НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В.Коваль
(підпис) (ініціали, прізвище)
“___”_____2019 р.

ДИПЛОМНА РОБОТА
на здобуття ступеня бакалавра

з напряму підготовки
6.050101 “Комп’ютерні науки’’

на тему: Розробка програмного агента моніторингу та управління теплонасосної установки

Виконав: студент 4 курсу, групи ТМ-51

Антонкін Дмитро Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник Ст. викладач Мірошниченко І.В. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 "Комп'ютерні науки''

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
"___"_____2019 р.

ЗАВДАННЯ

на дипломну роботу студенту

Антонкіну Дмитру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи "Розробка програмного агента моніторингу та управління теплонасосної установки "
керівник роботи Ст. викладач Мірошниченко І.В.
(прізвище, ім'я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від "___"_____ 201__ р. № _____

2. Строк подання студентом роботи ___ _____ 201__ р.
3. Вихідні дані до роботи звіт з моніторингу процесу роботи теплонасосної установки з інформацією про потенціал генеруваня теплової енергії
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та засоби моніторингу та упраління теплонасосної установки, розробити програмний агент, який у реальному часі може аналізувати та розраховувати потенціал сгенерованої теплової енергії обраного теплового насоса, кількість спожитої електроенергії за утримання теплонасосної установки, доцільність її використання за заданих умов.

_____
_____
_____

5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових креслень) 1. Актуальність 2. Мета та завдання роботи 3. Огляд існуючих рішень 4. Функції системи 5. Етапи розрахунку 6. Використанні програмні засоби 7. Висновки

Дата видачі завдання "___"_____ 201__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі		
2.	Розробка архітектури та загальної структури системи		
3.	Розробка структур окремих підсистем		
4.	Підготовка матеріалів		
5.	Програмна реалізація системи		
6.	Захист програмного продукту		
7.	Оформлення пояснювальної записки		
8.	Передзахист		
9.	Захист		

Студент _____ Антонкін Д.О
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мірошниченко І.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Пояснювальна записка містить 71 сторінок, включає 25 рисунків, 3 таблиці, 4 формули та 23 посилання.

Метою дипломної роботи є створення програмного агента управління та моніторингу теплонасосної установки. Було створено клієнт-серверний додаток на базі кроссплатформеної мови програмування Java, з використанням архітектурного фреймворку Spring MVC та реляційної бази даних PostgreSQL. Клієнт частину додатку було розроблено з використанням мови гіпертекстової розмітки HTML5, каскадної мови стилів CSS та веб-мови програмування JavaScript. Програмний агент було виконано у інтегрованому середовищі розробки IntelliJ IDEA компанії Jet brains.

Розроблено програмний агент моніторингу та управління теплонасосної станції має функції моделювання процесу генерації теплової енергії теплонасосною установкою впродовж довільного проміжку часу, функції збереження звіту процесу моделювання, можливість переглядати збережений звіт, функції конфігурування об’єкту моніторингу.

Ключові слова: програмний агент, Java, тепловий насос, теплонасосна установка, енергія, Spring, клієнт, сервер, PostgreSQL, реляційність.

ABSTRACT

The explanatory note contains 71 pages, including 25 figures, 3 tables, 4 formulas and 23 references.

The purpose of the thesis is to create a software agent for the management and monitoring of the heat pump installation. A client-server application based on the cross-platform Java programming language was created using the Spring MVC architectural framework and the PostgreSQL relational database. The client part of the application was developed using the HTML5 hypertext markup language, cascading CSS style language and JavaScript web programming language. The software agent was executed in the integrated development environment of the IntelliJ IDEA of Jet Brains.

The software agent of the monitoring and management of the heat pump station has developed the functions of modeling the process of generating heat energy by the heat pump installation during an arbitrary interval of time, the function of saving the report of the simulation process, the ability to view the saved report, the functions of the configuration of the monitoring object.

Keywords: software agent, java, heat pump, heat pump installation, energy, spring, client, server, postgreSQL, relational.

АННОТАЦИЯ

Пояснительная записка содержит 71 страниц, включает 25 рисунков, 3 таблицы, 4 формулы и 23 ссылки.

