University of Groningen

Software Patterns

Team 2

# Home Energy Monitoring System

**Authors**

Putra, Guntur

Fakambi, Aurélie

Schaefers, Joris

Menninga, Wouter

Monday 30th November, 2015

Version 0.3

# Authors

| Name | E-Mail |
|---|---|
| Putra, Guntur | G.D.Putra@student.rug.nl |
| Fakambi, Aurélie | A.Fakambi@student.rug.nl |
| Schaefers, Joris | J.Schaefers@student.rug.nl |
| Menninga, Wouter | W.G.Menninga@student.rug.nl |

# Revision History

| Version | Author | Date | Description |
|---|---|---|---|
| 0.1 | Schaefers | 15-11-15 | Added a stakeholders section |
| | Schaefers | 15-11-15 | Added a key drivers section |
| | Putra | 15-11-15 | Added Introduction and System context |
| | Fakambi | 15-11-15 | Added non-functional req. and risks |
| | Menninga | 15-11-15 | Added high-level and functional req. |
| | Menninga | 15-11-15 | Improvements to risks |
| 0.2 | Schaefers | 20-11-15 | Added an initial list of patterns with description |
| | Schaefers | 22-11-15 | Improved the analysis chapter, partially restructured and expanded explanations |
| | Menninga | 21-11-15 | Improvements to High-level requirements |
| | Menninga | 21-11-15 | Improvements to Functional requirements |
| | Menninga | 21-11-15 | Added system context / elaborated model |
| | Putra | 22-11-15 | Added hardware architecture |
| | Putra Fakambi | 22-11-15 | Edited introduction |
| | Menninga | 22-11-15 | Expanded system context / elaborated model |
| | Menninga | 22-11-15 | Improvements to use-cases |
| 0.3 | Menninga | 28-12-15 | Improved Unit of Work and Service Layer in ch. 3 |
| | Menninga | 29-12-15 | Changed key drivers |
| | Menninga | 29-12-15 | Added/modified Layers pattern ch. 3 |
| | Menninga | 29-12-15 | Added Unit of Work/Service Layer/Layers to ch.6 |
| | Putra | 29-12-15 | Edited introduction and hardware architecture |
| | Putra | 29-12-15 | Added MVC and Template View pattern in chapter analysis |
| | Putra | 29-12-15 | Added elaborated architecture of MVC and Template View pattern |
| | Fakambi | 29-12-15 | Improvements to the use cases |
| | Fakambi | 29-12-15 | Added the shared repository pattern |
| | Menninga | 29-12-15 | Improvements to Function req. and use cases |
| | Menninga | 29-12-15 | Improved the Elaborated Model |
| | Schaefers | 29-12-15 | Updated the analysis chapter and added the front page pattern. Added the front page structure image to the software chapter |

# Contents

# List of Figures

# List of Tables

# 1 Introduction

This document depicts the software architecture of the Home Energy Monitoring System (HEMS) as the first assignment of Software Patterns course. We proclaim ourself a software architect team working on the Home Energy Monitoring System.

Energy is one of the main concern in a household. The problem is how to monitor energy usage of a particular house in order to prevent inefficient consumption of energy and even to figure out how to save energy, resulting in lower household expenses for energy. An example of energy consumption for a house is shown in Figure 1.1, which indicates that electricity is one of the main energy consumption of a house. Thus, in the initial product this system focuses only on the electricity usage.

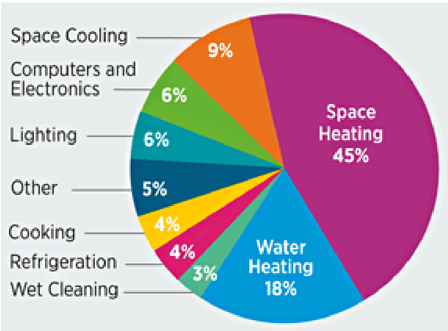


Figure 1.1: An example of home energy consumption [6].

This system will allow the user to collect and store their electricity consumption. This system will also have a web dashboard that shows the graph of electricity consumption, which the user can adjust the time range of the graph to see different data, and other useful data such as which device uses the electricity the most, etc. The data, which will be stored in our database, will be used to do analysis that will help user to predict the upcoming electricity bill in the next month and other required analysis. This system also provides a fault-tolerant computing platform to compute these analyses.

The data collection is not our part, we leave this portion of the system to the third party developers. Basically, we provide a service of cloud computing environment, which can be categorized as SaaS[1], to do energy management system. Figure 1.2 depicts the architectural vision of the HEMS.

[1]Software as a service

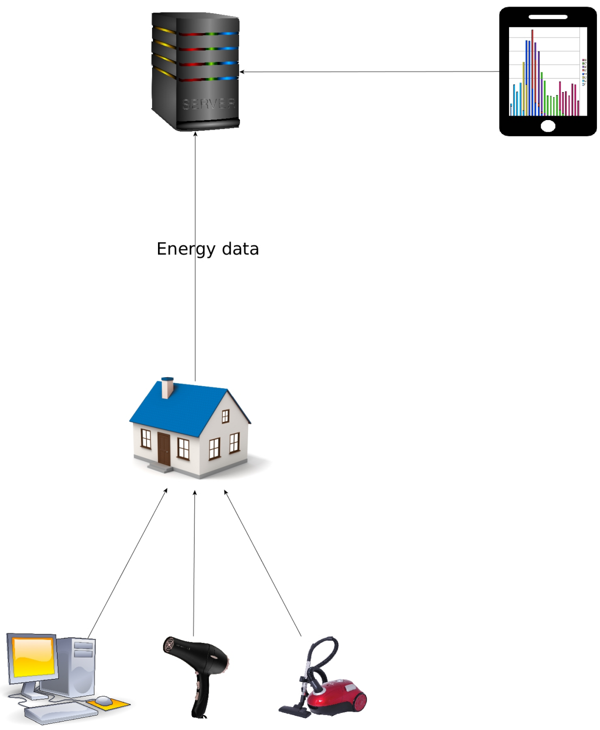


Figure 1.2: Architectural vision

This system provides monitoring for more than one house. Many houses can connect to the system through Internet. Sensors will be located at every device that users are willing to monitor. Real time measurement of electricity consumption will be sent to the HEMS server via Internet protocol by the sensor. Users may see the energy consumption through the web dashboard via computers, tablets, smartphones, or any devices that are able to connect to the Internet.

The rest of the document is structured as follows. The requirements, use cases, stake holders, and key drivers are explained in chapter 2. Chapter 3 describes the analysis and lists the possible pattens to be implemented in this system. Chapter 4 outlines the general system architecture of the system. Hardware selection and architecture are described in chapter 5, while chapter 6 elaborates on software architecture. The evaluation and evolution of this architecture is drawn in chapter 7 and 8.

# 2 Requirements

This chapter describes our vision and uses it to derive stakeholders. This information helps us to be able to properly write use cases and stories. These will be used to extract functional, commercial, technical and evolution requirements. Afterwards, a risk assessment will take place, to ensure that the project is not at great risk.

## 2.1 Stakeholders and their concerns

This section defines all stakeholders of our system and describes the concerns of the stakeholders. A stakeholder might be a person, group of persons, or organization that are involved in our system. There are ten stakeholders, ranged from first parties to third parties stakeholders. Several quality standards from the "Software Requirements" book by Microsoft [7] are used. Those quality standards are described in Table 2.1.

| Quality Attributes | Brief description |
|---|---|
| Availability | The extent to which the system's services are available when and where they are needed |
| Interoperability | How easily the system can interconnect and exchange data with other systems or components |
| Performance | How quickly and predictable the system responds to user inputs or other events |
| Reliability | How reliable the results of the system are (accuracy). |
| Security | How well the system protects against unauthorized access to the application and its data |
| Usability | How easy it is for people to learn, remember, and use the system |

Table 2.1: Quality attributes of Software Architecture from "Software Requirements" Book [7].

There are six quality attributes, as can be seen in Table 2.1, for measuring stakeholders' concern regarding our system. Furthermore, profitability has been added as another quality standard to improve measuring stakeholders' concern. The stakeholders are listed below and are then explained in more detail below.

- Product owner
- Developers
- Maintainers
- Government
- Home owners

**Product owner** is the owner of the website. This stakeholder funds the website and its main concern is creating a profit. This affects the quality attributes usability and availability.

**Developers** have to make sure the system provides the functionality that the users expect the system to have. This means that their main concern is usability. But also interoperability is their main concern, because users will want to connect any energy consuming device to this site in order to monitor the consumption.

**Maintainers** are mainly concerned that the website is up and running at all times. Meaning their concern is the availability and security of the system.

**The government** wants to lower the energy consumption of the citizens. It wants to comply with the aims of the EU to reduce the energy consumption by 20% by 2020. Their main concern is the usability and reliability of the system, because the system is only useful to the government if it is used by allot of people and the statistics of the system are reliable.

**Home owners** want to get an insight in their energy usage. They want to know how to effectively reduce their energy consumption. In order to do this, they might want to receive alerts about sudden increases of energy consumption in certain devices. These alerts have to be accurate and reliable. The main concern of the home owners, thus, is the usability and reliability of the system.

