

Politechnika Warszawska

W Y D Z I A Ł   M A T E M A T Y K I  
I   N A U K   I N F O R M A C Y J N Y C H



# Praca dyplomowa licencjacka

na kierunku Matematyka

NAJPIERW WYGENERUJ STRONĘ TYTUŁOWĄ

KTÓRA ZNAJDUJE SIĘ

Numer albumu 000000

promotor

W KATALOGU title\_page

konsultacje

I OTRZYMANEGO PDF-A NAZWIJ titlepage.pdf I WSTAW DO KATALOGU Z  
SZABLONEM!

WARSZAWA 2018

.....

supervisor's signature

.....

author's signature

## Abstract

### A System for Resources Management in a Small Chemical Laboratory

For the subject of our engineering thesis, we have resolved a problem of management of supplies in a small chemical laboratory. We have designed and implemented a system supporting management of resources (chemical reagents, instruments, etc.). The system keeps track of the state of resources in the laboratory, and stores the data in a database. The system has graphical user-friendly interface which facilitates displaying and modifying the gathered data. Multiple functionalities in the system allow the user to:

- Classify the resource into groups
- Assign descriptions with multimedia content to resources
- Define and generate reports and notifications. reports are displaying current state of resources as well as plot of activity in time, plot of demand for some resource in time, bar chart, pie chart, whereas notifications are alerts about low level of some chemical reagent, etc.
- For each resource, store suppliers' contact data and order new resources directly from the application
- Predict future demand for resources based on available historical data

The implemented system is a Web Application. It consists of a client application – developed in AngularJS – and server – developed in SpringBoot Java. Elements of application exchange information using RESTful API. As per development tools, we used IntelliJ IDEA (a Java integrated development environment (IDE)).

**Keywords:** Resource Management, Database, Time Series Forecast, Web Application, Java, AngularJS, SpringBoot, RESTful API, IntelliJ IDEA



## Streszczenie

### System zarządzania zasobami małego laboratorium chemicznego

Jako temat pracy inżynierskiej, rozwiązaliśmy problem zarządzania zasobami w małym laboratorium chemicznym. Zaprojektowaliśmy i zaimplementowaliśmy system wspomagający rozporządzanie zasobami (reagentami chemicznymi, itd.) System zbiera informacje o stanie zasobów i przechowuje je w bazie danych. Aplikacja posiada przyjazny dla użytkownika interfejs graficzny, co ułatwia wyświetlanie i modyfikacje zebranych danych. Pozostałe funkcjonalności systemu pozwalają na:

- Klasyfikację zasobów w grupy
- Przypisanie zasobom opisów i multimediów (np. zdjęć)
- Definiowanie i generowanie raportów i notyfikacji. Raporty przedstawiają obecny stan zasobów, jak również wykresy aktywności (produkcji) w czasie, wykresy zapotrzebowania na zasoby w czasie, wykresy słupkowe, wykresy kołowe. Notyfikacje to alerty o niskim poziomie zasobów.
- Dla każdego zasobu, przechowywane są informacje o danych kontaktowych do dostawcy oraz umożliwiające jest zamówienie nowych bezpośrednio z aplikacji.
- Przewidywanie przyszłego zapotrzebowania na zasoby na podstawie danych historycznych

Zaimplementowany system jest aplikacją internetową. Składa się z aplikacji klienta - zaimplementowanej w AngularJS – i serwera – zaimplementowanego przy użyciu SpringBoot Java. Elementy aplikacji wymieniają się danymi za pomocą RESTful API. jako narzędzie deweloperskie, korzystaliśmy z IntelliJ IDEA.

**Słowa kluczowe:** Zarządzanie zasobami, Baza danych, Prognozowania na podstawie szeregu czasowego, Aplikacja internetowa, Java, AngularJS, SpringBoot, RESTful API, IntelliJ IDEA



Warsaw, .....

### Declaration

I hereby declare that the thesis entitled „A System for Resources Management in a Small Chemical Laboratory”, submitted for the Engineer degree, supervised by dr inż. Agnieszka Jastrzębska, is entirely my original work apart from the recognized reference.

.....





# Contents

|                                     |           |
|-------------------------------------|-----------|
| <b>Introduction</b>                 | <b>11</b> |
| <b>1. Work Division Plan</b>        | <b>12</b> |
| 1.1. Work Division                  | 12        |
| <b>2. Background of the Problem</b> | <b>13</b> |
| 2.1. Other Known Solutions          | 13        |
| <b>3. Requirement Specification</b> | <b>14</b> |
| 3.1. Functional Requirements        | 14        |
| 3.2. Non-functional Requirements    | 15        |
| 3.3. Use Cases                      | 15        |
| 3.3.1. Administrator                | 15        |
| 3.3.2. Manager                      | 17        |
| 3.3.3. User                         | 19        |
| <b>4. Development Methodology</b>   | <b>21</b> |
| 4.1. Methodology                    | 21        |
| 4.1.1. Argumentation                | 21        |
| <b>5. System Architecture</b>       | <b>23</b> |
| 5.1. Software Architectural Pattern | 23        |
| 5.2. Backend Architectural Pattern  | 23        |
| 5.2.1. Data Access Object Pattern   | 23        |
| 5.2.2. Plain Old Java Object        | 24        |
| 5.2.3. Dependency Injection         | 25        |
| 5.2.4. RESTful API                  | 26        |
| <b>6. Technical Analysis</b>        | <b>29</b> |
| 6.1. Program Interface              | 29        |
| 6.2. System Classes                 | 29        |
| 6.3. Database Design                | 29        |
| 6.4. Prediction Model               | 32        |

**7. Post Execution Documentation . . . . . 33**

7.1. Evaluating Functional Requirements and Use Cases . . . . . 33

7.2. Evaluating Non-Functional Requirements . . . . . 33



## Introduction

The subject of the presented paper is the implementation of the system for management of supplies in a small chemical laboratory. The idea for such solution came from a real life situation, where an existing chemical laboratory needed an application to facilitate their work. Throughout all phases of design and implementation, the real needs of these existing chemical laboratory were taken into consideration and many elements of the application originated from there. For instance, when thinking of a database, we matched the design to what we knew would be stored in this database. For example there are tables to store different products and resources, but the relationship between those two tables reflects the laboratories' formulas, that is, which products are made of which resources and in which proportion. The following chapters will describe in depth:

- The set of initial requirements including business analysis and the background of the problem
- The detailed design of the application
- The technical information of how we realised this project, including development model and work division
- Some final conclusions along with post implementation valuation of our resulting application

## 1. Work Division Plan

The project was designed so as it could be completed by the group of three people. The tasks were divided so that each member of the group was assigned a part, and these parts were thought to be equally time consuming. That being said, all three participants of the project contributed towards the building of the system, its design and the frame application. What is more, great emphasis has been put on collaboration and team work, resulting in members of the group often performing tasks outside their divided part, which contributed toward the project's final success.