Целью дипломной работы является создание программного агента управления и мониторинга теплонасосной установки. Было создано клиент-серверное приложение на базе кроссплатформеного языка программирования Java с использованием архитектурного фреймворка Spring MVC и реляционной базой данных PostgreSQL. Клиент часть приложения была разработана с использованием языка гипертекстовой разметки HTML5, каскадного языка стилей CSS и веб-языка программирования JavaScript. Программный агент был выполнен в интегрированной среде разработки IntelliJ IDEA компании Jet brains.

Разработаный программный агент мониторинга и управления теплонасосной станции имеет функции моделирования процесса генерации тепловой энергии теплонасосной установкой в течение произвольного промежутка времени, функции сохранения отчета процесса моделирования, возможность просматривать сохраненный отчет, функции конфигурирования объекта мониторинга.

Ключевые слова: программный агент, Java, тепловой насос, теплонасосная установка, энергия, Spring, клиент, сервер, PostgreSQL, реляционность.

ЗМІСТ

ВСТУП

В наш час усе частіше постає питання про використання екологічно чистих та нешкідливих джерел енергії у тому числі теплової. Не так давно вчені енергетики зробили висновки, що джерелом теплової енергії може слугувати звичайне повітря, або енергія недр землі. Таким чином були винайдені теплонасосні установки, що у наш час активно замінюють застарілі твердопаливні котли.

На жаль в наш час не існує сервісу, який дозволив би обрати довільну теплову установку та змоделювати процесс її використання для певного проміжку часу. При цьому користувачеві необхідно було б охарактеризувати будівлю для моделювання.

Одними з найважливіших завдань дипломної роботи є розробка програмного агенту управління та моніторингу теплонасосною установкою.

Для вирішення даної проблеми було поставлено завдання розробити допоміжний інтерфейс, який би дав змогу звичайному користувачеві, знаючи паспортні характеристики будівлі, отримати у зрозумілому та лаконічному вигляді дані роботи теплового насосу, проаналізувати доцільність його використання за певних погодних та сезоних умов.

Під час проходження практики було створено програмний продукт, що має реляційну базу даних та в якому користувач має змогу вибрати теплонасосну установку, ввести необхідні дані для моделювання процессу її використання, отримати звіт з роботи, зберегти звіт на комп’ютер або до бази даних.

Для програмної реалізації було вирішено використати мову програмування Java для розробки серверної частини застосунку, фреймворк Spring для побудови архітектури системи та інтерфейс веб-сторінки реалізувати на мові розмітки HTML з використанням технології Freemarker, веб-мови программування JavaScript, та мови розмітки стилів CSS.

Для розгортання застосунку у мережі Інтернет було обрано використати технологію Maven, що дозволяє швидко встановлювати додаткові залежності до

проекту у вигляді сторонніх бібліотек, та швидко отримувати архівовані модулі проекту.

Особливу увагу приділено можливості розширення функціоналу даної системи сторонніми розробникам, для цього були використані технології, що стандартизують підхід до написання коду, такі як система контролю версій Git та використання структурних та поведінкових патернів.

1. ЗАДАЧА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АГЕНТА УПРАВЛІННЯ ТА МОНІТОРИНГУ ТЕПЛОНАСОСНОЮ УСТАНОВКОЮ У СКЛАДІ МУЛЬТИАГЕНТНОЇ СИСТЕМИ

Використовуючи теоретичний матеріал, наданий керівником дипломної роботи, реалізувати програмний агент для моніторингу та управління теплонасосною установкою, за допомогою якого можно проаналізувати доцільність використання теплонасосної установки у встановлений проміжок часу за певних температурних умов. Система повинна аналізувати продуктивність та коєфіцієнти завантаження теплонасосної установки, доцільність використання додаткових нагрівачів для вказаної площі та питомої тепловтрати приміщення.

В результаті розробки програмного продукту, користовачу буде надана можливість опрацьовувати дані різних температурних режимів, з урахуванням виробничих характеристик теплонасосної установки, прогнозувати її подальші режими роботи і отримувати найбільш ефективні стратегії її використання за певних температурних умов та характеристик будівель. За рахунок цього користувач зможе розрахувати та проаналізувати доцільність використання тієї чи іншої теплонасосної установки.