## 2.2 Key-drivers

The key drivers of this system are:

- KD- 1 . Usability
- KD- 2 . Reliability
- KD- 3 . Compatibility

**Usability** has to be the main focus of the system. Users will want to stick to using our system if the usability is better then similar systems of the competitors. Even if competitors have a better availability and or even reliability, there is a good chance that customers will stick to this system if the usability is better.

**Reliability** is very important, because for this kind of system the trust everyone has is crucial. Making the stakeholders trust the system is the most important aspect of this system. If the system at some points does not provide reliable data, without specifically informing about it, then the all the information from the system will be useless. This not necessarily mean that the system has to be very precise in calculating the statistics. It just has to be very clear about how inaccurate the data is.

**Compatibility** is important because in order for the system to be useful to the user, it has to collect and process the energy consumption of devices. All the different homes have different devices who each have a different set of relevant statistics to be calculated. The sensor data will be monitored with a energy monitoring plug. However, to be able to receive valuable statistics from this data, the system needs to be able to cope with different types of devices.

## 2.3 High-level requirements

The high-level requirements describe the high-level functionality of the system. The high-level requirements are used to derive the functional requirements in Section 2.5. The high-level requirements are also used to classify the severity of the risks in Section 2.8.

This table uses different priorities according to RFC-2119[1]. First there is the 'must' ('required') priority, this requirement is an absolute must. The system could not function without the 'must' requirement. Besides 'must' there is the 'should' ('recommended') priority. This priority is highly desirable for the system, but the system could function without this requirement. The last priority is 'may' ('optional'), these requirements are nice to have but not really necessary for the functionality of the system. These different priorities are used through the rest of the document.

| Nr. | Prio | Description |
|---|---|---|
| HL- 1 | **Must** | **Collecting electricity usage data**<br>The system must collect electricity usage data, which has to be stored so it can be used to compute statistics and detect changes in energy usage. |
| HL- 2 | **Must** | **Computing usage statistics**<br>The system must be capable of computing statistics about the energy usage, like total usage per month, but also periods of peak energy usage. It has to be able to compute such statistics not only per house, but also for individual devices. |
| HL- 3 | **Must** | **Displaying statistics**<br>Data which has been gathered and statistics that have been computed must be displayed to the user in an understandable way. One of the main goals of the system is to make users of the system aware about their energy usage. The data and statistics should be displayed in a way that is intuitive and easy to understand for average consumers. |
| HL- 4 | **Must** | **Configuring the system**<br>The users of the system must be able to configure the system, i.e. register their house/add new devices/subscribe to alerts etc., using the web interface. |
| HL- 5 | **Must** | **User-friendly interface**<br>The web interface of the system should be user-friendly and intuitive, such that an average consumer is able to use it. |

Table 2.2: High Level Requirements

## 2.4 Stories and use-cases

In this section the architectural significant use-cases will be presented.

| | |
|---|---|
| **UC- 1** | **The End user registration** |
| **Goal** | The user has to register on the website to create his account. |
| **Primary actor** | End-User |
| **Level** | User goal |
| **Precondition** | The website has the internal process to add an user to the database |
| **Main success scenario** | 1. The user access the HEMS URL<br>2. The website display a registration form requests the full name, username, email address and address of the user.<br>3. The system checks if the user isn't already in the database.<br>4. If not the user is added in the database.<br>5. The user receives a confirmation link on his email address and clicks on it.<br>6. The user account is activated |
| **Postcondition** | The account is created. |
| **Extensions** | 4a. The user is already in the database.<br>1. The user gets to see a message telling him the user is already registered. |
| **Related requirements** | FR-9 |

Table 2.4: UC-1: The End user registration

| | |
|---|---|
| **UC- 2** | **Receiving, collecting and store external energy usage data** |
| **Goal** | The sytem needs to receive energy data and store it. |
| **Primary actor** | The energy usage sensors |
| **Precondition** | The energy data is available |
| **Main success scenario** | 1. The sensor sends energy usage information to the REST API with a security token attached.<br>2. The REST API confirms the attached security token.<br>3. The system stores the energy usage data. |
| **Postcondition** | The data is stored in the database. |
| **Extensions** | |
| **Related requirements** | FR-1, FR-2 |

Table 2.6: UC-2: Receiving external energy usage data

| | |
|---|---|
| **UC- 3** | **Display the estimated bill** |
| **Goal** | The End-User wants to see the estimation of his next bill |
| **Primary actor** | End-User |
| **Level** | User goal |
| **Precondition** | The user is registered in the system. |
| **Main success scenario** | 1. The user gets access to his interface by clicking on "Sign in" Button<br>2. In the menu he clicks on "Bill".<br>3. The system computes the expected bill.<br>4. The system shows the expected bill to the end user |
| **Postcondition** | The estimated bill is displayed |
| **Extensions** | |
| **Related requirements** | FR-6, FR-8 |

Table 2.8: UC-3: Display the estimated bill

| | |
|---|---|
| **UC- 4** | **Display analysis, daily/weekly/monthly report** |
| **Goal** | The End-User wants to display several kind of analysis about his energy consumption |
| **Primary actor** | End-User |
| **Level** | User goal |
| **Precondition** | The user is registered in the system. |
| **Main success scenario** | 1. The user gets access to his interface by clicking on "Sign in" Button<br>2. In the menu he clicks on " Analysis- Chart "<br>3. The user selects the type of statistic<br>4. The user select the period for which to compute the statistic<br>5. The system computes/make the analysis.<br>6. The computed result is shown to the user. |
| **Postcondition** | Charts with different item are displayed. |
| **Extensions** | The statistics aren't made. |
| **Related requirements** | FR-8, FR-5, FR-4, FR-7 |

Table 2.10: UC-4: Display analysis, daily/weekly/monthly report

| UC- 5 | Configuration : adding new devices |
|---|---|
| **Goal** | The user must be able to configure the system with his home devices' characteristics. |
| **Primary actor** | The user |
| **Level** | User goal |
| **Precondition** | The user is registered in the database of the system. |
| **Main success scenario** | 1. After logging in on the dashboard/website the user clicks on the " Settings" button.<br>2. The user gets access to the Settings section with all the internal sections and chooses "Adding new devices".<br>3. The user enters the new device' characteristics and submits. |
| **Postcondition** | The new device and its related information are added in the database. |
| **Extensions** | |
| **Related requirements** | FR-10 FR-11 FR-12 |

Table 2.12: UC-5: Configuration- adding new devices

| UC- 6 | Computation / Statistics |
|---|---|
| **Goal** | The system performs several kind of statistics. |
| **Primary actor** | System |
| **Precondition** | The database of the system has the necessary data. |
| **Main success scenario** | 1. The data needed for the selected computation is filtered<br>2. The system invoks Map/Reduce Functions or Spark and computes |
| **Postcondition** | The analysis is made and the result available/displayed. |
| **Extensions** | The system can't compute the statistics. |
| **Related requirements** | FR-4 FR-5 FR-6 FR-7 FR-8 FR-10 FR-16 FR-19 |

Table 2.14: UC-6: Computation-statistics

## 2.5 Functional requirements

This section lists the functional requirements of the system.

| Nr. | Prio | Description |
|---|---|---|
| FR-1 | **Must** | The system must be able to receive electricity usage data from devices. |
| FR-2 | **Must** | The system must be able to store electricity usage data. |
| FR-3 | **Must** | The system must be able to retrieve previously stored electricity usage data. |
| FR-4 | **Must** | The system must be able to compute the total electricity consumption for a given time period for a particular house. |
| FR-5 | **Must** | The system must be able to compute the electricity consumption per device for a given time period. |
| FR-6 | **Must** | The system must be able to compute an estimated electricity bill for the current month based on the electricity consumption to that point. |
| FR-7 | **Must** | A user of the system must be able to select which statistics to compute in a web interface. |
| FR-8 | **Must** | The system must be able to show the computed statistics in a web interface. |
| FR-9 | **Must** | The system must allow users to register an account on the web interface. |
| FR-10 | **Must** | The system must require users to be logged in, before the user can view electricity usage information about his/her house. |
| FR-11 | **Must** | A user of the system must be able to register a new house using the web interface. |
| FR-12 | **Must** | A user of the system must be able to register a new device for his house using the web interface. |
| FR-13 | **Must** | A user of the system must be able to configure the price of a kWH in the web interface. |
| FR-14 | **Must** | The system must be able to send feedback to registered devices about the current electricity usage. |
| FR-15 | **Must** | The system must be able to take the inaccuracy of the sensors into account when computing the statistics. |
| FR-16 | **Must** | The system must be able to display the history of electricity usage. |
| FR-17 | Should | Users of the system should be able to subscribe to alerts in the web interface, alerting them about sudden energy increases or when they are using more energy than in previous months/weeks. |
| FR-18 | Should | The system should send alerts to users by mail when the user is subscribed for this alert and the condition for the alert is met. |
| FR-19 | Should | The system should be able to show the computed statistics in a graph. |

Table 2.15: Functional Requirements

## 2.6 Technical non-functional requirements

### 2.6.1 Usability

The sytem will be hopefully used by a lot of people who don't necessarily have knowledge in the technology field so the Usability is an important requirement so the end-user can acces all the information he needs.

| | |
|---|---|
| **US- 1** | An application is available for tablets and phone with those OS Windows Phone,Android, and Iphone. |
| **US- 2** | The end-user needs maximum ten minutes to get a basic understanding of system features through the UI. |
| **US- 3** | Every feature/major option of the system can be accessed through the home page(Single Page Application). |

### 2.6.2 Reliability

| | |
|---|---|
| **RE- 1** | A margin of error of 5% in the energy measurements is tolerated. |
| **RE- 2** | The sytem should be available 99.9%. of the time which means down for 44 minutes. |
| **RE- 3** | The system should be down for maximum ten minutes when the user installs a new release(version). |

### 2.6.3 Security

Ensuring the security for the system is a major issue so all the data needed for its good functioning remain protected and consistent.

| | |
|---|---|
| **SEC- 1** | Each user is identified and has to log in in order to view his "Home Energy Monitor" account. |
| **SEC- 2** | The data is encrypted. |

### 2.6.4 Scalability

The system should make the increase of workload, resources and users easy that's why the scalability is an important Key driver.

| | |
|---|---|
| **SCA- 1** | The system should function as efficiently even if the number of users increases. |

## 2.7 Evolution requirements

## 2.8 Risk assessment

The system is confronted by several risks which are determined and mitigated in this section. Taking those risks into account allows to avoid them or at least reduce their impact. The risk management involves the identification of the risks, their probability and potential impact or consequences.