### 1.1. Work Division

Table 1.1: Work Division

| Name             | Responsibility  |
|------------------|---|
| Klaudia Jarosz   | Implementation of the frame application   |
|                  | Creating a database containing data about users, resources, suppliers and user's activity |
|                  | Design and implementation of a user-friendly interface                                    |
| Maciej Głowala   | Implementation of the frame application   |
|                  | Handling users with different roles   |
|                  | Saving and restoring system state   |
|                  | Virtual server setup  |
| Aleksandra Bułka | Implementation of the frame application   |
|                  | Implementation of reports and notifications   |
|                  | Forecasting module  |
|                  | Ordering module   |

## **2. Background of the Problem**

The idea of the L.I.M.E. project comes from the well-known problems of managing the warehouses and laboratories. Small companies tackle with many perplexities connected with governance of production and the resources.

### **2.1. Other Known Solutions**

The known solutions for the problem:

1. World-Class Warehousing and Material Handling by Edward Frazelle
2. Essentials of Inventory Management by Max Muller
3. Warehouse Management: A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse by Gwynne Richards
4. Inventory Accuracy: People, Processes, & Technology by David J. Piasecki
5. Introduction to Materials Management by Steve Chapman

Those problems may be hard to resolve. And there comes the L.I.M.E. which allows to register products and resources, managers and workers. Application delivers job records, predictions, and auto resource orders. It's ideal for small companies, who wants to improve control of staff, products and resources.

### 3. Requirement Specification

#### 3.1. Functional Requirements

The functional requirements cases of the applications are different for different users of the application. The table contained in this chapter provide functional requirements for different groups of application users. The three groups of users of the application are:

1. Administrator (manager of the whole system and its users)
2. Manager (a person with rights for laboratory resources management)
3. Registered user

Table 3.1: Functional Requirements

| Actor         | Description  |
|---------------|--|
| Administrator | Log in to the system, change and recover his password                                    |
|               | Create, modify and remove an account in the system, modify roles                         |
|               | View, create, modify and remove resources and products and their groups                  |
|               | Define, generate and send a report or prediction   |
|               | Define notifications, turn notifications on/off  |
|               | Order resources, turn on/off automatic ordering  |
|               | Save the current state of the system, schedule a system backup or restore it from backup |
| Manager       | Log in to the system, change and recover his password                                    |
|               | View, create, modify and remove resources and products and their groups                  |
|               | Define, generate and send a report or prediction   |
|               | Define notifications, turn notifications on/off  |
|               | Order resources, turn on/off automatic ordering  |
| User          | Log in to the system, change and recover his password                                    |
|               | View, create, modify and remove resources and products                                   |

### 3.2. NON-FUNCTIONAL REQUIREMENTS

|  |  |
|--|--|
|  | Define, generate and send a report or prediction |
|  | Order resources                                  |

### 3.2. Non-functional Requirements

Table 3.2: Non-functional requirements

| Area           | Number | Details   |
|----------------|--------|---|
| Usability      | 1      | Application must be responsive. It must be working on PC, tablets and phones with resolution at least 720p.   |
| Reliability    | 2      | Application must be of type High Availability. It should be available 24h/7d between 08:00 and 23:00. There could be service breaks during the week between 24:00 and 8:00. |
|                | 3      | Application must have quick restart in case of app machine failures.  |
| Recovery       | 4      | Application must have daily database recovery performed between 24:00 and 08:00.  |
| Performance    | 5      | Application should respond no longer than 3 seconds while strain being on level 100 queries per minute.   |
| Supportability | 6      | Documentation should contain instruction for recovery data from database backup.  |
|                | 7      | Application should keep backward compatibility between the released versions.   |
| Security       | 8      | Application must have user levels security. It shall not pass a user who has inappropriate privileges.  |

### 3.3. Use Cases

The uses cases of the applications, similarly to functional requirements are different for different users of the application. The tables contained in the following chapters provide descriptions of use cases for different groups of application users.

#### 3.3.1. Administrator



Table 3.3: Uses Cases for Administrator

| Actor         | Name                      | Description  |
|---------------|---------------------------|--|
| Administrator | Login                     | Log in to the system   |
|               | Password Management       | Recover his password   |
|               |                           | Change his password  |
|               | User Account Management   | Create an account in the system, assign the account to a role (user, manager)  |
|               |                           | Modify an account in the system – change either personal data or assignment to a role (user, manager)  |
|               |                           | Remove an account from the system  |
|               | Resource View             | Display nicely current availability of resources and their categorization - multiple viewing perspectives, sorting and filtering are available                               |
|               | Resource Management       | Define a new type of resource, describe it with description card, add multimedia content to this resource (for example a photograph) and assign the resource with a supplier |
|               |                           | Modify a resource, change description card, multimedia content associated with this resource (for example a photograph) and its assignment to a supplier                     |
|               |                           | Delete a resource from database  |
|               | Resource Group Management | Create a group of laboratory resources, define which resources will belong to this group   |
|               |                           | Modify a group of laboratory resources, redefine which resources will belong to this group   |
|               |                           | Delete a group of laboratory resources   |
|               | Product View              | Display nicely products produced by laboratory and their categorization  |
|               | Product Management        | Define a new type of product, describe it with description card, add multimedia content to this product (for example a photograph)   |
|               |                           | Modify a product, change description card, multimedia content associated with this product (for example a photograph)  |