Розроблена система повинна забезпечувати наступні можливості:

- зчитування вхідних даних про температурний режим з Excel документа;
- калібрування системи відносно використовуємої теплонасосної установки;
- пошук оптимального варіанту теплонасосної установки для заданих умов;
- моделювання використання теплового насоса для заданих параметрів будівлі;
- отримання оптимальних показників щодо використання теплового насоса для заданих температурних умов;

- моделювання годинного температурного режиму впродовж заданого терміну;
- вивід та збереження графіків споживання електроенергії впродовж заданого терміну;
- вивід та збереження графіків спожитої та виробленої теплової енергії впродовж заданого терміну;
- порівняння ефективності використання теплових установок;
- збереження звіту з графіками роботи теплової установки.

Для розробки агенту мультиагентної системи була використана мова програмування Java. Розробка графічного інтерфейсу користувача відбувалась на основі Freemarker, Java Script, HTML5 та CSS.

Метою розробки інтелектуального агенту мультиагентної системи моніторингу та управління теплонасосною установкою є створення програмного продукту, що дасть змогу отримати дані, спираючись на які, користувач, маючи кілька параметрів будівлі, зможе оцінити раціональність використання теплонасосної уставноки за певних температурних умов. Це дасть зважено та чітко проаналізувати електроефективність роботи теплового насоса та додаткових догрівачів. Також за допомогою цієї системи користувач матиме можливість більш раціонально та відповідально вирішити питання теплового обігріву приміщення або будівлі.

Мета цього проекту полягає в тому, щоб дати змогу звичайному користувачу ознайомитись з ресурсами та продуцентами роботи теплового насоса. Цей проект дає змогу користувачам підібрати найліпший варіант обігріву їх оселі з врахуванням температурних режимів та параметрів будівлі.

2. ОГЛЯД ІСНУЮЧИХ АГЕНТІВ УПРАВЛІННЯ ТА МОНІТОРИНГУ ТЕПЛОНАСОСНОЮ УСТАНОВКОЮ

На жаль в наш час не існує сервісу, який дозволив би обрати довільну теплову установку та змоделювати процесс її використання для певного проміжку часу. При цьому користувачеві необхідно було б охарактеризувати будівлю для моделювання.

Існують електронні ресурси, які перевантаженні складними математичними формулами та фізичними термінами. Цей факт ускладнює вибір та відлякує потенційного користувача від переходу до використання «чистої» енергії нашої планети.

Провідні постачальники та виробники теплових насосів розміщують на своїх сайтах заклики зателефонувати або отримати консультацію від менеджера у офісі виробника – такий підхід не є раціональним, оскільки потребує часу, не є мобільним та як правило супроводжується додатковими витратами.

3. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ АГЕНТА

Одним із найважливіших завдань при розробці програмних продуктів є вибір таких засобів, які б полегшили роботу програміста, надавши всі необхідні інструменти для реалізації поставленого завдання, і дали б змогу отримати результат, який повністю задовольняє користувача.

При створенні програмного продукту було використано засоби реалізації, що зображені на рисунку 3.1.

Рисунок 3.1 — Засоби реалізації програмного забезпечення

Як зображено на рисунку 3.1, при створенні програмного забезпечення були використані такі засоби реалізації:

— середовище розробки IntelliJ IDEA, адже має найбільше функціоналу серед конкурентів;

— мову програмування Java для написання серверної частини;

— фреймворк Spring, а саме його модулі — Core Container, Web та Data

Access/Integration, для організації архітектури та взаємодії серверної частини з інтерфейсом та базою даних;

— мову розмітки HTML;

— генератор шаблонів Freemarker, що дозволяє створювати динамічні веб-сторінки без дублікації коду;

— мову розмітки CSS з використання бібліотеки Bootstrap для розробки графічного інтерфейсу користувача;

— технологію Maven для підключення всіх модулів та бібліотек проекту;

— Git для впровадження системи контролю версій та легкої інтеграції з хостинг сервісом Heroku через створений репозиторій проекту;

— Tomcat контейнер сервлетів, що дозволяє зменшити час на розгортання проекту на хостингу або у локальній мережі;

— реляційну базу даних PostgreSQL, вона є безкоштовную; має легкий та зрозумілий веб-інтерфейс та дуже просто інтегрується з середовищем розробки IntelliJ IDEA. Також безкоштовний хостинг Heroku, на якому розміщенно додаток пропонує користувачам безкоштовну базу даних PostgreSQL, що і стало остаточною перевагою.

