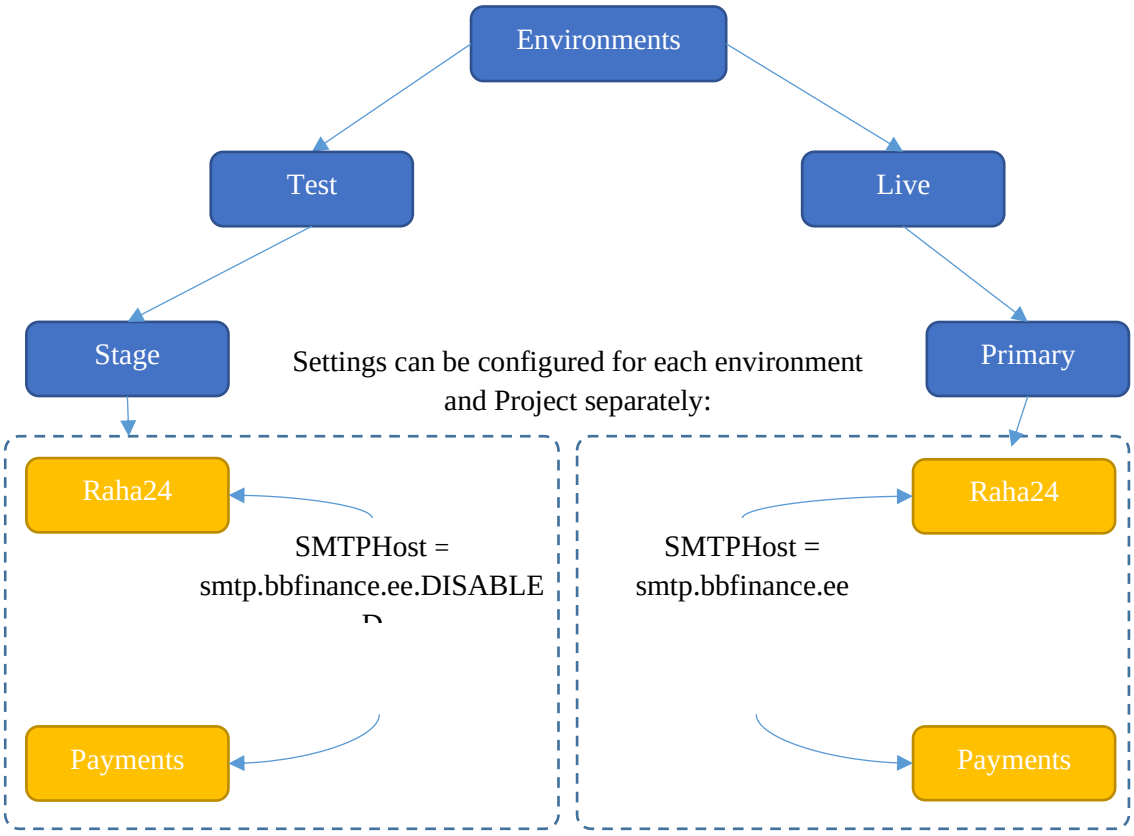


Figure 18. Settings configuration.

The configuration settings are applied in this order:

- App.config = default value
- App.CobaltDeploy = Overrides previous value during deployment
- Monitoring – Environment – Override previous value with global environment value
- Monitoring – Project – Override previous value with project specific value
- Monitoring – Server – Override previous value with server specific value

This way, if there are tools or endpoints that are used by many projects, these can be defined once per environment and automatically re-used by all the projects within that environment.

Compared to previous system for Raha24:

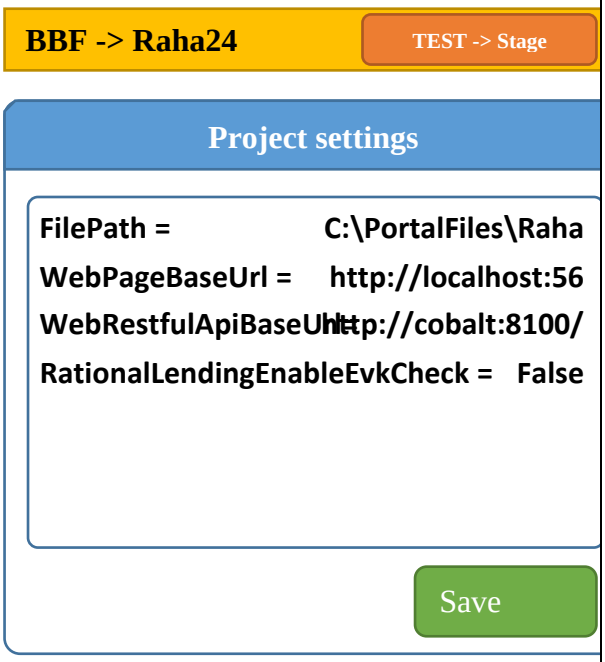
Before

```
<applicationSettings>
   <BusinessService.Properties.Settings>
     <setting name="DOMAIN" serializeAs="String">
       <value />
     </setting>
   </BusinessService.Properties.Settings>

<Nortal.Utilities.MessenteSms.MessenteConnectionSetti
ngs>
     <setting name="SuppressSmsSending"
serializeAs="String">
       <value>False</value>
     </setting>
     <setting name="UserName" serializeAs="String">
       <value>12345USERNAME</value>
     </setting>
     <setting name="Password" serializeAs="String">
       <value>12345PASSWORD</value>
     </setting>

</Nortal.Utilities.MessenteSms.MessenteConnectionSett
ings>
  </applicationSettings>
  <appSettings file="_localConfig\appSettings.config">
    <add key="InkassoKeskusUsername"
value="12345USERNAME" />
    <add key="InkassoKeskusPassword"
value="12345PASSWORD" />
    <add key="IMakseServiceFeePercent" value="0.01"/>
    <add key="IMakseMinServiceFeeEur" value="0.2"/>
    <add key="SentryDSN" value="" />
    <add key="ClientSettingsProvider.ServiceUri"
value="" />
    <!-- Supress SMS -->
    <add key="SuppressSMS" value="True" />
    <!-- Amount of months after loan end that user is still
regular customer -->
    <add
key="RegularCustomerWhenLoanEndedXMonthsAgo"
value="3" />
    <add key="RespLendingPensionCutoffDays"
value="45" />
    <add key="RespLendingMaxPaymentRatio"
value="88" />
    <add
key="RespLendingManualConfirmMonthlyLimit"
value="500" />
    <add key="RespLendingManualCheckCreditLimit"
value="500" />
    <add key="RespLendingNoEvkCreditLimit"
value="500" />
    <add key="RespLendingEvkTimeoutDays"
value="14" />
    <add key="RespLendingEvkMinimum" value="350"
/>
    <add key="AllowCreditLimitIncrease" value="false"
/>
    <!-- Files -->
    <add key="FilePath" value="C:\PortalFiles\Raha24\"
/>
    <!-- Instantor -->
    <add key="InstantorEnvironment" value="test" />
    <add key="InstantorSourceName"
value="12345USERNAME" />
    <add key="InstantorApiKey" value="12345APIKEY"
/>
```

After

```
<appSettings file="_localConfig\appSettings.config">
    <add key="MonitoringRouterEndpointUrl"
value="http://router.monitoring" />
<add key="ProjectId" value="Project1234556"/>
</appSettings>
```

- NOTE: All other settings are declared on the monitoring side for specific project:



- Or module level:

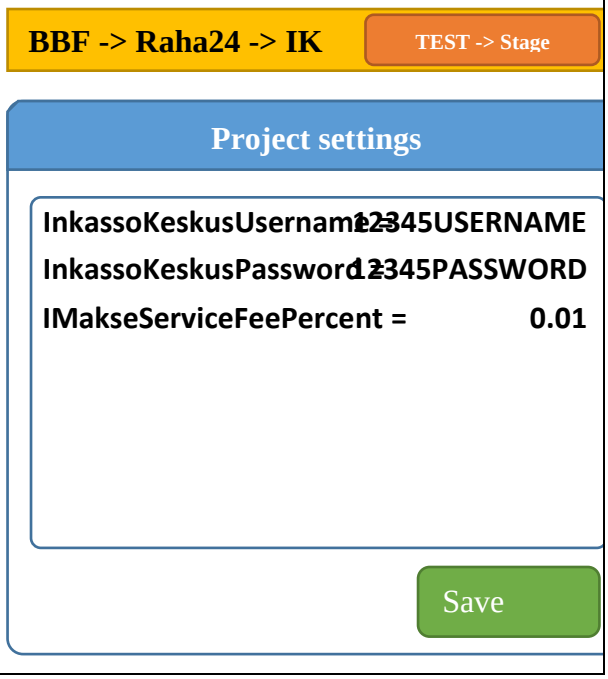


Table 8. Settings configuration comparison.

How this is beneficial:

- For example, having an e-mail whitelist defined for the entire environment. This way adding new user to the white list doesn't require any code changes and is almost instant, may not even require a service restart
- Disable or override configurations on the fly, if there is a need to Turn Off specific feature or change time, it can be done almost on the fly without re-deployment
- One single place for all the settings, simplifies maintenance as it's a single source of truth. All existing config values can be shown as well so that it's always going to show current valid value.
- In theory, since we know exactly where specific Project is running, the settings can be generated automatically and wouldn't require any configuration changes to be done at all.

For example: setting for e-mail sending SMTP host, can be configured with different values for different environments. It's going to be picked automatically by all projects in that environment.