### 3.3. USE CASES

|                              |  |  |
|------------------------------|--|--|
|                              |  | Delete a product from database   |
| Product Group Management     |  | Create a group of laboratory product, define which products will belong to this group                                      |
|                              |  | Modify a group of laboratory products, redefine which products will belong to this group                                   |
|                              |  | Delete a group of laboratory products  |
| Report Definition            |  | Define what the report will contain, for example a plot of production in time, plot of demand for some resource in time    |
|                              |  | Define how the data will be presented, for example bar chart, pie chart, table with adjustable columns/rows                |
|                              |  | Generate the desired report  |
|                              |  | Define the recipients and send the report  |
| Prediction Report Definition |  | Define what the report will contain, this can be either for example production in time or demand for some resource in time |
|                              |  | Define for which resources, products or groups of resources or products the prediction should be made                      |
|                              |  | Generate the desired prediction report   |
|                              |  | Define the recipients and send the prediction report   |
| Notification Definition      |  | Define whether notifications (an alert about low level of some chemical reagent) will it be sent                           |
|                              |  | Define when the notifications will it be sent (set the value which is critical for each resource)                          |
| Order Management             |  | Define how many and which resources are to be ordered  |
|                              |  | Send an order  |
|                              |  | Turn on automatic ordering of resources or turn it off   |
| System State Management      |  | Save the current state of the system   |
|                              |  | Schedule an automatic back-up of a system state  |
|                              |  | Restore system state based on an archived backup   |

#### 3.3.2. Manager

Table 3.4: Uses Cases for Manager

| Actor   | Name                      | Description  |
|---------|---------------------------|--|
| Manager | Login                     | Log in to the system   |
|         | Password Management       | Recover his password   |
|         |                           | Change his password  |
|         | Resource View             | Display nicely current availability of resources and their categorization - multiple viewing perspectives, sorting and filtering are available                               |
|         | Resource Management       | Define a new type of resource, describe it with description card, add multimedia content to this resource (for example a photograph) and assign the resource with a supplier |
|         |                           | Modify a resource, change description card, multimedia content associated with this resource (for example a photograph) and its assignment to a supplier                     |
|         |                           | Delete a resource from database  |
|         | Resource Group Management | Create a group of laboratory resources, define which resources will belong to this group   |
|         |                           | Modify a group of laboratory resources, redefine which resources will belong to this group   |
|         |                           | Delete a group of laboratory resources   |
|         | Product View              | Display nicely products produced by laboratory and their categorization  |
|         | Product Management        | Define a new type of product, describe it with description card, add multimedia content to this product (for example a photograph)   |
|         |                           | Modify a product, change description card, multimedia content associated with this product (for example a photograph)  |
|         |                           | Delete a product from database   |
|         | Product Group Management  | Create a group of laboratory product, define which products will belong to this group  |
|         |                           | Modify a group of laboratory products, redefine which products will belong to this group   |

### 3.3. USE CASES

|  |                              |  |
|--|------------------------------|--|
|  |                              | Delete a group of laboratory products  |
|  | Report Definition            | Define what the report will contain, for example a plot of production in time, plot of demand for some resource in time    |
|  |                              | Define how the data will be presented, for example bar chart, pie chart, table with adjustable columns/rows                |
|  |                              | Generate the desired report  |
|  |                              | Define the recipients and send the report  |
|  | Prediction Report Definition | Define what the report will contain, this can be either for example production in time or demand for some resource in time |
|  |                              | Define for which resources, products or groups of resources or products the prediction should be made                      |
|  |                              | Generate the desired prediction report   |
|  |                              | Define the recipients and send the prediction report   |
|  | Notification Definiton       | Define whether notifications (an alert about low level of some chemical reagent) will it be sent                           |
|  |                              | Define when the notifications will it be sent (set the value which is critical for each reasource)                         |
|  | Order Management             | Define how many and which resources are to be ordered  |
|  |                              | Send an order  |
|  |                              | Turn on automatic ordering of resources or turn it off   |

#### 3.3.3. User

Table 3.5: Uses Cases for User

| Actor | Name                     | Description  |
|-------|--------------------------|--|
| User  | Login                    | Log in to the system   |
|       | Password Man-<br>agement | Recover his password   |
|       |                          | Change his password  |
|       | Resource View            | Display nicely current availability of resources and their categorization - multiple viewing perspectives, sorting and filtering are available |

### 3. REQUIREMENT SPECIFICATION

|  |                              |  |
|--|------------------------------|--|
|  | Resource Management          | Define a new type of resource, describe it with description card, add multimedia content to this resource (for example a photograph) and assign the resource with a supplier |
|  |                              | Modify a resource, change description card, multimedia content associated with this resource (for example a photograph) and its assignment to a supplier                     |
|  |                              | Delete a resource from database  |
|  | Product View                 | Display nicely products produced by laboratory and their categorization  |
|  | Product Management           | Define a new type of product, describe it with description card, add multimedia content to this product (for example a photograph)   |
|  |                              | Modify a product, change description card, multimedia content associated with this product (for example a photograph)  |
|  |                              | Delete a product from database   |
|  | Report Definition            | Define what the report will contain, for example a plot of production in time, plot of demand for some resource in time  |
|  |                              | Define how the data will be presented, for example bar chart, pie chart, table with adjustable columns/rows  |
|  |                              | Generate the desired report  |
|  |                              | Define the recipients and send the report  |
|  | Prediction Report Definition | Define what the report will contain, this can be either for example production in time or demand for some resource in time   |
|  |                              | Define for which resources, products or groups of resources or products the prediction should be made  |
|  |                              | Generate the desired prediction report   |
|  |                              | Define the recipients and send the prediction report   |
|  | Order Management             | Define how many and which resources are to be ordered  |
|  |                              | Send an order  |

## 4. Development Methodology

### 4.1. Methodology

For our project, the development methodology of choice was the Waterfall Development Model. This methodology was highly recommended to us.