Вибір технологій зумовлений тим, що програмний агент повинен працювати та підтримувати будь-які електро обчислювальні пристрої, а також не менш важливою є вимога до умов, в яких буде встановлена мультиагентна система та її складові компоненти, саме тому були обрані незалежні від операційної системи технології — Java, HTML, Maven та MySql.

Розробка веб-застосунку, окрім незалежності від платформи, обумовлена тим, що ми живемо у епоху швидкого Інтернету. Користувачу швидше та легше скористатися своїм телефоном або ноутбуком у будь-якому місці на планеті з підключеним інтернетом. Веб-агент дозволяє мінімізувати витрачений час на розробку на тестування, в той же момент розроблений додаток працюватиме на більшость існуючих пристроїв, що мають доступ до глобальної мережі.

Мовою програмування високого рівня була обрана Java, адже вона окрім, вже

зазначеної кроссплатформеності, має багато бібліотек, що допомагають спростити процес розробки, а також стандартизують архітектуру та стиль написання коду, що важливо для можливого розширення функціоналу іншими розробниками. Окрім цього Java є провідною мовою для розробки сучасних додатків на базі мікросервісної архітектури.

Базою даних була обрана PostgreSQL. PostgreSQL не просто реляційна, а об'єктно-реляційна СУБД. Це дає їй деякі переваги над іншими SQL базами даних з відкритим вихідним кодом, такими як MySQL, MariaDB і Firebird. Фундаментальна характеристика об'єктно-реляційної бази даних - це підтримка об'єктів і їх поведінки, включаючи типи даних, функції, операції, домени і індекси. Це робить PostgreSQL неймовірно гнучким і надійним. Серед іншого, він вміє створювати, зберігати та видавати складні структури даних.

3.1 Мова програмування Java

Уся логіка процесів системи написана мовою високого рівня Java — об’єктно-орієнтованою мовою програмування. Вибір зумовлений тим, що платформа Java має велику кількість технологій та фреймворків, що були створені для вирішення широкого спектру прикладних задач. Іншою перевагою Java є віртуальна машина, що забезпечує кросплатформеність. Головним підходом Java вважають – «Написати один раз і використовувати скрізь».

Окрім кросплатформеності сильною стороною цієї мови є також висока надійність роботи, адже вона розроблялася як об’єктно-орієнтована мова високого рівня з жорсткою типізацію. Так, для запобігання неоднозначності та спрощення розуміння коду з мови було виключено множинне наслідування. Замість цього, було введено поняття інтерфейсу, що являє собою, так званий контракт для класів, що будуть наслідуватися від нього. Іншим чинником високої надійності цієї мови програмування є система винятків та обробки помилок, які були поділенні на два види:

— виняткові ситуації, що можуть виникнути під час роботи програми, які обов’язково розробник має обробити, наприклад: користувач увів у поле у якому допустимі символи, або програма намагається відкрити файл на системі, якого не існує;

— ситуації, коли програма зустрічається з неочікуваними труднощами, наприклад: операція над елементами масиву поза його межами, або переповнення пам’яті через допущену помилку програмістом.

Java-програмісту не потрібно стежити за розподілом пам'яті, так як збирач сміття управляє пам'яттю автоматично. Збирач сміття запускається віртуальною машиною Java (JVM). Збирач сміття - це фоновий процес, який запускається періодично і звільняє пам'ять, використану об'єктами, які більше не потрібні. Об'єкт може підлягати утилізації в різних випадках:

— Якщо змінна посилального типу, яка посилається на об'єкт, встановлена в положення "0", об'єкт підлягає утилізації, в тому випадку, якщо на нього немає інших посилань;
— Якщо змінна посилального типу, яка посилається на об'єкт, створена для посилання на інший об'єкт, об'єкт підлягає утилізації, в тому випадку, якщо на нього немає інших посилань;
— Об'єкти, створені локально в методі, підлягають утилізації, коли метод завершує роботу, якщо тільки вони не експортуються з цього методу (тобто, повертаються або генеруються як виняток);
— Об'єкти, які посилаються один на одного, можуть підлягати утилізації, якщо жоден з них не доступний живому потоку.

Різні JVM мають відмінні один від одного алгоритми збору сміття. Існує кілька використовуваних алгоритмів, наприклад: алгоритм підрахунку посилань або алгоритми розмітки і очищення.