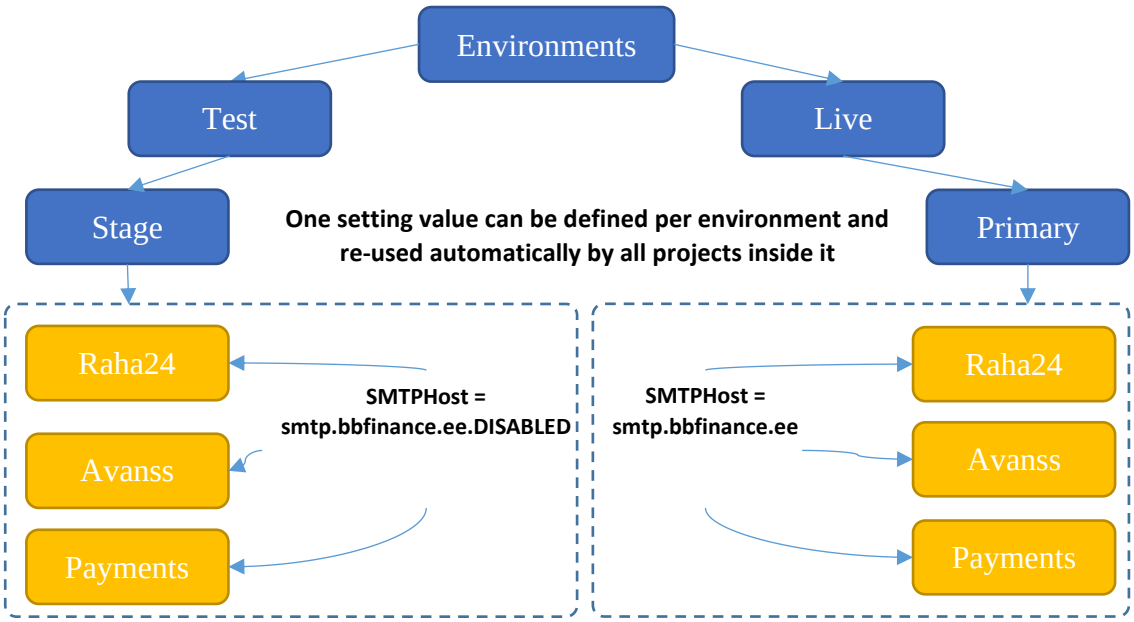


Figure 19. Settings per environments

Example for the UI:

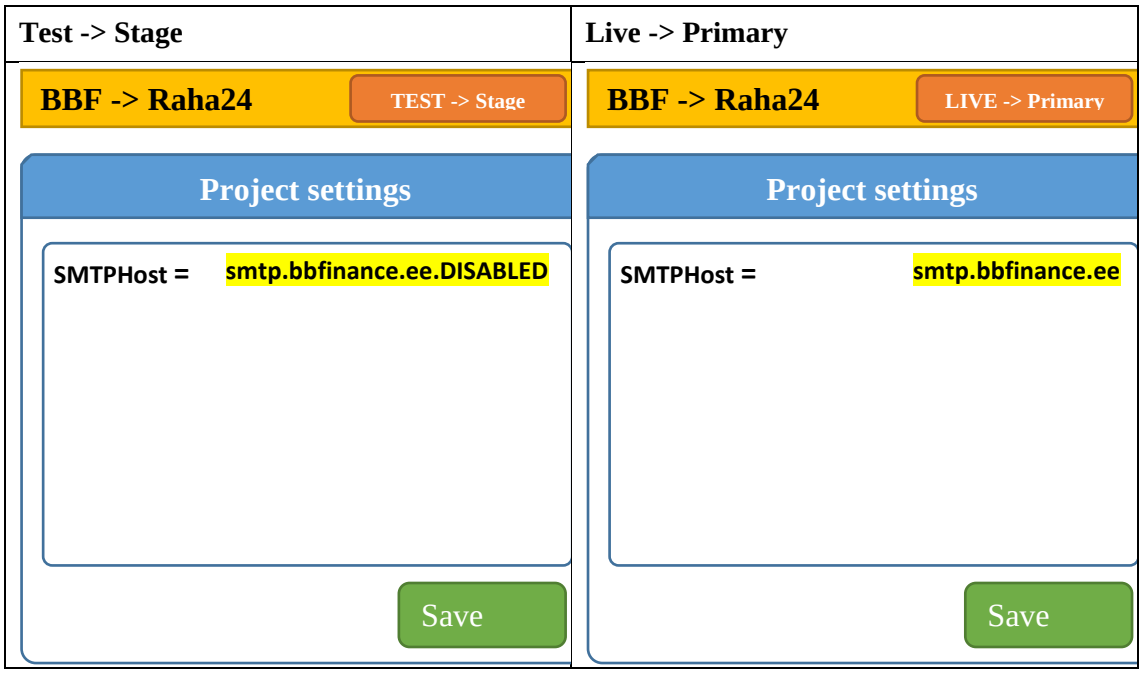


Figure 20. Settings per environments UI example.

Unified Project communication structure

To simplify communication between different project, the following structure is proposed. Monitoring router can be used as P2P mediator that either routes messages between projects or establishes communication between them. This is similar to what we have done already with Integrator / Data Integrator, but this solution is more unified and project agnostic. Basically, it is endpointless.

The advantages:

- Single endpoint URL to configure against – only monitoring endpoint is needed, no other endpoints are needed
- The communication API and packet format are simplified and standardized, so all messages are going to have the same structure
- All messages and communicates within environment are contained within that environment and can be logged or saved for troubleshooting.
- Gather statistics per module – how many requests were received, failed, etc
- Each module can expose a set of API that it supports. This allows module API to be checked and verified without making any requests.
- Project modules can be checked if they are operational, as they are going to signal when they're online or offline. This means we can see which projects are online/offline in real-time.