The tables below explain the meaning of the definition for probability and consequence.

| Probability | Likelihood of occurrence |
|---|---|
| High | 0.65 - 1.00 |
| Medium | 0.35 - 0.65 |
| Low | 0.00 - 0.35 |

Figure 2.1: The different probabilities used to classify risks

| Severity | Explanation |
|---|---|
| Severe | A risk that can lead to loss of live or casualties. |
| Significant | A risk that can lead to damages, can delay the project more than 3 months or causes one of the high-level requirements not to be fulfilled. |
| Moderate | A risk that can lead to one of the high-level requirements not to be fulfilled to an acceptable level. |
| Minor | A risk that can lead to one of the high-level requirements not being fully fulfilled, but still fulfilled in an acceptable level. |

Figure 2.2: The different severities used to classify risks

### 2.8.1 Technical

| **T-RISK1** | **The statistics provided by the data processing framework are wrong** |
|---|---|
| **Probability of occurrence** | Low |
| **Consequences** | Moderate. If the statistics computed by the system are not accurate, this will lead to loss of thrust of the end user in the system. |
| **Prevention** | Make sure the algorithms used to compute the statistics are correct. |
| **Reaction** | Correct the algorithm, if the change is significant also inform end users about the error. |

Table 2.20: T-RISK1 – The statistics provided by the data processing framework are wrong

| **T-RISK2** | **The data center storing the energy measurements becomes unavailable** |
|---|---|
| **Probability of occurrence** | Low |
| **Consequences** | Significant |
| **Prevention** | Store the data in a redundant way, preferably in multiple data centers, so that one data center going offline does not lead to downtime of the system. |
| **Reaction** | If the data storage does become unavailable, new incoming data should be cached so it is not lost and users should be informed in the web interface that viewing the statistics is temporarily unavailable. |

Table 2.21: T-RISK2 – The data center storing the energy measurements becomes unavailable

| T-RISK3 | **Somebody gains unauthorized access to someone else's data** |
|---|---|
| **Probability of occurrence** | Low |
| **Consequences** | Significant. Data about electricity usage can be used, e.g. to derive when people are home. Unauthorized data access will lead to a loss of thrust in the system by consumers. |
| **Prevention** | Make sure access to the data requires authentication using a password at all time. Enforce users to use a strong password. |
| **Reaction** | Make sure the unauthorized access is removed. Inform end users about which data was accessed by the unauthorized party. |

Table 2.22: T-RISK3 – Somebody gains unauthorized access to someone else's data

### 2.8.2 Business

| B-RISK1 | **Wrong estimation of the budget** |
|---|---|
| **Probability of occurrence** | Medium |
| **Consequences** | Significant. The final product does not have the features expected. |
| **Prevention** | The team needs an accountant or at least someone taking care of the follow-up of the money. Make sure there are regular evaluations to keep track of the money flow. |
| **Reaction** | Remove some requirements or features of the product, or change the hardware components used. |

Table 2.23: B-RISK1 – Wrong estimation of the budget

| B-RISK2 | **Wrong estimation of the budget: the money invested in the fabrication and achievement of the product/system is not covered by the sales (shortfall/deficit)** |
|---|---|
| **Probability of occurrence** | Medium |
| **Consequences** | Moderate. Stopping the sale |
| **Prevention** | The team needs an accountant or at least someone taking care of the follow-up of the money. |
| **Reaction** | Adding more features to the product in order to make it more competitive in the market. |

Table 2.24: B-RISK2 – Wrong estimation of the budget: the money invested in the fabrication and achievement of the product/system is not covered by the sales (shortfall/deficit)

### 2.8.3 Schedule

| **S-RISK1** | **The project is not finished at the deadline** |
|---|---|
| **Probability of occurrence** | Low |
| **Consequences** | Significant. Pressure for all the team members, loss of credibility regarding the customers, selling a product with less features than expected. |
| **Prevention** | SRA , Schedule Risk analysis : Estimation of the duration of the project by its manager ( with the use of probability and statistics ) . Meeting for the team members every week to keep track of the timing and take decisions according to the deadline. |
| **Reaction** | Postpone the deadline or remove some features when the deadline can't be postponed. |

Table 2.25: S-RISK1 – The project is not finished at the deadline

# 3 Analysis

## 3.1 Layers

**Source**

Pattern-oriented Software Architecture - Volume 1, P.31 [2]

**Issue**

The system consists of high-level components (e.g. user interface), which are dependent on lower level components (domain logic, database). Decoupling these components is important.

**Assumptions/Constraints**

- It is assumed that the system will have higher-level components that depend on lower-level components.

**Positions**

1. Relaxed Layered System
2. Layers

**Decision**

The system will use the Layers pattern to divide the application in multiple layers. The top layer will implement the Presentation logic. Below that will be the Service Layer pattern, the third layer will implement the domain logic (domain model) and the bottom layer will be responsible for the data storage.

**Argument**

1. The relaxed layered system is a variant of the Layers pattern which allows layers to use services of any layer below it, instead of only the next layer. This increases the flexibility and performance of the system. However, this pattern has a large negative impact on the maintainability.
2. Using the layers pattern increases maintainability by decoupling components of different levels of abstraction.

**Implications**

Using layers will have a positive impact on the maintainability and re-usability of the system. An increased maintainability will help prevent bugs, which on the longer term helps to increase the reliability of the system.

The performance of the system will have some negative impact, since the request have to pass all the layers, even if they only need logic/data in the bottom layer.

**Related requirements/decisions**

Service Layer,

## 3.2 Service Layer

**Source**

Patterns of Enterprise Application Architecture, P.133 [4]

**Issue**

The system will have different interfaces with different kinds of clients. For example, commands to compute statistics can come from the user interface, but also from the alerting module. These different interfaces have common interactions with the system to invoke the business logic.

**Assumptions/Constraints**

- The application's business logic and/or data are accessed using several interfaces.

**Positions**

1. Domain Model
2. Service Layer

**Decision**

The system will use the Service Layer pattern to define the boundary of the application using a layer of services.

The service layer exposes a set of services to be used by clients and for each service, there is a certain script that will be executed when the service is called. The service layer will be used with the "operation script" variation. This means that the Service Layer consists of thick classes that contain logic.

**Argument**

1. Using just the Domain Model pattern alone does not make a distinction between application logic and domain logic.
2. The service layer does allow making a distinction between the application- and domain logic, which

**Implications**

There is a boundary between the application- and domain logic.

**Related requirements/decisions**

Layers, FR-7, FR-1

## 3.3 Front page

**Source**

- Pattern-oriented Software Architecture - Volume 4 [3] P.339
- Patterns of Enterprise Application Architecture[4] P.344

**Issue**

All the different kinds of HTTPS requests that can be made to the system have similar initial handling. If every request has a dedicated handler, the code for handling these similar functions would be duplicated. This decreases the reliability of the system and might have negative affects on the availability as well.

**Assumptions/Constraints**

- The users use a web browser to view a graphical interface
- There is a central system handling all the HTTPS requests

**Positions**

1. Page controller
2. Front controller

**Decision**

The front page provides more functionality to this system. The front page resides in the service layer. Here it will create a pipe of filters and functions using an Intercepting Filter ([5]). This will allow the pipe to be created using a decorator pattern ([5]) The front page will be configured to handle the logging, authentication and initial security of the request. Finally the front page dispatches the requests to the domain using the command pattern ([5]).

**Argument**

1. By using the front controller, a pipeline can be created that handles the similar activities in handling requests, having all the similar request functionality in one central place.
2. It simplifies configuring the system.

3. Front page reduces concurrency issues, because a new command object is created for each request. Reducing thread-safety concerns. The model, however, can have shared objects that do require thread safety management.

**Implications**
The page controller is more intuitive. Having a controller for each request is a clear way of structuring a web server. It increases the understanding of each single request separately, however it decreases the overall understanding because each request could be handled completely different.