Об’єктно-орієнтований підхід до написання коду дозволяє оперувати поняттями, що зустрічаються в реальному житті з певною долею абстракії. Парадигма ООП наділила мову такими властивостями як масштабованість, що дає змогу неодноразово

розширювати розроблену систему. Розширюваність системи полягає в тому, що в систему можна додавати нові компоненти, без зміни вже існуючих. Іншою перевагою цього підходу є багаторазове використання написаного коду, що значно скорочує кількість написаного коду.

Наразі Java являє собою одну з найрозповсюдженіших мов програмування, адже може застосовуватися для написання багатьох типів програм.

3.2. Фреймворк Spring

Для створення архітектури систем з використання веб-інтерфейсу використовують сучасні бібліотеки, чи фреймворки, які спрощують процес написання коду.

Програмний фреймворк (англ. software framework) — це комплекс готових до використання програмних рішень, що включають в себе архітектуру побудови проекту, логіку та базову функціональність системи або підсистеми та, навіть, дизайн.

Фреймворк містить в собі набір бібліотек, що пропонують готовий набір рішень, також може містити допоміжні програми, скрипти та загалом все те, що полегшує створення та поєднання різних компонентів великого програмного забезпечення чи швидке створення готового і не обов’язково об’ємного програмного продукту. Одна з головних переваг використання фреймворків — це стандартизованість структури застосунку.

Spring — це фреймворк з відкритим вихідним кодом, що вільно розповсюджується, створений з метою спрощення розробки корпоративних додатків, забезпечує комплексну модель розробки і конфігурації для сучасних бізнес-додатків на Java. Ключовий елемент Spring — це підтримка інфраструктури на рівні програми: основна увага приділяється "водопроводу" бізнес-додатків, тому розробники можуть зосередитися на бізнес-логіці без зайвих налаштувань в залежності від середовища виконання [6]. Spring можна використовувати для побудови будь-якої програми на мові Java (тобто автономних, веб додатків, додатків JEE і т.д.), що позитивно відрізняє

Spring від багатьох інших платформ [7].

Для спрощення розробки додатків мовою Java у основі Spring лежать наступні стратегії:

— слабке зв’язування шляхом ін’єкції залежностей через конструктор, або сетери (англ. setter) та організація взаємодії через інтерфейси, цей підхід дозволяє легше тестувати застосунок, а також дозволяє координувати роботу кожного об’єкта в системі від третьої сторони;

— інверсія управління, що дозволяє писати незалежні один від одного компоненти, що надає переваги при розробці застосунку в команді, перенесенні та заміні модулей;

— декларативне програмування через аспекти та загальноприйняті угоди, що полегшує сприйняття коду;

— використання простих Java об’єктів, або їх ще називають POJO (Plain Old Java Object), без зобов’язання реалізовувати, чи наслідувати специфічні для фреймворку інтерфейси та класи відповідно;

— Spring виступає у ролі контейнера для об’єків, а отже, управляє їх життєвим циклом.

Ін’єкція залежностей (англ. Dependency Injection) — це паттерн проектування та архітектурна модель, що описує ситуацію, коли один об’єкт реалізує свій функціонал через інший об’єкт. Об’єкт передає керування створенням необхідних йому залежностей зовнішньому, спеціально призначеному для цього механізму. Наприклад, з’єднання з базою даних передається конструктору об’єкта через аргумент, замість того, щоб контруктор сам встановлював з’єднання. Перевагами цього підходу є те, що він дозволяє відділити об’єкти від реалізації механізмів, котрі він використовує, в наслідок чого, отримаємо гнучкість в розробці застосунку.

Spring складається з модулів (рисунок 3.3), що незалежні один від одного, окрім модуля Core Container, який являється ядром і до нього приєднують всі інші модулі. Цей модуль з чотирьох складових: Beans, Core, Context, SpEL [8].