Disadvantages:

- Single point of failure, can be mitigated by adding a load balancer.
- Potential performance impact if Router is heavy and slows down packets.

Example (project level):

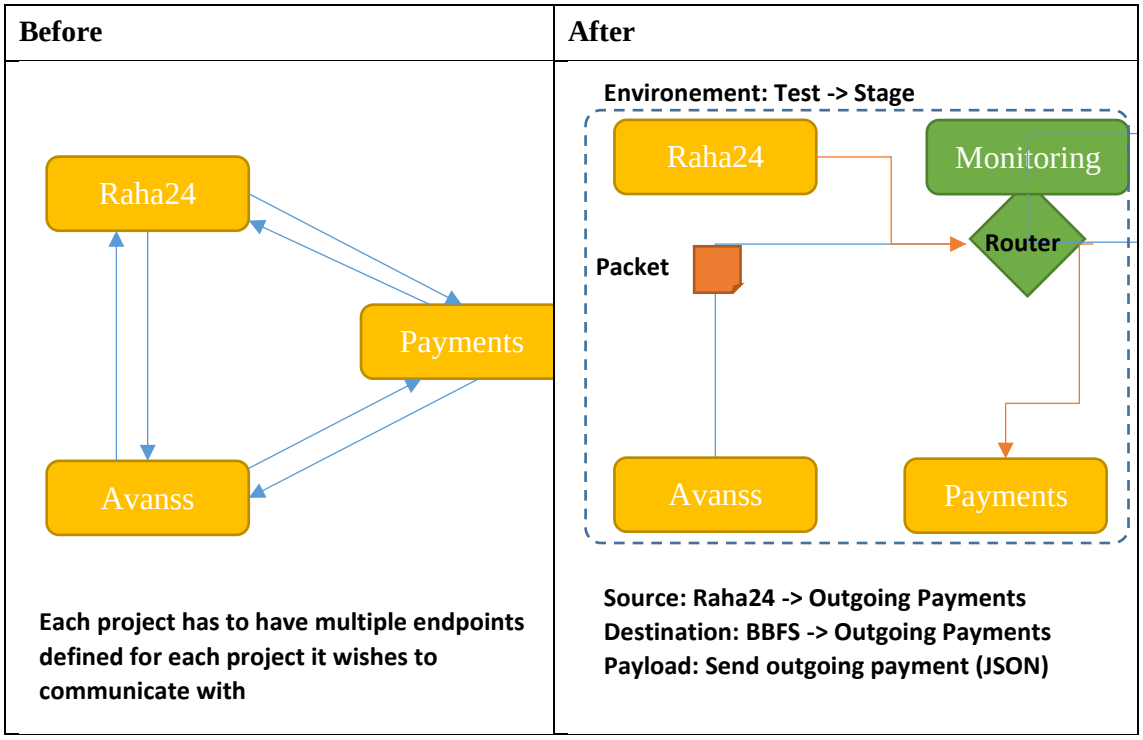


Figure 21. Communication structure.

6.4.1 Monitoring Packet

To communicate between projects a monitoring packet is used, the contents of the packet are:

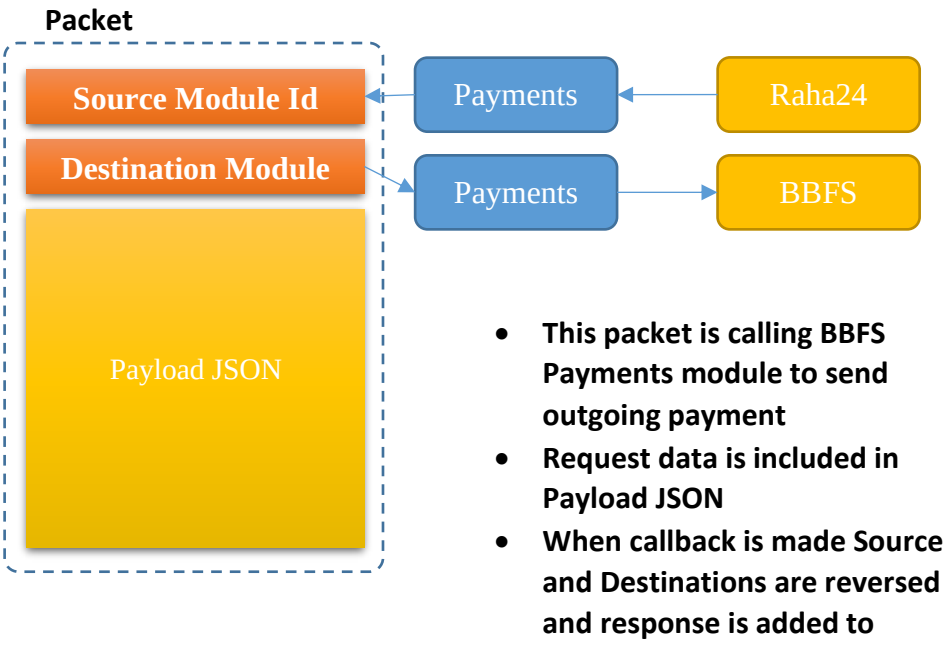


Figure 22. Packet structure.