**Related requirements/decisions**
KD-2, KD-3, FR-1

## 3.4 Domain layer

**Source** Patterns of Enterprise Application Architecture[4] P.116 Pattern-oriented Software Architecture - Volume 4, P.182 [3]

**Issue** The domain logic consists of complex functions for serving web request and analyzing data. The functionality of the system must be modifiable and must reduce the amount of duplicate code to a minimum in order to prevent inconsistencies.

**Assumptions/Constraints**

- The presentation layer uses a front controller pattern

**Positions**

1. Transaction Script
2. Table Module
3. Domain model

**Decision** The domain model pattern will be used.

**Argument** The domain logic is complex and so it requires the use of the domain model pattern. This means that the domain is Object Oriented, with every class representing one specific, individual, meaningful part. The domain model is the most advanced domain pattern, minimizing code duplication and increasing the flexibility of the system. The domain model also integrates well with the front page controller in the first layer.

**Implications** Having a script that is solely responsible for handling a specific request, would not be conform the domain model. This makes it harder to understand how the requests are handled, because the used functions might be spread across the classes in the domain layer.

**Related requirements/decisions**
FR-4 FR-5 FR-6 FR-15 FR-17 KD-3

## 3.5 Unit of work

**Source**
Patterns of Enterprise Application Architecture, P.184 [5]
Pattern-oriented Software Architecture - Volume 4, P.541 [3]

**Issue**
The system has several object stored in a database which can be edited and created. For example, new sensor data of devices becomes available and changes to the configuration (changing device names etc.) can be made. Updating database records on each change leads to a lot of database calls, which is bad for performance.

**Assumptions/Constraints**

- The domain layer uses the domain model pattern.
- Requests to store and receive data are being made continuously while the server runs.

**Positions**

1. Active Record
2. Unit of Work with Caller Registration
3. Unit of Work with Object Registration

**Decision**
: Unit of Work pattern will be used to keep track of changes to objects and to coordinate writing these changes to the database in one database call. The variant used is the Object Registration variant, which removes the burden of having to register the object explicitly with the Unit of Work. Registering the object self could introduce bugs.

**Argument**

1. With Active Record, every change to an in-memory object leads to one or multiple database calls. This reduces the performance significantly, especially when using the domain model pattern.
2. The Unit of Work instead keeps track of these changes to allow writing these changes to the database in a single call.
   When using Unit of Work with Caller Registration, the creator of the object should register it with the Unit of Work explicitly, or else the Unit of Work will not keep track of its changes. This offers the flexibility to have changes to in-memory objects that are never written to database.
3. Using Unit of Work with Object Registration means that the creator of the object does not need to explicitly register it with the Unit of Work. Instead, the created object is implicitly registered (e.g. as part of the logic in the constructor).

**Implications**
: Using Unit of Work will reduce the load on the database (the number of database calls). It does however introduce a delay before the change in the in-memory object is present in the representation in the database.

**Related requirements/decisions**
: KD-1, FR-2, FR-3

## 3.6 Broker

**Source**
: Pattern-oriented Software Architecture - Volume 4, P.237 [3]

**Issue**
: The system uses several servers to compute the statistics. This introduces a lot of challenges, like communication to these servers and dividing the work over these servers. The application code should not have to address these challenges.

**Assumptions/Constraints**
: It is assumed that the workload for computing statistics can be divided over multiple nodes, and that multiple nodes are available.

**Positions**

1. Publisher-Subscriber
2. Broker
3. Message Queuing
4. Remote Procedure Call

**Decision**
: The broker pattern will be used. The broker pattern is part of the domain layer.

**Argument**
Using a broker creates a layer of abstraction for the specific layer using it. This increases encapsulation, location independence and scale ability.

**Implications**

- Adding an other layer of abstraction which must be used can reduce the performance of the system
- If the broker fails, the sourced that the broker provided can't be accessed any more.
- A bug in the broker is hard to trace back.

**Related requirements/decisions** FR-4, FR-6, FR-6

## 3.7 Data source layer

**source**
- Pattern-oriented Software Architecture - Volume 4, P.540 [3]
- Patterns of Enterprise Application Architecture[5],

**Issue**

**Assumptions/Constraints**

- The domain layer uses the domain model pattern

**Positions**

1. Table data gateway
2. Row data gateway
3. Active record
4. Data mapper

**Decision**
The data mapper pattern will be used. The data mapper pattern creates the most abstraction between the domain and data source layer.

**Argument**

- The data mapper pattern best suits the domain model pattern used in the domain layer
- The logic that will operate on the data will generate statistics of the data, which can become quite complex. This is why the data mapper pattern is chosen as a pattern for connecting to the data sources. This pattern is the most advanced pattern, and provides the best functionality and abstraction. With the active record pattern, the database access/communication, data and logic of that data is all stored in the same class. Since the logic that will be executed on the sensor could be quite complex, this patterns will not be useful. A gateway of any kind leads to performance overhead, because for each call coming from the domain model pattern, a database call is made. It lacks the necessary coordination.

**Implications**
The data source mapper requires the most amount of work to set up, but it

**Related requirements/decisions**

## 3.8 Repository

The repository pattern is used by the broker pattern to get the sources the domain layer requests. The repository clients create a criteria object, specifying what kind of data is wanted. For example $criteria.equals(Device.NAME, Computer$

Then the clients use this criteria by invoking repository.matching(criteria) to receive the data from the repository. The client just asks the data, it has no further knowledge about any interaction with any data source/data base.

The repository gives a lot more control over how the data is handled. The benefits are:

- Reduces code (and code complexity)
- Increases performance
- Separated domain and data layers, increasing flexibility and changeability

Performing analysis on the data also consists of executing complex queries on the data source. The database that executes these queries, however, might change. Or the system might decide to use multiple databases and data sources. Using the Repository pattern, these changes can be made fast. The repository also allows for multiple configurations to exist. So an extra repository could be created for testing purposes, only using an in-memory database to increase the test execution speed.

The Shared Repository pattern will be applied and the Architectural decisions documented according to the template presented in *Using Patterns to Capture Architectural Decisions*
"In the SHARED REPOSITORY pattern one component of the system is used as a central data store, accessed by all other independent components."

**Source**
: Patterns of Enterprise Application ArchitectureP.322 [5] *Architectural Pattern Revisited- A Pattern Language*

**Name**
: Shared Repository (also called Repository)

**Issue**
: The external data must be stored, updaded, accessed by multiple clients and servers simultaneously and analyzed.

**Position**
: When using the Active Repository Pattern the clients are notified when there is a change is the repository which is not a feature of the system.

**Decision**
: The system uses the Shared Repository Pattern in order to centralized the access to the data.

**Implication**
: After the integration of the SRP to the system the system as one central repository (the database) and all clients can access it using loggin. It ensures Reusability, Changeability, Maintainability.

**Related decisions**
: Data source layer pattern

**Related requirements**
: FR-1, FR-2, FR-3

## 3.9 Model-View-Controller

**Source**
: Pattern-oriented Software Architecture - Volume 1 P.125 [2]
Patterns of Enterprise Application Architecture, P.330 [4]

**Issue**

The system must handle the request from user by dynamic web interface. In order to increase re-usability, the views may be decoupled from the logic, resulting in separated logic (controller and model) and view.

**Assumptions/Constraints**

- It is assumed that the client uses web browser to connect to the system.
- It is assumed that there are multiple views in the system.

**Positions**

1. Presentation-Abstraction-Control
2. Layers-Model-View-Controller

**Decision**

The system will use the Model-View-Controller pattern to decouple the view and the logic behind it. This pattern resides in the service layers of this system.

**Argument**

1. The Presentation-Abstraction-Control (PAC) pattern decompose the system into a tree-like hierarchy of agents, which each agent is made of its own presentation (UI), abstraction, and control. This type of pattern is not well suited to the HEMS since the HEMS only needs decoupled views with the same control and abstraction.
2. The Model-View-Controller (MVC) pattern only decouple the views, models, and controller, which indicates that different views may use the same controller and/or model. This pattern is well-suited with the HEMS and is good for re-usability, which we think has less complexity.

**Implications**

By implementing the MVC pattern, several views are created to support multiple pages. Each view may use the same controller that provides required process. This will also allows building a filter chain, handling authentication, logging etc. Thus, this will drive better reusability and have less complexity. Furthermore, it is also easier to modify the code since the view and the logic are separated.

**Related requirements/decisions**

FR-3, FR-7, FR-10, FR-18, FR-19

## 3.10 Template view

**Source**

Patterns of Enterprise Application Architecture, P.350 [4]

**Issue**

In the view component of MVC, the system must handle dynamic web interface in a Single Page Application (SPA) flavored application. The same view (HTML code) will be displayed several time in different page, which will be possibly causing code smells.

**Assumptions/Constraints**

- This pattern is only applied on web application.
- The system is developed using framework that supports this pattern.

**Positions**

1. Transform view
2. Two step view
3. Template view

**Decision**

The system will utilize template view in order to increase reusability and to prevent code smells in the project. In this way, several HTML code snippets that corresponds to same element of a page will be reused by embedding markers. Thus, this will lead to a better modifiability as well.

**Argument**

1. Transform view processes domain data element by element and transforms it into HTML. In the system, such task is handled by the models. Thus, if this pattern is used, there will be duplication of work.
2. Two step view turns domain data into HTML in two steps: first by forming some kind of logical page, then rendering the logical page into HTML. These operations are considered as inefficient, as the forming logical page and rendering HTML can be joined. There is also overlapping between models, controllers, and views in this pattern.
3. Template view renders information into HTML by embedding markers in an HTML page, which solves the issues and prevents code smells.