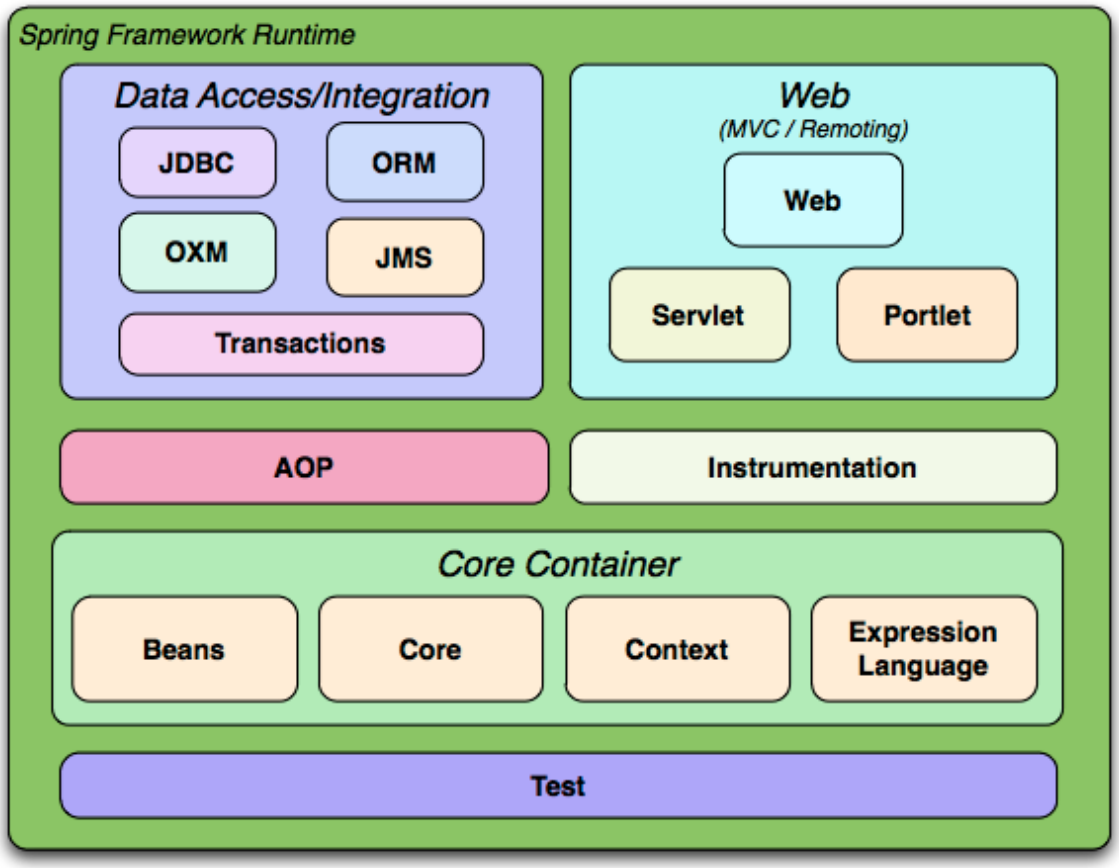

Рисунок 3.3 — Модулі Spring фреймворку

Beans та Core забезпечують найбільш фундаментальні частини фреймворку — ін’єкцію залежностей та інверсію управління, забезпечую реалізацію шаблона фабрики, що усуває необхідність реалізовувати паттерн Singleton та відокремлює конфігурацію з специфікацією від фактичної логічки програми. Context надає можливість отримувати доступ до об’єктів та додає підтримку інтернаціоналізації. SpEL (англ. Spring Expression Language) додає підтримку мов запитів, а саме: встановлення і отримання значень властивостей, зчитування з конфігураційного файлу, виклик методу, доступ до контексту масивів, колекцій та індексаторів, логічних і арифметичних операцій, пошук об'єктів по імені.

Модуль Data Access/Integration складається з компонентів, що відповідають за роботу з базою даних. Так, підмодуль ORM (англ. Object-relational mapping) надає шар програмної реалізації взаємодії з популярними бібліотеками для вирішення задач об’єктно-реляційного відображення. Наприклад, з: JPA, JDO, Hibernate та iBatus.

Підмодуль JDBC (Java DataBase Connectivity), що створює шар програмної реалізації однойменного стандарту взаємодії Java застосунку з різними СУБД. Підмодуль Transaction, як зрозуміло з назви — за підтримку та обробку транзакцій.

Модуль AOP додає підтримку аспектно-орієнтованого програмування в застосунок, що надає можливість ще гнучкіше налаштовувати його. Модуль Instrumentation надає можливість контролювати продуктивність застосунку. Модуль Test містить в собі бібліотеки для написання модульних та інтеграційних тестів.