Packet will be sent out to router, since Router knows Endpoint Urls of all the projects it will be able to route this message to the right Project, which will then take and process the message and trigger a callback.

Since Monitoring knows what modules are on-line, any packets that haven't been delivered will be known and reported.

6.5 Implementation in detail

When module is used to receive or sent communication it can be in different communication states. Modules that are not used for communication do not have exposed methods.

Module communication status:

- Unknown – Module is unknown state (typically it hasn't been started)
- Starting - Module is started but is waiting for dependency to become enabled (dependency need to be in either Started or Starting status)
- Started – Module has started and is ready to receive communication packets
- Failed – Failure occurred within a module and it is unable to communicate
- Stopped – Module has stopped and cannot receive communication packets at the moment

Every change in the module status will be sent out to Monitoring Agent, so it's always going to know accurate and up-to-date state of every module in the environment. Occasionally Agent is going to send ping requests to verify current status of the module and if it's still Online.

As defined in Project Module associations dependent modules are going to be started and stopped automatically if their dependency has been started or stopped.

For example, if Project BBFS is started, the following is going to happen:

1. (R24 -> Incoming Payments) [Stopped] – (BBFS -> Incoming Payments) [Stopped]
2. (R24 -> Incoming Payments) [Stopped] – (BBFS -> Incoming Payments) [Starting]
3. (R24 -> Incoming Payments) [Starting] – (BBFS -> Incoming Payments) [Starting]
4. (R24 -> Incoming Payments) [Started] – (BBFS -> Incoming Payments) [Started]

For example, if Project BBFS is stopped, the following is going to happen:

5. (R24 -> Incoming Payments) [Started] – (BBFS -> Incoming Payments) [Started]
6. (R24 -> Incoming Payments) [Started] – (BBFS -> Incoming Payments) [Stopped]
7. (R24 -> Incoming Payments) [Stopped] – (BBFS -> Incoming Payments) [Stopped]

Modules that have circular dependency can be started even if another module is offline, but they will only send requests if dependent module is online (Starting status).

6.6 Testing improvements

For testing, separating environments are going to be a key. By separating environments and creating them independently, it's going to allow us to gather information from all the defined projects in that environment and nothing else.

A typical testing scenario can look like that:

8. When starting testing, correct version will be deployed (version under test)
9. Database will be restored with clean mocked data
10. Monitoring will reset collected data state for that environment
11. Testing will be performed (automatic or manual)
12. Testing will be completed, data generated by the monitoring will be presented

13. Potentially, results can be saved and compared against previous testing runs (logs, errors, performance)

6.7 Monitoring Project structure

The following describes structure of monitoring project itself.

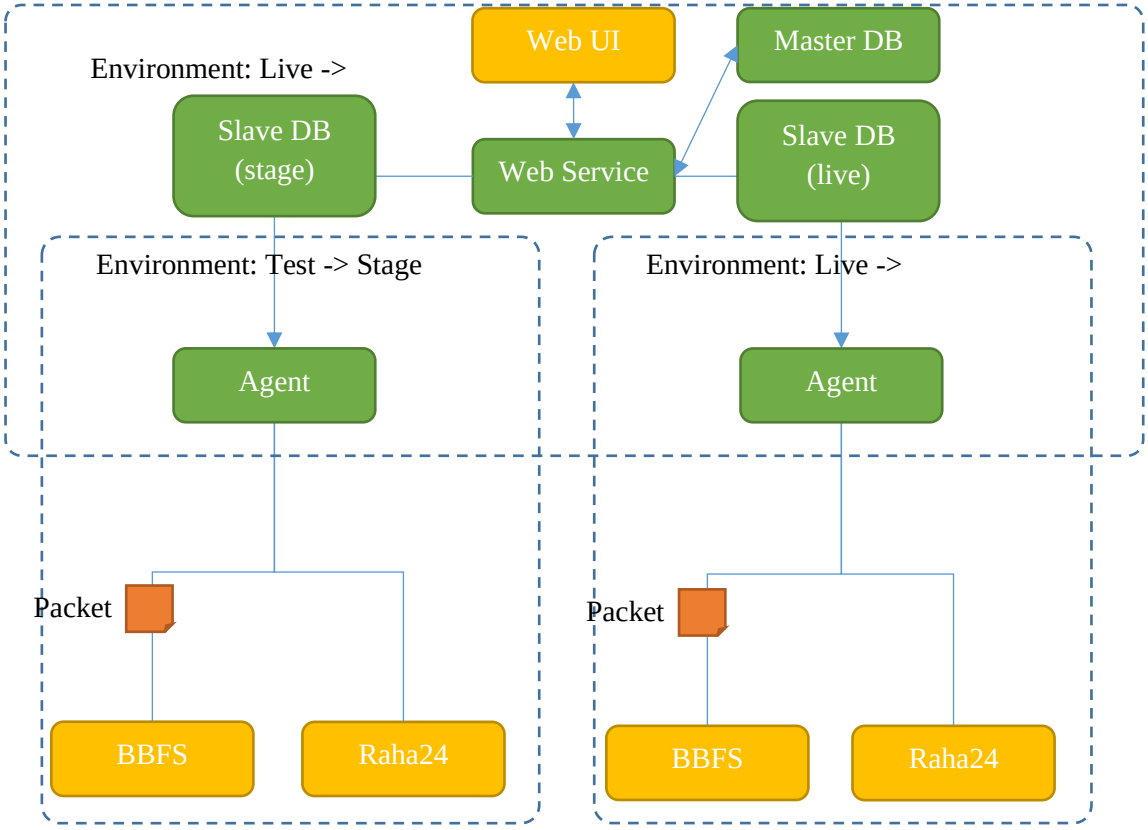


Figure 23. Monitoring project structure.