**Implications**

Several HTML code snippet will be reusable, which is good for reusability and simplicity. Furthermore, code smells can be prevented using this template view.

**Related requirements/decisions**

FR-3, FR-7, FR-10, FR-18, FR-19

# 4 System architecture

This chapter outlines the general system architecture of the system. The first section shows the system with its inputs and outputs to and from external factors.

## 4.1 System context

The system context diagram in Figure 4.1 gives an overview of the entities that interact with the system.

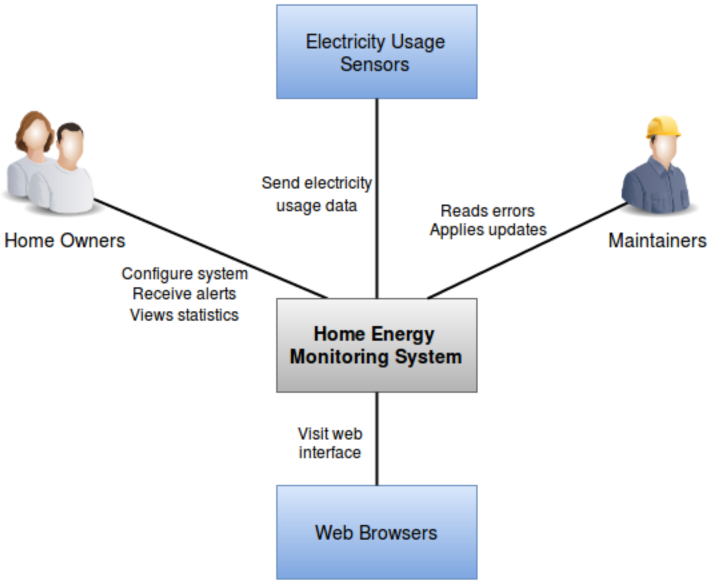


Figure 4.1: System context diagram

### 4.1.1 Users and roles

**Home Owners**

The home owners are the main users of the system. They want to know about the electricity usage in their house.

They interact with the system by viewing statistics, configuring the system and they receive alerts if they configured the web interface to send these.

**Maintainers**

The maintainers of the system will apply updates to the system and read errors that might have occurred in order to solve these.

### 4.1.2 External systems

**Electricity Usage Sensors**

The electricity usage sensors are the sensors that measure the electricity usage of the devices. These sensors work by measuring the electricity passing through a power outlet. This means that they in fact measure the electricity usage of a power outlet and not that of a device. This is relevant if multiple devices are connected to the same power outlet .

**Web Browsers**

The web interface of the system, where the system can be configured and statistics can be viewed, is presented as a web page. Users will use a range of different web browsers (on a range of different devices) to visit this web interface. It is important that the web interface works equally well in all these different web browser.

## 4.2 Elaborated Model

In Figure 4.2 the elaborated system model is shown. This figure gives a high-level overview of the software and hardware of the system. In this figure, the arrows represent the data flow.

Each house sends collected electricity usage data to the systems 'Data receival API', which stores the collected data in a storage cluster. An end-user of the system can visit the web interface using a web browser on any device. Using this web interface, the user can generate statistics data, which is done by a separate software component of the system.

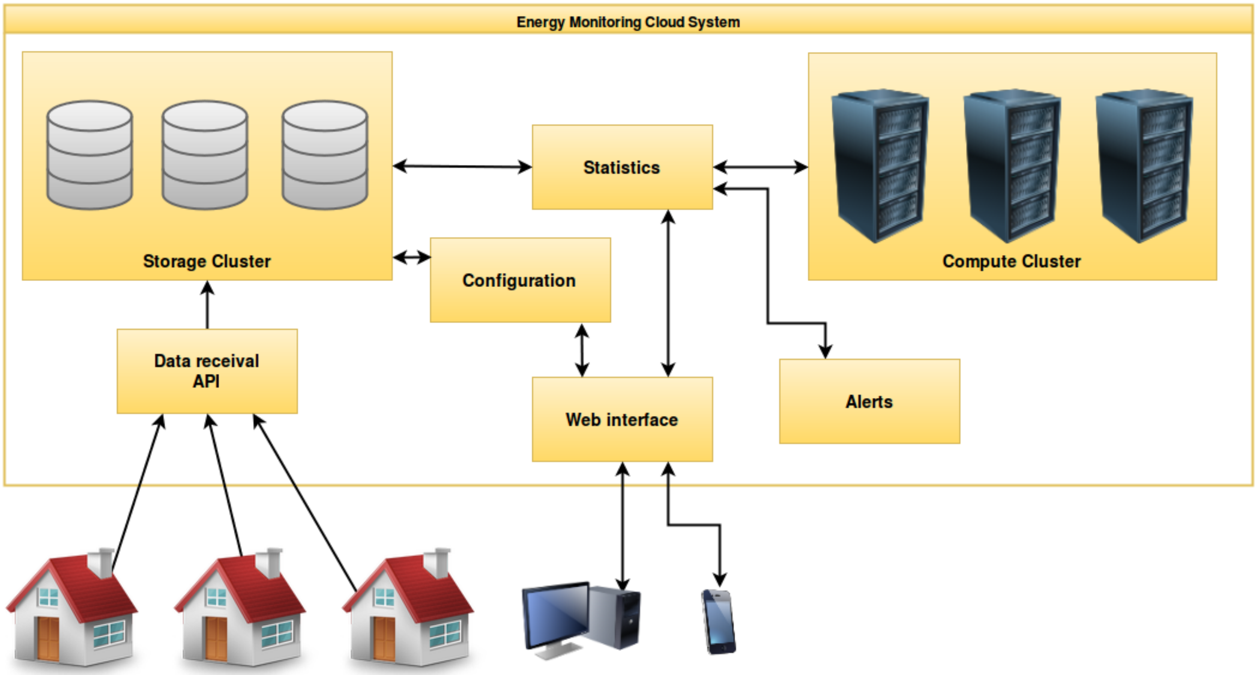


Figure 4.2: Elaborated model

**Data receival API**
: The 'Data receival API' component exposes a REST interface, which is used by the homes to send electricity usage data to the system. The API is responsible for storing the data in the storage cluster.

**Storage cluster**
: The storage cluster is a set of servers which will store all the data in a replicated way.

**Web interface**
: Users of the system (home owners) visit the web interface (presented as a website) through all kinds of devices. The web interface allows users to compute and view statistics from the collected usage data. The web interface also allows users to register new devices with the system.

**Statistics**
: The statistics part of the system is responsible for computing statistics from the collected usage data. It will receive commands to do so through the web interface.

**Compute cluster**
: The compute cluster is a cluster of servers, which are used to compute statistics from the raw collected electricity usage data.

**Configuration**
: The configuration part offers the functionality needed to edit the configuration of the system (add/change devices etc.) to the web interface.

**Alerts**

The Alerts components checks computes statistics (as configured by the user) regularly, to see if a certain criteria is met. If this is the case, the user will receive an alert by email.

# 5 Hardware Architecture

This section describes the hardware architecture of Home Energy Management System (HEMS). The description will be more high-level along with explanations about the hardware platform and the application interfaces between each components of the system. The rest of this chapter is organized as follows; First section, section 5.1, presents an overview of the hardware implemented in this system depicted in big schema. Decisions made in this system are detailed in section 5.2 with tables. Lastly, the hardware is described in section 5.3.

## 5.1 Hardware Overview

As mentioned in previous chapters, HEMS focuses on providing services to monitor electricity usages based on installed sensors on each customer's house. Thus, this system works on the cloud part, which is providing data storage, monitoring, and analysis, both in terms of application (software or service) and hardware. This system deals with no electricity collecting devices. Therefore, the electricity collection part is delegated to third party developers.

According to chapter system architecture, the main part of the HEMS hardware consists of storage cluster and compute cluster. The storage cluster is responsible to handle incoming data from sensors through the exposed data acquisition API. This cluster is capable to store real-time data. The compute cluster mainly deals the data presentation of the stored usage data. Furthermore, compute cluster is also needed to perform analysis based on stored data.

The detailed hardware selection to perform and build this system is explained in the following section.