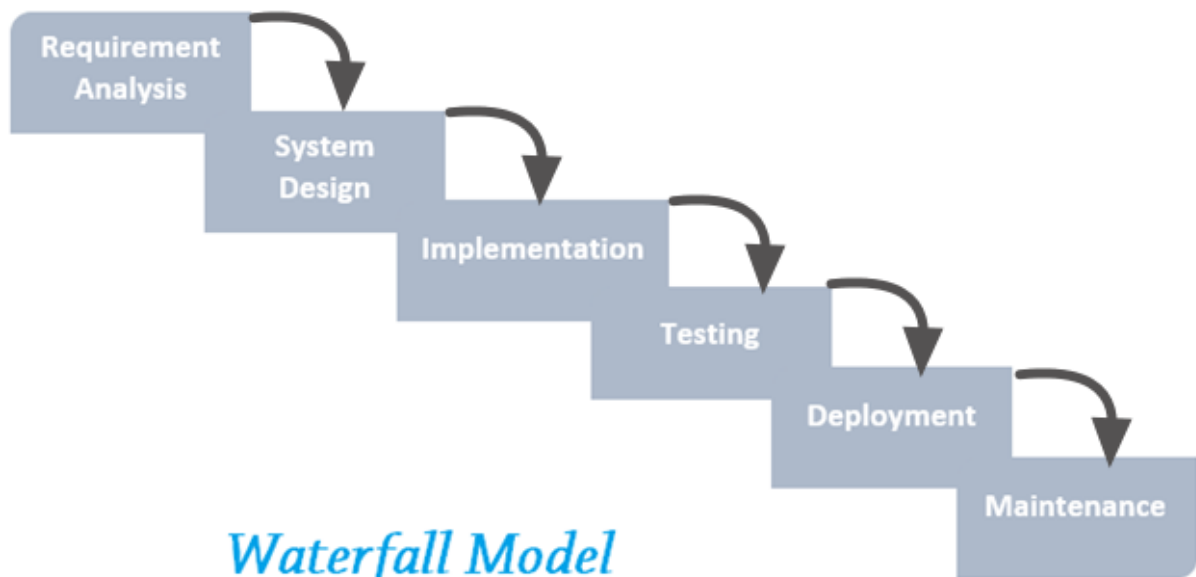


Figure 4.1: Waterfall Model

In Waterfall Model phases are executed sequentially, in linear way. We have a steady set of requirements, non changeable in time. The system is developed progressively and the user is involved only in the early phases Advantages of this model include it being easy to manage. Disadvantages are that it contains a strict sequence of activities, it has high cost of errors in beginning stages and high importance and cost of documentation and also a contact with the customer is weak. Therefore it can be used when it is possible to precisely define the requirements

#### 4.1.1. Argumentation

The reasons we have chosen this particular development model are presented below:

- The sequential order of phases matched perfectly the organized schedule of the development of our Engineering Thesis
- Fixed set of requirements, as the initial requirements were submitted by team to the Faculty and could not be changed later
- Easy management is facilitating our work as we work as a team and have no manager
- It was highly recommended by the coordinators of the Group Project

## 5. System Architecture

### 5.1. Software Architectural Pattern

To facilitate the design of classes, we have decided to follow a software architectural pattern. A pattern of choice here was Model-view-controller (MVC) Pattern. This pattern is used to separate a given application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to, and accepted from, the user.

- Model - Model represents an object carrying data. It can also have logic to update controller if its data changes.
- View - View represents the visualization of the data that model contains.
- Controller - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

As mentioned before, it has helped us develop the division into classes, visible on the class diagram below. LIME – is the program's main class. Then, the Model, View and Controller Classes were designed for each UI element. The database objects are handled by Servlet (classes taking an article from http POST and passing it to JDBC), DAO (classes responsible for the communication with database) and Query (classes to parse SQL) classes.

### 5.2. Backend Architectural Pattern

L.I.M.E. project base on many architectural patterns. Precisely chosen patterns make overall architecture strong and well built.

#### 5.2.1. Data Access Object Pattern

Data access object (DAO) is an object that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence



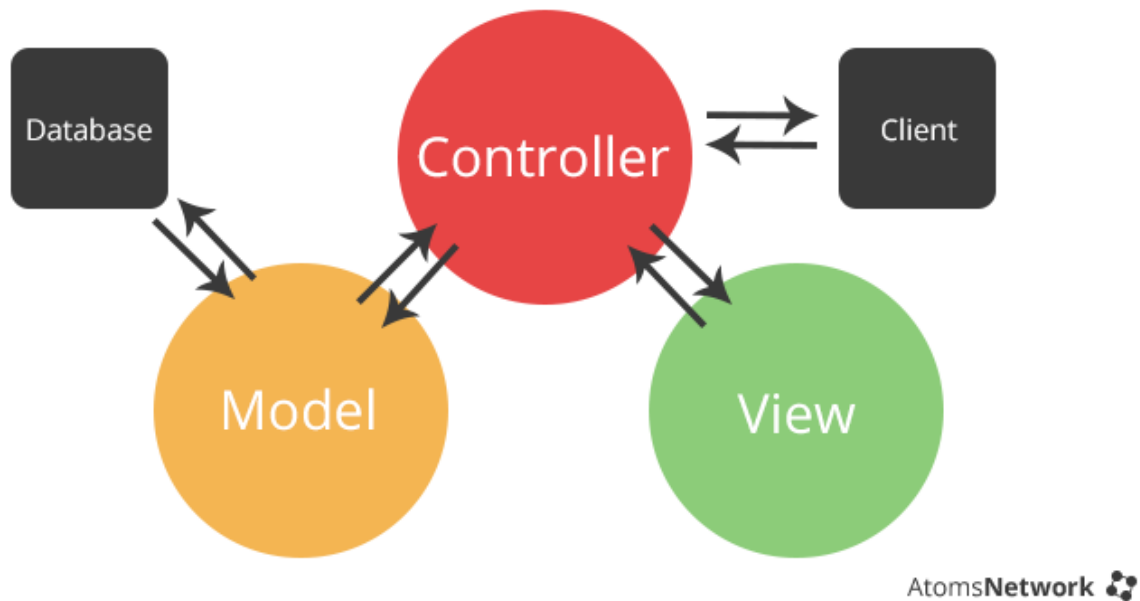


Figure 5.1: MVC Pattern

layer, the DAO provides some specific data operations without exposing details of the database. This isolation supports the Single responsibility principle. It separates what data access the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), from how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO). The most important fact and advantage is the relatively simple and rigorous separation between two important parts of an application that can but should not know anything of each other, and which can be expected to evolve frequently and independently. Changing business logic can rely on the same DAO interface, while changes to persistence logic do not affect DAO clients as long as the interface remains correctly implemented. All details of storage are hidden from the rest of the application (see information hiding). Thus, possible changes to the persistence mechanism can be implemented by just modifying one DAO implementation while the rest of the application isn't affected. L.I.M.E. implements one template parent interface with methods which are used by all DAO classes. `IBasicCrudRepository` extends `hibernate CrudRepository` and inherits from it all CRUD database operations. Singular DAO classes implement service-specific methods which are defined in DAO interfaces.

### 5.2.2. Plain Old Java Object

Plain old Java object (POJO) is an ordinary Java object, not bound by any special restriction and not requiring any class path. The main aim of the POJO classes is to differentiate business logic from database entity. It makes code cleaner and easier to read and understand.

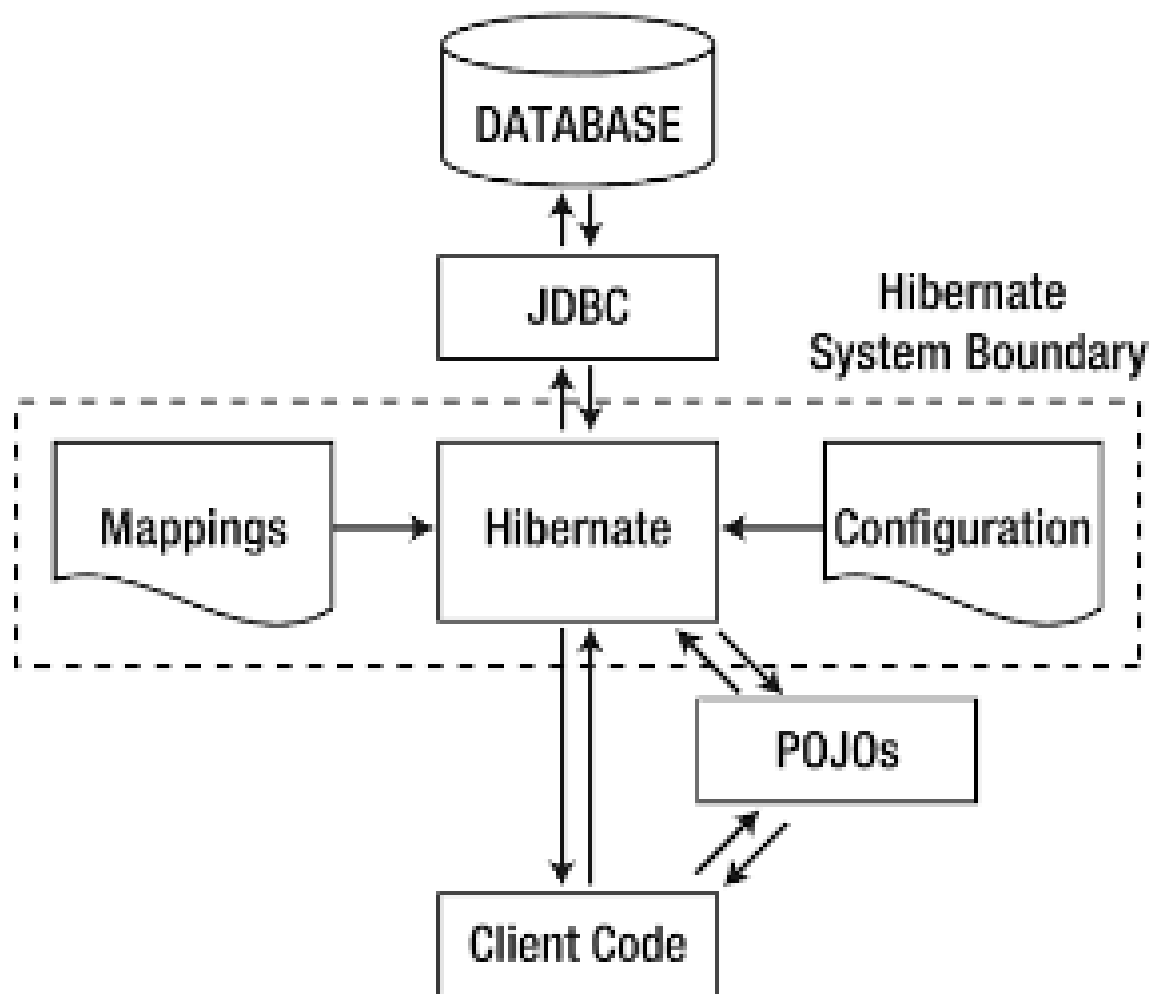


Figure 5.2: Plain Old Java Object in Backend Architecture

The POJO phenomenon has most likely gained widespread acceptance because of the need for a common and easily understood term that contrasts with complicated object frameworks.

L.I.M.E implements POJO classes for all existing entity class. Those classes are free from frameworks and complicated annotations. There is object mapper implemented, which maps directly Entity to POJO.

### 5.2.3. Dependency Injection

Dependency Injection is a kind of broader version of "inversion of control" (IoC) principle. It relates to the way in which an object obtains references to its dependencies - the object is passed its dependencies through constructor arguments or after construction through setter methods or interface methods. It is called dependency injection since the dependencies of an

object are 'injected' into it, the term dependency is a little misleading here, since it is not a new 'dependency' which is injected but rather a 'provider' of that particular capability. For example, passing a database connection as an argument to a constructor instead of creating one internal would be categorized as dependency injection. The pattern seeks to establish a level of abstraction via a public interface and to remove dependencies on components by supplying a 'plugin' architecture. This means that the individual components are tied together by the architecture rather than being linked together themselves. The responsibility for object creation and linking is removed from the objects themselves and moved to a factory.