Monitoring is part of its own custom Environment “Monitoring”, this is used to make it unique and independent of other environments. The project itself consist of:

Stage	**Description**
Web UI	Web project that can be accessed through the browser and can be used for monitoring other environments.
Monitoring Web Service	Business logic, that is going to connect to other Monitoring database instances to retrieve values or perform tasks (diagnostics)
Master DB	Is going to be used to keep configuration settings or other information that are required to manage other monitoring

	environments. Overall, it's going to be very light and doesn't need to have a lot of data in the first place. Most data is stored on the Environment Slave DB itself.
Slave DB*	One or many databases created for each environment. Each environment is going to have a separate database where their data is stored. Database are created dynamically through the UI with a script.

Table 9. Monitoring project structure.

Each Environment, contains the following:

Stage	**Description**
Projects that will be monitored (with Modules)	
Monitoring Router	Will be used to exchange communication between different projects.
Monitoring Agent	Is used to receive or gather information (hardware/software) and write this information to the Monitoring DB. Everything that goes to Slave DB goes through the Agent first.
Monitoring Packet	Is a packet with information that is exchanged between different modules.

Table 10. Monitoring project environment structure.

6.7.1 SQL Table structure

As mentioned above, there are database for Master and Slave environments.

Slave environments collected data from Modules, but also keep association with different places, so that it can be easily analyzed in the future.

6.8 Used Tools

Tool	**Description**
Nagios	https://nagios.primendcloud.com/nagios/ Tool to monitor hardware infrastructure. Hosted by Primend
GrayLog	Collects messages from our applications
Sentry	https://sentry.io/organizations/bb-finance-group-ou

	Collects error messages from our applications
Datadog	Monitoring and logging platform
New Relic	New Relic

Table 11. Used tools.

7 Practical part

Practical solution is monitoring one of the business processes called “Outgoing payments”, this means when loan is created customer need to receive his loan as money through bank, this is done automatically using different third-party bank’s APIs.

Our WEB Api service PaymentsApi consists of many queues, through them data received from portals is moving step by step till payment is successfully paid out or failed. This is quite complicated process, what is working asynchronously, and that’s why it’s needed to be monitored for any delays and errors. Practical solution will monitor some most important patterns and make statistics based on different sets of statuses from different queues.

New view with monitoring and statistics data will be included into existing Backoffice system written in Angular.

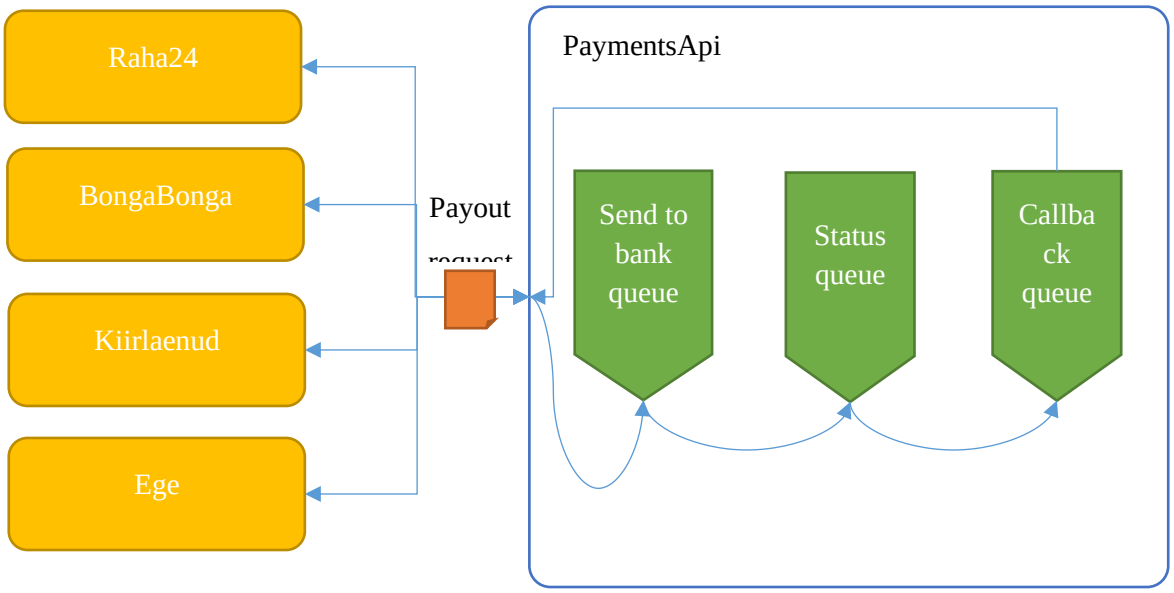


Figure 24. Practical part system to monitor structure.