## 5.2 Hardware Design Decisions

This section defines decisions made regarding the hardware selection. Tables will be used to make our justification in regard to hardware selection more clear.

| Name | Compute cluster selection |
|---|---|
| **Decision** | **HW- 1** |
| **Status** | **Approved** |
| **Problem/Issue** | HEMS needs a reliable computers to do the analytical processing. |
| **Decision** | HEMS will use clustered Dell PowerEdge R530 to act as the main analytic cluster and to provide API to the actors. |
| **Alternatives** | *HP ProLiant DL360 Gen9 Base*<br>This server rack has 16GB of memory and 2.4GHz of processor speed. As other server computer, this machine utilizes Intel Xeon E5 2600v3. This server is suitable for high dense computing, however the price is not so suitable for this kind of specification. It does not have LCD screen that will help technician to look the current status of the server.<br>*Lenovo System x3550 M4 7914*<br>This server rack has only 8GB of memory. However, the processor is a bit faster, it runs on 2.6GHz. As other server computer, this machine also utilizes Intel XEON E5-2600. The price is a little bit lower than the others but the memory limitation makes it not so valuable. It has LCD screen that will help technician to look the current status of the server.<br>*Dell PowerEdge R530*<br>This 2U server rack has 16GB of memory and 2.4GHz of processor speed. This machine utilizes Intel Xeon E5-2620V3 with 15MB of cache. This server is suitable for high dense computing. It has LCD screen that will help technician to look the current status of the server. |
| **Arguments** | |

| | Reliability | Performance | Interoperability | Security | Scalability | Cost | **Score** |
|---|---|---|---|---|---|---|---|
| Weights 3 | 2 | 3 | 2 | 2 | 2 | 2 | |
| Dell PowerEdge R530 5 | 3 | 5 | 4 | 4 | 4 | 5 | 70 |
| Lenovo System x3550 M4 7914 4 | 3 | 4 | 4 | 3 | 4 | 4 | 60 |
| HP ProLiant DL360 Gen9 Base 5 | 3 | 5 | 4 | 3 | 4 | 3 | 64 |

Table 5.1: Decision – Analytic cluster selection

| Name | Storage cluster selection |
|---|---|
| **Decision** | **HW- 2** |
| **Status** | **Approved** |
| **Problem/Issue** | The system needs reliable computers to store the data. |
| **Decision** | HEMS will utilize Synology RackStation RS814RP to store the data. |
| **Alternatives** | *Synology RackStation RS814RP*<br>This storage machine has the fastest connection among the others. This machine will run at SATA with 6 Gbps connection.<br>*70BJ NAS-server*<br>This machine form factor is 1U which is suitable for saving space. However, the connection speed is limited to 3 Gbps.<br>*Thecus N8810U-G NAS-server*<br>This machine also runs in 3Gbps connection. However, the form factor is 2U which makes this machine takes more space in the rack. |
| **Arguments** | |

| | Reliability | Performance | Interoperability | Security | Scalability | Cost | **Score** |
|---|---|---|---|---|---|---|---|
| Weights 3 | 2 | 3 | 2 | 2 | 2 | 2 | |
| Synology RackStation RS814RP 5 | 2 | 4 | 4 | 4 | 4 | 4 | 63 |
| 70BJ NAS-server 4 | 2 | 3 | 4 | 4 | 4 | 3 | 55 |
| Thecus N8810U-G NAS-server 4 | 2 | 3 | 4 | 4 | 4 | 3 | 55 |

Table 5.2: Decision – Choice of storage machine

## 5.3 Hardware Description

This section gives an outline of the hardware implemented in this system. This section also elaborates on hardware decisions.

### 5.3.1 Storage cluster

HEMS will use clusters of computer to manage the database system and to store our data. The cluster enables the database to replicate data. This means no backup is needed, the data is always available in multiple servers. Furthermore, user account information is hashed using bcrypt after being salted with 128 randomly generated characters to increase security. The cluster will also be accessible by the main analytics part as the data will come and go through the main analytics part. The logical schematic of the storage cluster is depicted in Figure 5.1

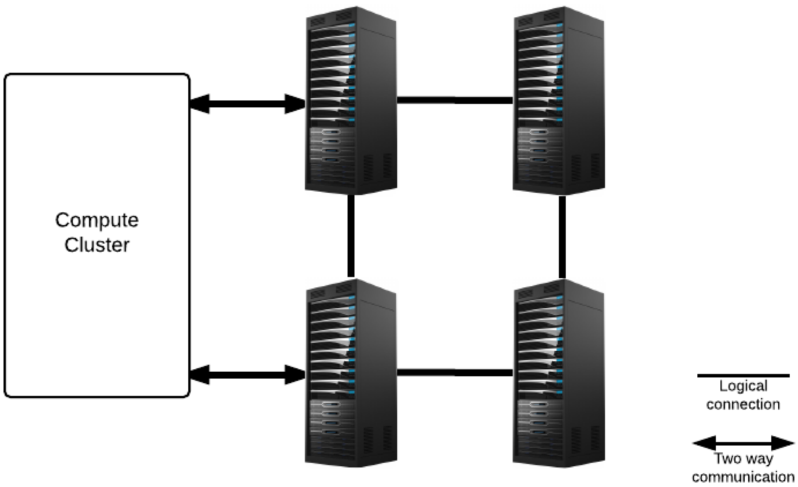


Figure 5.1: Logical schematic of storage cluster of HEMS

As can be seen in Figure 5.1, HEMS will use four database racks to have redundancy in the system. The server is connected as a ring, which is the common way to setup database server. By using this form of architecture, HEMS will be more reliable and fault tolerant. There will also be two physical connection to the main analytic cluster to make this system more fault tolerant in terms of connection. HEMS database cluster will use the same server, Dell PowerEdge R530, for controlling the SATA storage machine.

### Related patterns

This storage cluster implements broker, shared repository, and unit of work patterns. Broker pattern is implemented in a mechanism that any other component of this system can connect to the cluster and see this as a single entity, although actually the cluster consists of more than one entity. If this storage cluster is seen as a single entity, then this is also an implementation of shared repository pattern, where a client or other instance connects to the storage cluster and proceeds with corresponding operations. Unit of Work pattern will be used to keep track of changes to objects and to coordinate writing these changes to the database in one database call.

### 5.3.2 Compute cluster

Compute cluster will be the main brain of HEMS. The data presentation will be handled by this system. The analysis process also runs on top of this hardware. To increase availability and reliability, HEMS will have six server racks to do the processing as depicted in Figure 5.2.

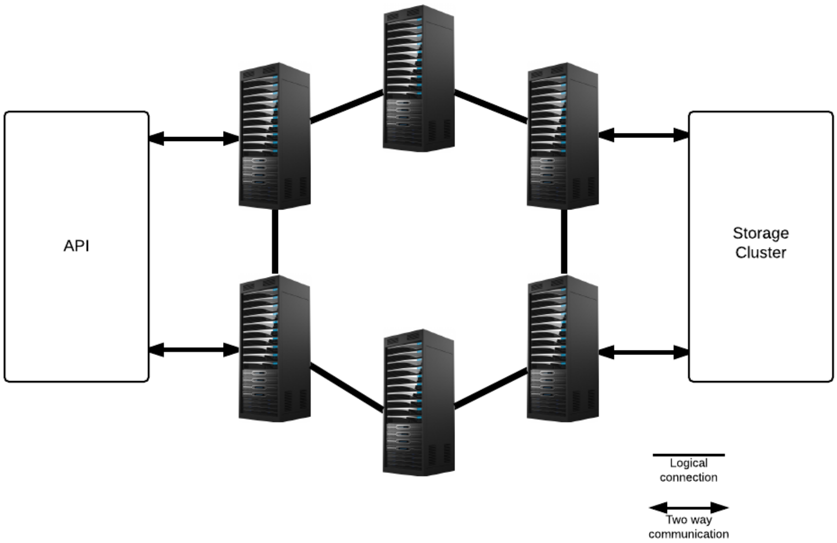


Figure 5.2: Logical schematic of analytic cluster of HEMS

## Related patterns

Front page controller pattern is implemented in the compute cluster, which includes decorator and command pattern, because the compute cluster is also responsible for handling incoming connection through API.

# 6 Software Architecture

## 6.1 Views

The following section will elaborate the two of the '4+1 model' views of the system, namely the Logic View and the Process View.

### 6.1.1 Logic view

### 6.1.2 Process view

## 6.2 Elaborated model with patterns

This section will describe the elaborated model on the basis of the patterns used in the architecture. For each patterns, this section will describe how it is implemented and how it affects the quality attributes of the system.