The main advantage of Dependency Injection is fact that there is only one instance of each object shared through multiple controllers. What is more, it allows a client the flexibility of being configurable. Only the client's behavior is fixed. The client may act on anything that supports the intrinsic interface the client expects. Dependency injection can be used to externalize a system's configuration details into configuration files, allowing the system to be reconfigured without recompilation. Separate configurations can be written for different situations that require different implementations of components. This includes, but is not limited to, testing. L.I.M.E. uses spring dependency injection. There are bean services, which implements the domain logic. Each service implements interface defining the methods. Interfaces and services corresponds to hibernate entities and fulfill the needs of application logic. Interface beans are autowired into the REST controllers.

#### 5.2.4. RESTful API

REpresentational State Transfer (REST) is an architectural style that defines a set of constraints and properties based on HTTP. REST-compliant web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations. Other kinds of web services, such as WSDL and SOAP, expose their own arbitrary sets of operations. In a RESTful web service, requests made to a resource's URI will elicit a response that may be in XML, HTML, JSON, or some other format. The response may confirm that some alteration has been made to the stored resource, and the response may provide hypertext links to other related resources or collections of resources. When HTTP is used, as is most common, the operations available are GET, POST, PUT, DELETE, and other predefined CRUD HTTP methods.

L.I.M.E. implements RESTful controllers which are used to communicate with a client part of application. It communicates with JSON body type. There are CRUD used methods like PUT, DELETE, POST, GET to make action more readable and indicate what exactly it does. There

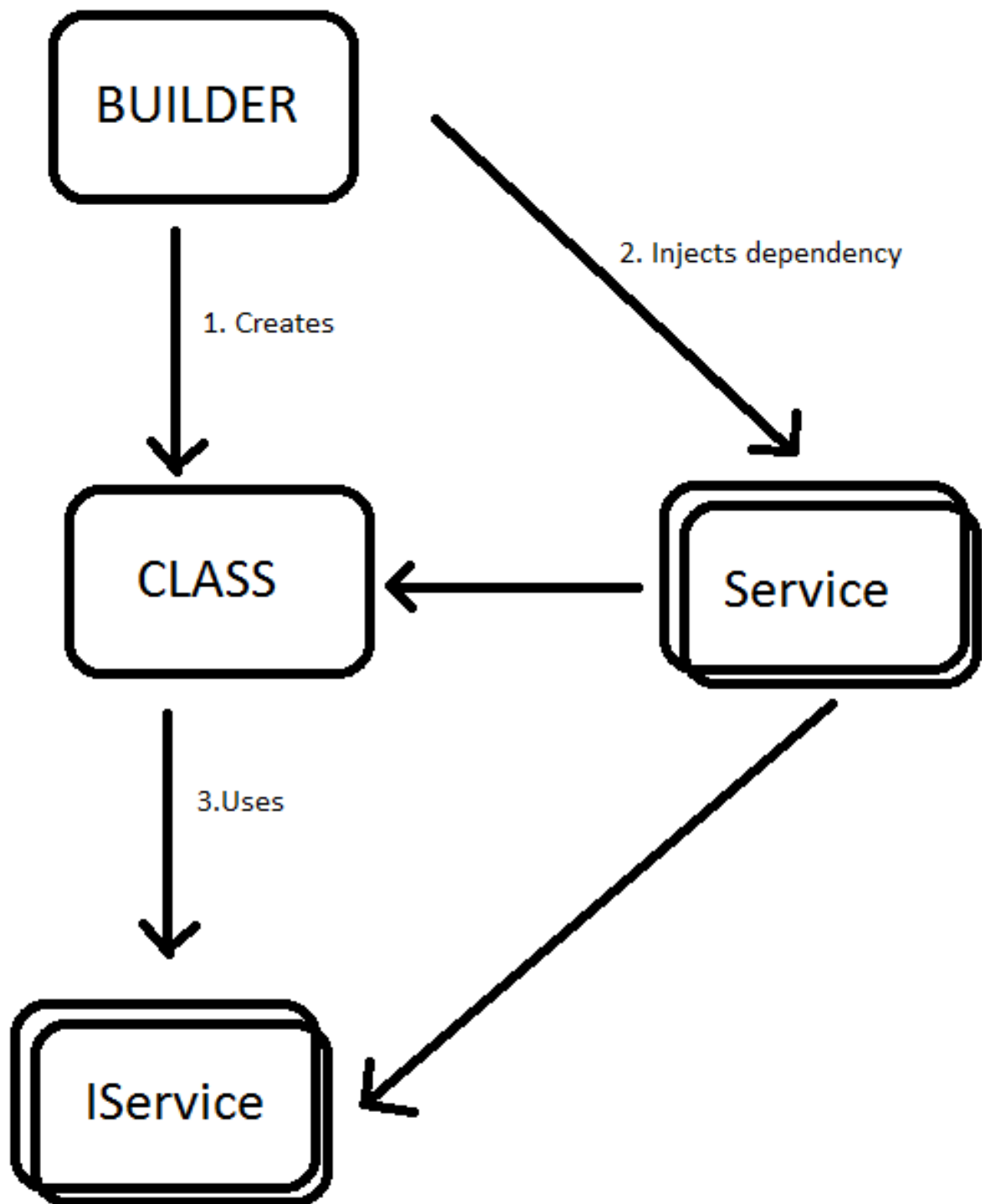


Figure 5.3: Dependency Injection

are Controllers corresponding to view in a client part. .