7.1 Technologies stack

Used part	Technology
Service	.NET Core 2.2 GraphQL
View (Front-End)	Angular 6.1.7 Angular-redux 9.0.1 Ag-grid-angular 19.1.1 rxjs

Figure 25. Practical part technologies stack.

7.2 Data to be analyzed

To create analyze and make conclusions, all queues and data should be taken into account.

Analyzing and statistics result are created mainly on statuses and their combinations on entities in related tables of database

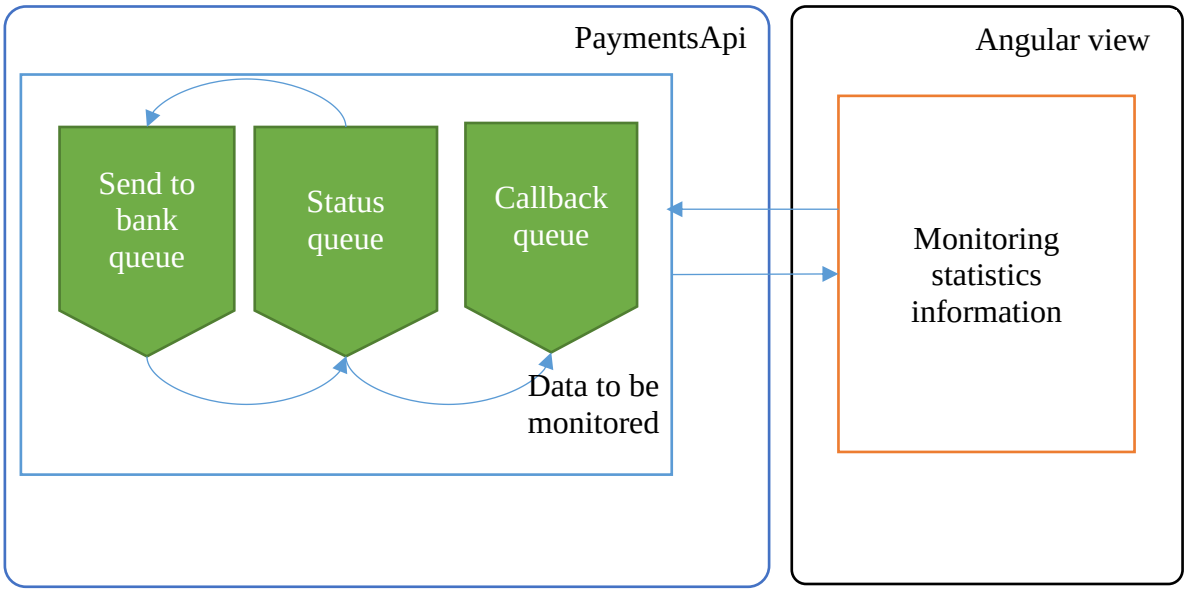


Figure 26. Practical part data to be analysed.

Next statistics is taken:

Statistics name	**Statistics description**
Upload pending	Number of outgoing payments what are taken into processing and waiting to be paid out `(a => a.Queue.Status != QueueStatus.Failed && a.Queue.Status != QueueStatus.Done)`
Upload sent	Number of outgoing payments sent to bank API for pay out `(a => a.Queue.Status == QueueStatus.Done)`
Upload fail	Number of outgoing payments failed to be sent to bank API `(a => a.Queue.Status == QueueStatus.Failed)`
Status request pending	Number of outgoing payments, what status requesting is in progress `(a => a.OutgoingPaymentStatuses.Any(s => s.Queue.Status != QueueStatus.Failed && s.Queue.Status != QueueStatus.Done)`
Status request success	Number of outgoing payments, what status requesting successfully finished `(a => a.SourceCallbackStatus == SourceCallbackStatus.Success)`
Status request fail	Number of outgoing payments, what status requesting failed `(a => a.SourceCallbackStatus == SourceCallbackStatus.Fail && a.OutgoingPaymentStatuses.Any())`
Callback pending	Number of outgoing payments, what callback is in progress `(a => a.OutgoingPaymentCallbacks.Any(s => s.Queue.Status != QueueStatus.Failed && s.Queue.Status != QueueStatus.Done) && a.SourceCallbackStatus == SourceCallbackStatus.Pending)`
Callback success	Number of outgoing payments, what callback successfully finished `(a => a.SourceCallbackStatus == SourceCallbackStatus.Success)`
Callback fail	Number of outgoing payments, what callback failed `(a => a.OutgoingPaymentCallbacks.Any(s => s.Queue.Status != QueueStatus.Failed && s.Queue.Status != QueueStatus.Done) && a.SourceCallbackStatus == SourceCallbackStatus.Pending)`

Figure 27. Practical part statistics data.

7.3 View and output

View is implemented using Angular and for grid drawing Ag-Grid Angular.

View is refreshed every 2 seconds with rsjx observables to receive updated information from API.