### 6.2.1 Layers pattern

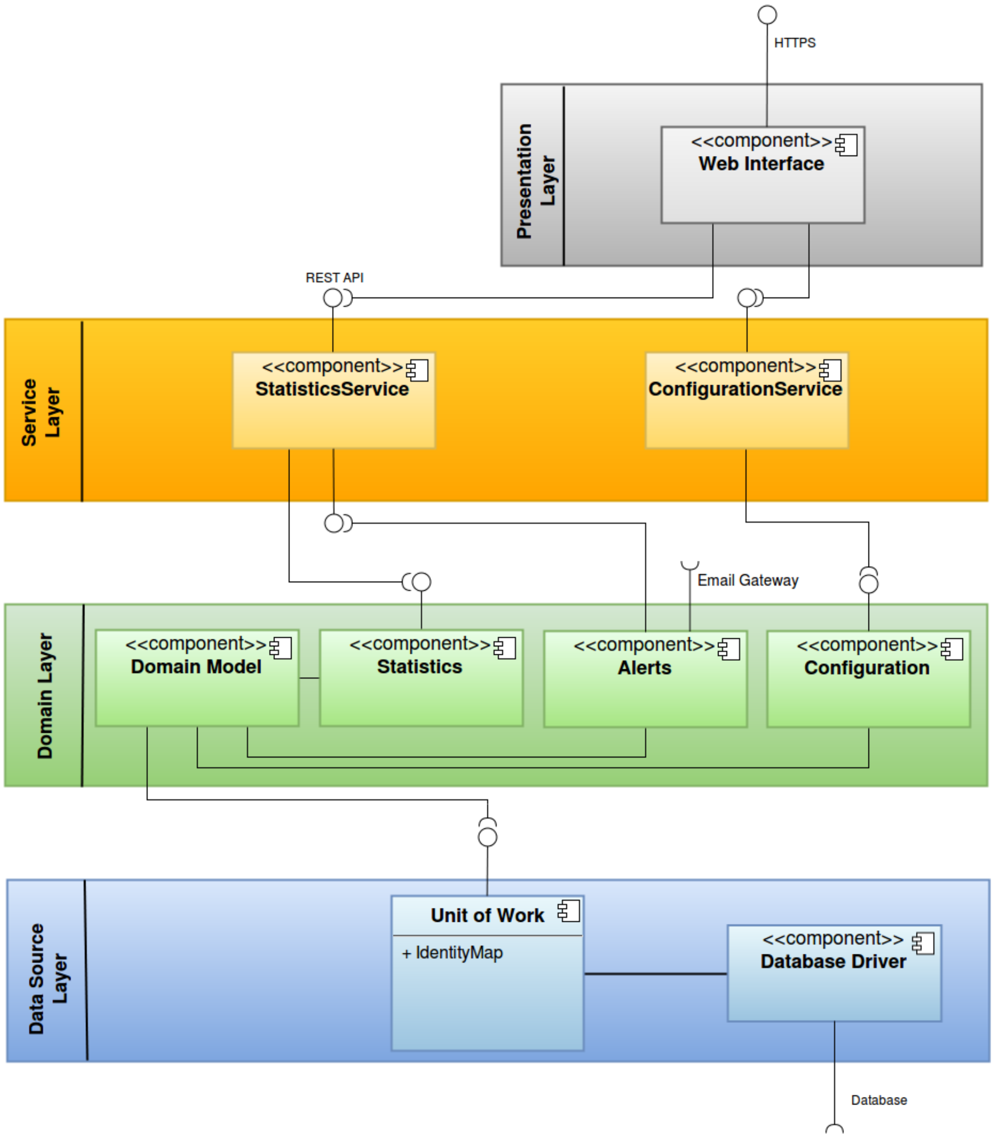


Figure 6.1: The Layers

The system is divided into four layers. The first layer is the presentation layer. This layer is responsible for handling the interaction with the end user. It contains the Web Interface, which is accessible to the user over HTTPS.

The second layer is the Service Layer, see Section 6.2.2 for more information about this layer. This layer offers services, which can be used by other (external) components.

The third layer is the Domain Layer and is responsible for the domain logic. The Domain Model contains all the classes, has an in-memory representation of the data and contains the logic which is inherent to the objects. It uses the Unit of Work pattern (see Section 6.2.4) to keep track of the changes to objects, so not every change will lead to a new database call.
The components in the Domain Layer are connected to the Domain Model, so they have access to the classes in there.

The Alerting component is responsible for sending alerts by email to the end user, for which it depends on an external Email Gateway. It also uses the StatisticsService in order to compute statistics, which it needs to decide if the user should be alerted by email.
The Configuration and Statistics components expose their functionality to the Service Layer.

The Data Source layer contains the Unit of Work, which keeps track of the changes to the objects and translates those changes to database transactions when the object is committed. The layer also contains a Database Driver, which handles the communication with the database.

### 6.2.2 Service Layer pattern

The service layer encapsulates the application's business logic and defines the set of available operations/interfaces to clients. In Figure 6.1, the Service Layer and its components can be seen.

It contains the 'StatisticsService', which exposes an interface used by clients to store the electricity usage data and is also used by the Web Interface for computing statistics. It is also used by the Alerts component in the Domain Layer, since it needs the same computed statistics as are displayed in the web interface, to decide if the user should be alerted by email.

The Service Layer also has the 'Configuration Service', which is used by the Web Interface to allow the user to add new devices, change their properties or to configure new alerts.

The Service Layer makes a common set of application functionality available to many kinds of clients. This promotes the interoperability and also prevents having duplicate code.

### 6.2.3 Front page

In the figure below, the structure of the front page pattern is shown, which uses the decorator and command pattern.

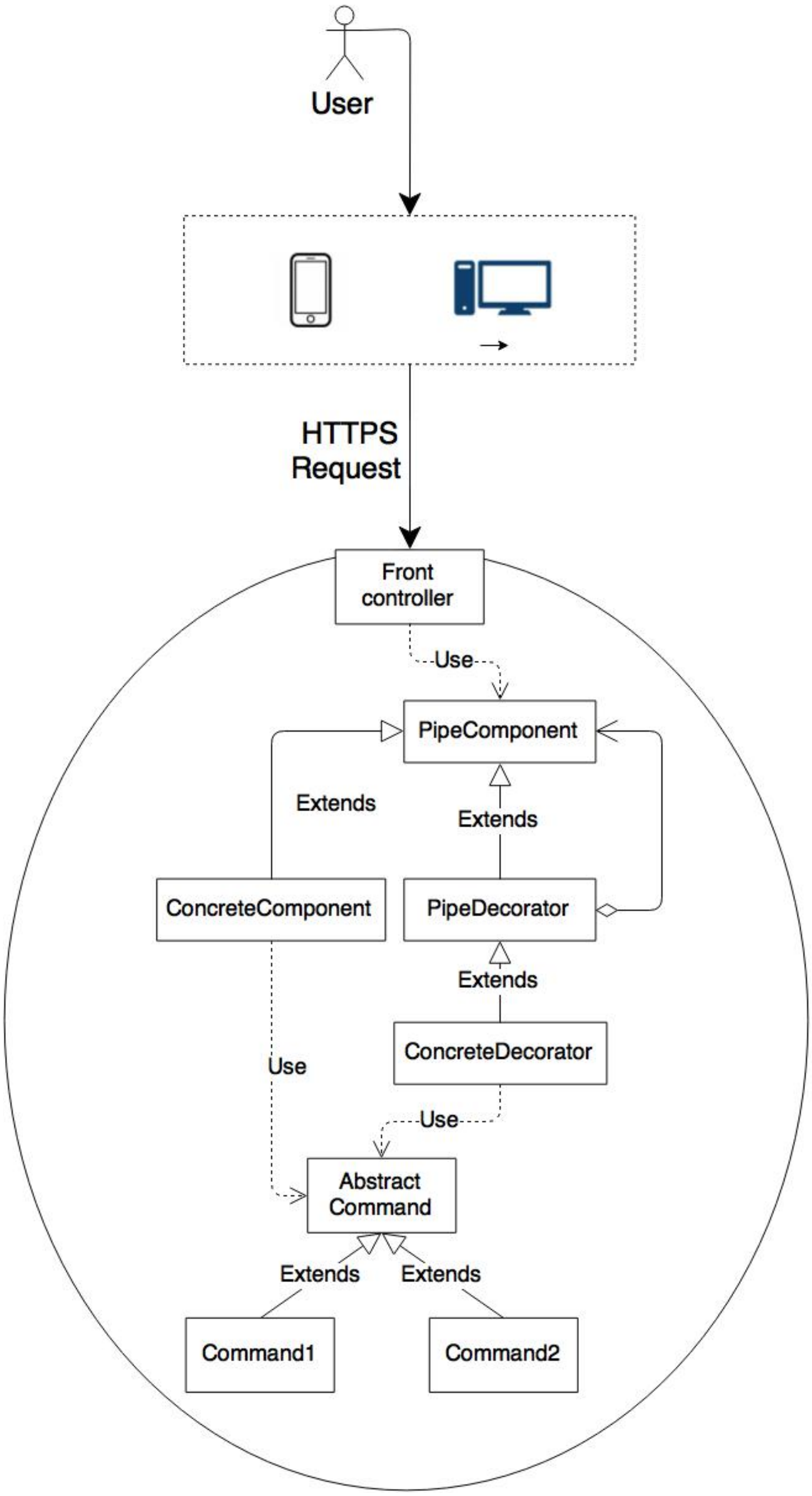


Figure 6.2: The front controller

i

### 6.2.4 Unit of Work pattern

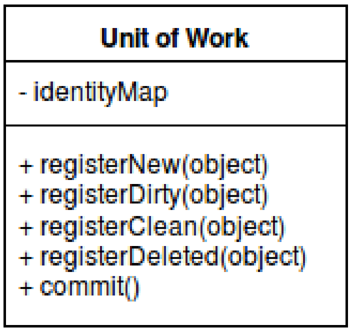


Figure 6.3: The Unit of Work class

The Unit of Work pattern is used to keep track of the changes made to objects and newly created objects. Whenever an object is created, changed or deleted, the Unit of Work is told about this. Whenever the object can be saved to the database, the `commit()` method of the Unit of Work is called, which translates the stored changes into database transactions.

A sequence diagram showing an example of this can be seen in Figure 6.4. Here, the StatisticsController constructs a new Device object, which is fetched from the database and then registers itself with the Unit of Work. When the StatisticsController changes the name of this device, the device object registers itself as dirty with the Unit of Work. When the device object is saved, it calls `commit()` on the Unit of Work, which leads to the device updating the appropriate fields in the database.