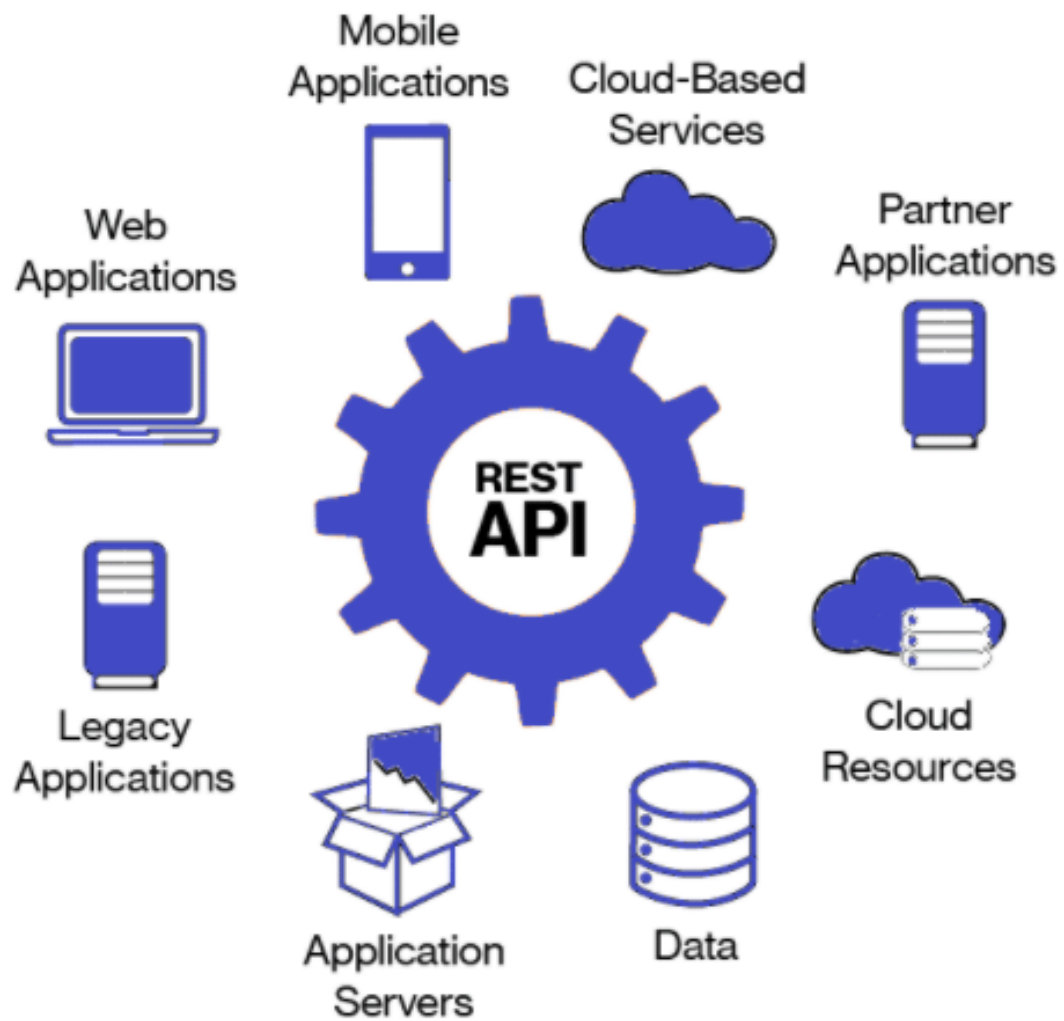


Figure 5.4: RESTful API

## 6. Technical Analysis

More detailed - all the classes and functions

### 6.1. Program Interface

### 6.2. System Classes

### 6.3. Database Design

The database consists of the following tables:

1. **Product** Stores information about products (complex structures of resources) with their properties:
  - (a) **Product ID** ID of a product
  - (b) **Added At** Date of adding a product to a database
  - (c) **Description** Description of a product
  - (d) **Expected Value** Expected value - how much of a product shall be obtained after production
  - (e) **Image** Image for this product
  - (f) **Name** Name of a product
  - (g) **Unit** Unit of a measurement
  - (h) **Category ID** Assigns a product to a category
2. **Product Category** Stores information about product categories: its name and ID
3. **Job** Logs the jobs performed by the user and the following details:
  - (a) **Job ID** ID of a job
  - (b) **Details** More details about the job

- (c) **Result Value** Result value - how much of a product was actually obtained after production
- (d) **Start Date** Start date of a job
- (e) **End Date** End date of a job
- (f) **Product ID** ID of a product that resulted from this job
- (g) **User ID** Assigns a job to a user which have performed it

4. **User** This table stores properties of every user of the system, such as:

- (a) **User ID** ID of a user

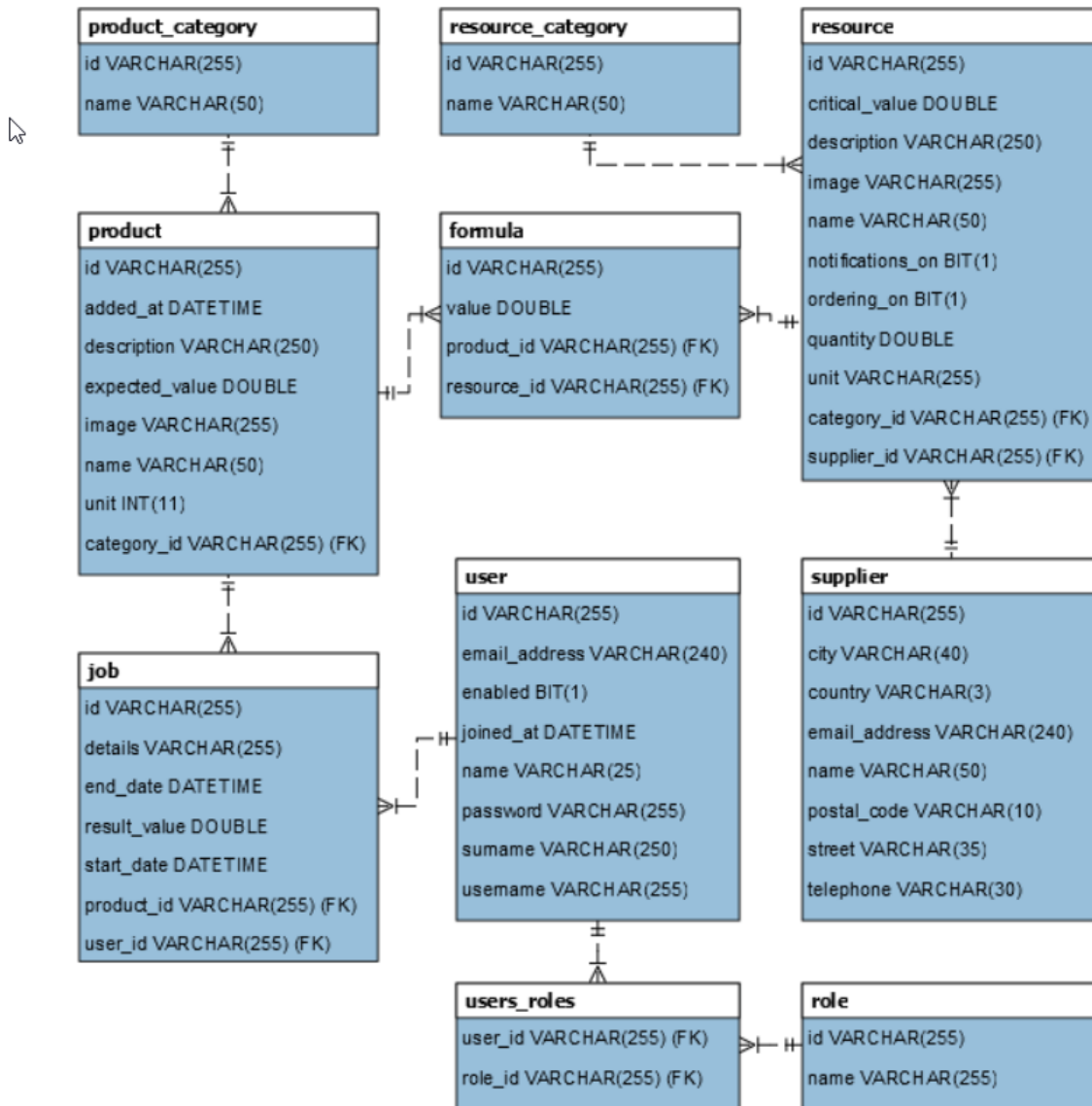


Figure 6.1: Database Design

### 6.3. DATABASE DESIGN

- (b) **Email address** Email address of this user
  - (c) **Enabled** Is this an active user of the system
  - (d) **Joined At** Date of joining the system
  - (e) **Name** Given Name
  - (f) **Surname** Surname
  - (g) **Username** Username chosen by the user
  - (h) **Password** User's password in encrypted form
5. **User Roles** Assigns users to their roles (resolves the many-to-many relationship)
6. **Role** Stores each role with its name and ID
7. **Formula** Resolves the many-to-many relation between resources and product, represents the quantity of resources used to make a product
- (a) **Formula ID** ID of a formula
  - (b) **Product ID** ID of a product
  - (c) **Resource ID** ID of a resource
  - (d) **Value** How much of a given resource is needed to make the given product
8. **Resource** Stores information about a basic laboratory resource and its properties, such as:
- (a) **Product ID** ID of a resource
  - (b) **Description** Description of a resource
  - (c) **Image** Image for the resource
  - (d) **Name** Name of a resource
  - (e) **Critical Value** critical value - how much of a resource shall be left for the notifications of the low level of resource to be triggered
  - (f) **Notifications On** Are the notifications about the low level of the resource turned on
  - (g) **Ordering On** Is automatic ordering of the resource turned on
  - (h) **Quantity** How much of a resource at the storage at the moment
  - (i) **Unit** Unit of a measurement
  - (j) **Category ID** Assigns a resource to a category
  - (k) **Supplier ID** Assigns a resource to a supplier



9. **Resource Category** Stores information about resource categories: its name and ID
10. **Supplier** Stores data about suppliers assigned to each resources
  - (a) **Supplier ID** ID of a supplier
  - (b) **Email address** Email address of this supplier
  - (c) **Name** Name of the supplier
  - (d) **Street** Address of the supplier: street name and number
  - (e) **City** Address of the supplier: City
  - (f) **Postal Code** Address of the supplier: Postal Code
  - (g) **Country** Address of the supplier: Country
  - (h) **Telephone** Telephone number of this supplier

#### 6.4. Prediction Model

## **7. Post Execution Documentation**

### **7.1. Evaluating Functional Requirements and Use Cases**

### **7.2. Evaluating Non-Functional Requirements**

## Conclusions

1 Page of Our Conclusions of Writing This Project

## Bibliography

- [1] A. Author, *Title of a book*, Publisher, year, page–page.
- [2] J. Bobkowski, S. Dobkowski, Title of an article, *Magazine X*, No. 7, year, PAGE–PAGE.
- [3] C. Brink, Power structures, *Algebra Universalis* 30(2), 1993, 177–216.
- [4] F. Burris, H. P. Sankappanavar, *A Course of Universal Algebra*, Springer-Verlag, New York, 1981.

## Glossary

**LIME**   Laboratory Internal Management Entity - the name of the application discussed in this document. 1

# List of Figures

|     |   |    |
|-----|---|----|
| 4.1 | Waterfall Model . . . . .                               | 21 |
| 5.1 | MVC Pattern . . . . .                                   | 24 |
| 5.2 | Plain Old Java Object in Backend Architecture . . . . . | 25 |
| 5.3 | Dependency Injection . . . . .                          | 27 |
| 5.4 | RESTful API . . . . .                                   | 28 |
| 6.1 | Database Design . . . . .                               | 30 |

If you don't need it, delete it.

**List of tables**

1.1 Work Division . . . . . 12

3.1 Functional Requirements . . . . . 14

3.2 Non-functional Requirements . . . . . 15

3.3 Uses Cases for Administrator . . . . . 16

3.4 Uses Cases for Manager . . . . . 18

3.5 Uses Cases for User . . . . . 19

If you don't need it, delete it.

## List of appendices

1. Appendix A. User Manual
2. Appendix B. Testing Scenarios



# A. User Manual

There we will have manual

## B. Test Report

Testi Report