View is divided into two parts:

14. Statistics part
15. Monitoring part

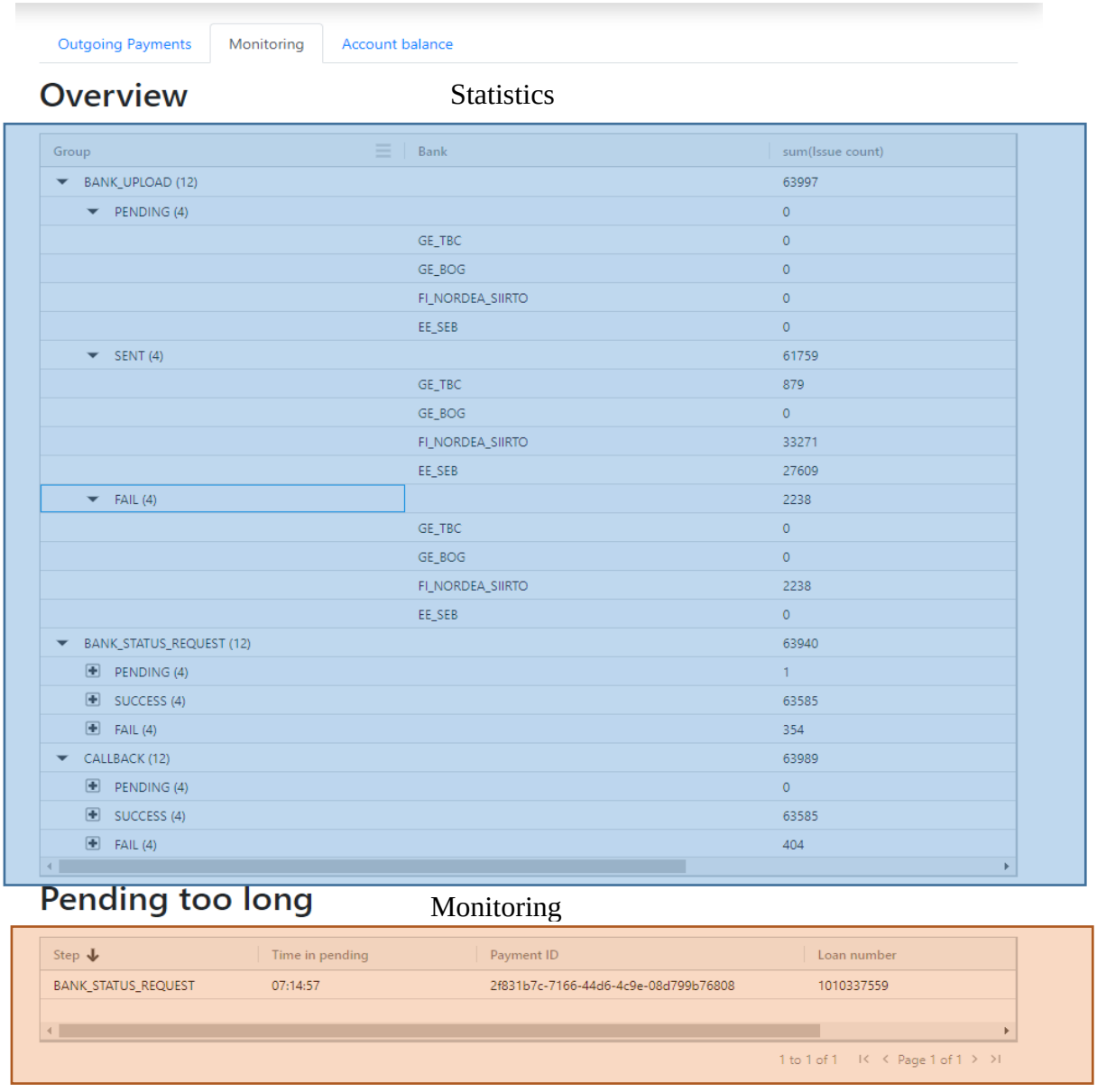


Figure 28. Practical part view.

8 Summary

In this proposed solution or new monitoring and logging system concept I described structure, made comparison with existing solutions and implemented practical part.

In comparison there were three most popular solutions taken, what could meet our needs and as result found out, that there is no such. The biggest problem with all of them, that they store data as they want, they show data as they want, and process it as they want, but storing data architecture should be done on company's side to provide opportunity to process data and show to users as we want.

New system should be structured by projects, project modules, module containers and modules, all this will support full information of the problem: where did it occur, what software was this module running on, state of this software at this time, hardware information, other connected module information and based on this data appropriate action can be taken.

For implementing this concept to real system next point should be analyzed:

- Complexity of the system
- Time/programmers needed for implementation
- Possibility of implementation "step-by-step"
- Possible technical issues

Prototype creation most probably will answer these questions.

Presented practical part of this work is already used to monitor most critical parts of business: money transfer operations. When transfers are in fault state or complies with implemented in work pattern (pending too long in bank), then users can act accordingly to resolve these problems and make customers happy.

Based on this work prototype system will be implemented and full implementation future will be decided.

9 References

[1] https://www.datadoghq.com/product/ (2020)
[2] https://newrelic.com/platform/ (2020)
[3] https://www.nagios.com/products/ (2020)

10 Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis1

I Jevgeni Gavrilov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis ”Logging and monitoring system for BBFinance Group OÜ”, supervised by Henn Sarv
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

04.12.2020

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.