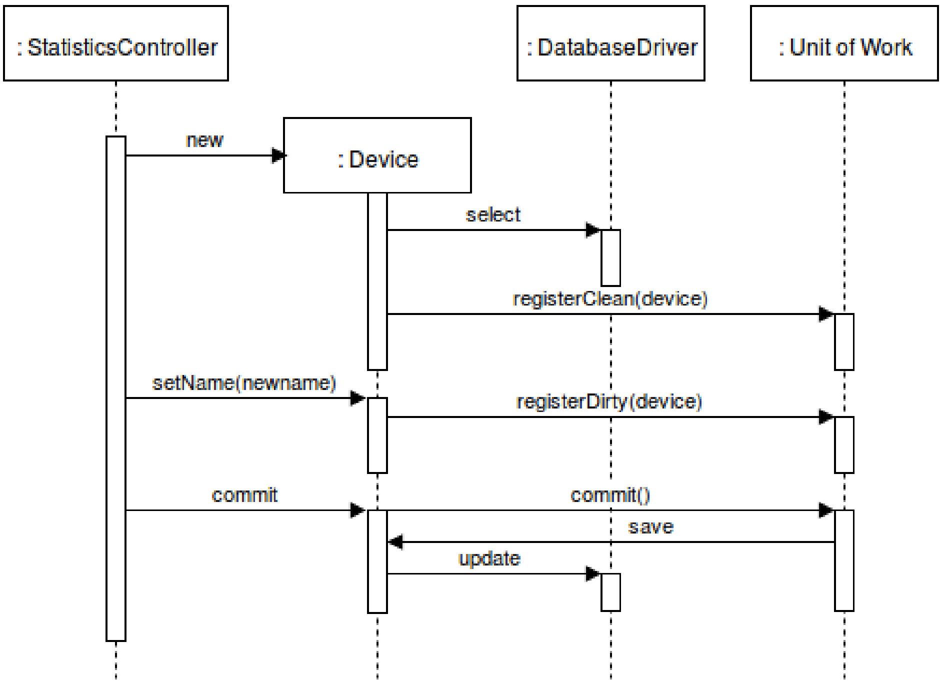


Figure 6.4: Sequence diagram showing an update to a Device-object using Unit of Work

### 6.2.5 Model-View-Controller

As mentioned in chapter 3, MVC pattern is applied to decouple user-interface and the logic behind it. In this way, reusability is increased because the same models or controllers can be coupled with the same view. Modifiability is also increased because it becomes easier to modify a particular user interface or data model without interfering the logic, and vice-versa. Figure 6.5 depicts an example of MVC implementation in the HEMS.

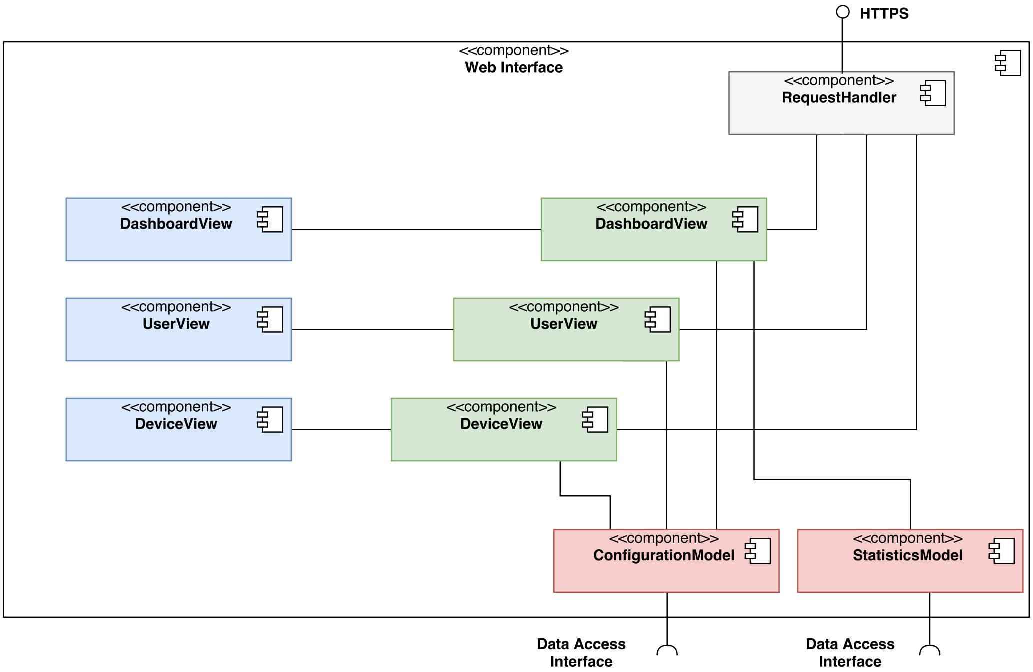


Figure 6.5: Model-view-controller pattern implementation

Some models, views, and controllers are depicted in Figure 6.5. Request handler handles incoming user request via HTTPS and routes it to the corresponding controller. Required data is then obtained through the models. Suitable views are used to provide user interface to the user. Some models, views, and controllers are presented in Figure 6.5. However, there are more models, views, and controllers than those which are represented in the Figure 6.5.

### 6.2.6 Template View

Template view is implemented in this system to make the HTML code reusable in different pages. This will also make the view structure more simple. Code duplication can be prevented because instead of duplicating the code, the HTML will use a certain template.

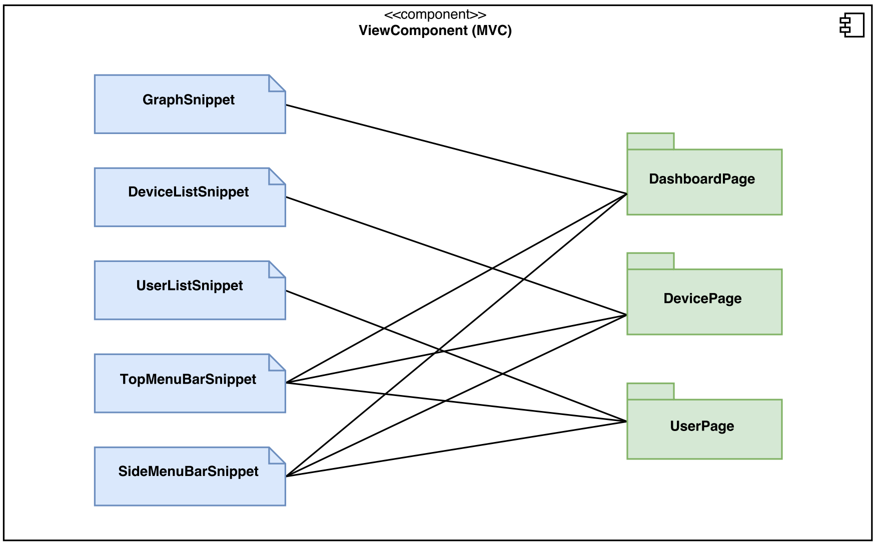


Figure 6.6: Template view pattern implementation

Figure 6.6 shows an example of implementation of the template view pattern. Each page of the system (presented in green color) will combine several HTML code snippets (presented in blue color) together. `TopMenuBarSnippet` and `SideMenuBarSnippet` are used several times, as each page contains top menu bar and side menu bar. This is also good for expandability because the new page may just combine existing page template to create new web page.

### 6.2.7 Shared repository view

This view represents the communication with the central database all the clients of the system.

# 7 Architecture evaluation

# 8 System evolution

# A Time Tracking

## Week 1

| Person | Task | Hours |
|---|---|---|
| Putra | Meetings, writing introduction and system context. | 4 |
| Fakambi | Meeting, First approach on non functional requirements and risks | 4 |
| Schaefers | Meetings, Added a stake holders and key drivers description. | 5 |
| Menninga | Meeting, high-level and functional requirements and improvements to risks | 5 |

## Week 2

| Person | Task | Hours |
|---|---|---|
| Putra | Meetings, editing introduction and adding hardware architecture. | 5 |
| Fakambi | Meetings , work on the use-cases | 6 |
| Schaefers | Meetings, researched patterns to use and created initial analysis draft | 9 |
| Menninga | Meetings, system architecture, use case improvements | 8 |

## Week 3

| Person | Task | Hours |
|---|---|---|
| Putra | Meetings, processing feedback, working on MVC and template view patterns in analysis and software architecture chapter | 12 |
| Fakambi | Meetings,updated use-cases, work on the Shared repository pattern | 8 |
| Schaefers | Meetings, researching patterns in the various books, describing front controller and command patterns | 14.0 |
| Menninga | Meetings, feedback, working on Layers/Unit of Work/Service Layer patterns (ch. 3 & 6), improved elaborated model | 13.0 |

# Bibliography


[1] S. Bradner. Key words for use in RFCs to indicate requirement levels, 3 1997. RFC 2119.

[2] K Henney F Buschmann and D.C. Schmidt. *Pattern-oriented Software Architecture - Volume 1 - A system of patterns*. 2007.

[3] K Henney F Buschmann and D.C. Schmidt. *Pattern-oriented Software Architecture - Volume 4*. 2007.

[4] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[5] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[6] Rik Langendoen. How to reduce home energy-related costs. `http://greenifynow.com/wp/?p=795`, 2013. [Online; accessed 22-November-2015].

[7] K. Wiegers and J. Beatty. *Software Requirements*. Developer Best Practices. Pearson Education, 2013.