Модуль Web забезпечує основний функціонал веб-застосунків. Компонент Web-Servlet надає реалізацію шаблону проектування архітектури проекту MVC (англ. Model View Controller — модель-представлення-контроллер) та HTTP протоколу для веб-застосунку. На рисунку 3.4 зображено архітектуру цього шаблону. За цим шаблоном забезпечується чітке розділення доменної (англ. domain) моделі, у якій міститься бізнес логіку від веб форм. Web Socket – модуль, що надає підтримку двошляхової комунікації між клієнтом та сервером.

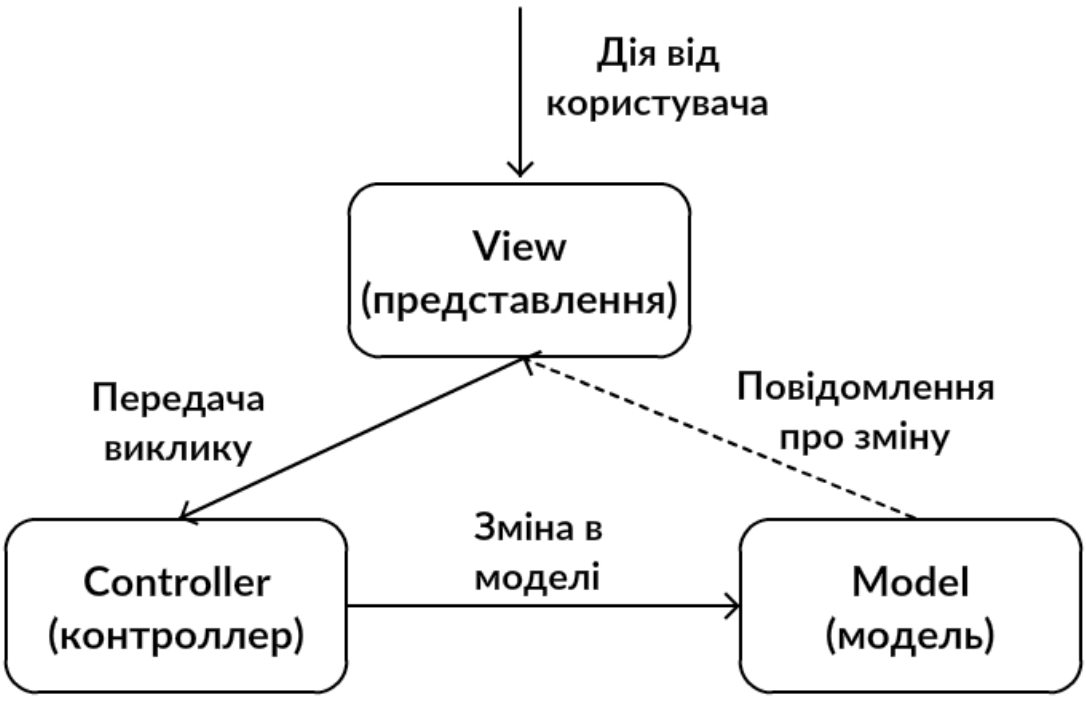

Рисунок 3.4 — Шаблон проектування MVC

В Spring Web MVC ви можете використовувати будь-який об'єкт в якості команди або об'єкта зі зворотним зв'язком; вам немає необхідності реалізовувати будь-якої спеціальний інтерфейс фреймворка або базовий клас. Завдяки гнучкості зв'язування даних в Spring, невідповідність типів розглядається як помилки валідації і тому це може бути оброблено в додатку, а не в якості системних помилок. Таким чином, вам не потрібно дублювати властивості бізнес-об'єктів, як простих нетипізований рядків для ваших об'єктів форм. Тому можна легко обробляти неправильні підтвердження або правильно конвертувати їх в рядки. Замість цього, бажано пов'язувати такі об'єкти безпосередньо з об'єктами бізнес логіки.

Цей модуль Spring побудований навколо центрального сервлету, який називається Dispatcher Servlet, котрий розподілює запити між контролерами. В ньому налаштовується обробка запитів, локалізація та часові зони [9]. Dsipatcher Servlet (з англ. Диспетчер сервлетів), це звичайний сервлет, що реалізує протокол HTTP. На рисунку 3.5 показана схема роботи цього сервлету з програмою. Спочатку запит потрапляє до Dispatcher Servlet, після чого він передає його у Handler Mapping, що переглядає усі наявні контролери та знаходить той, який знає як обробляти цей запит та повертає його ім’я назад до Dispatcher Servlet.

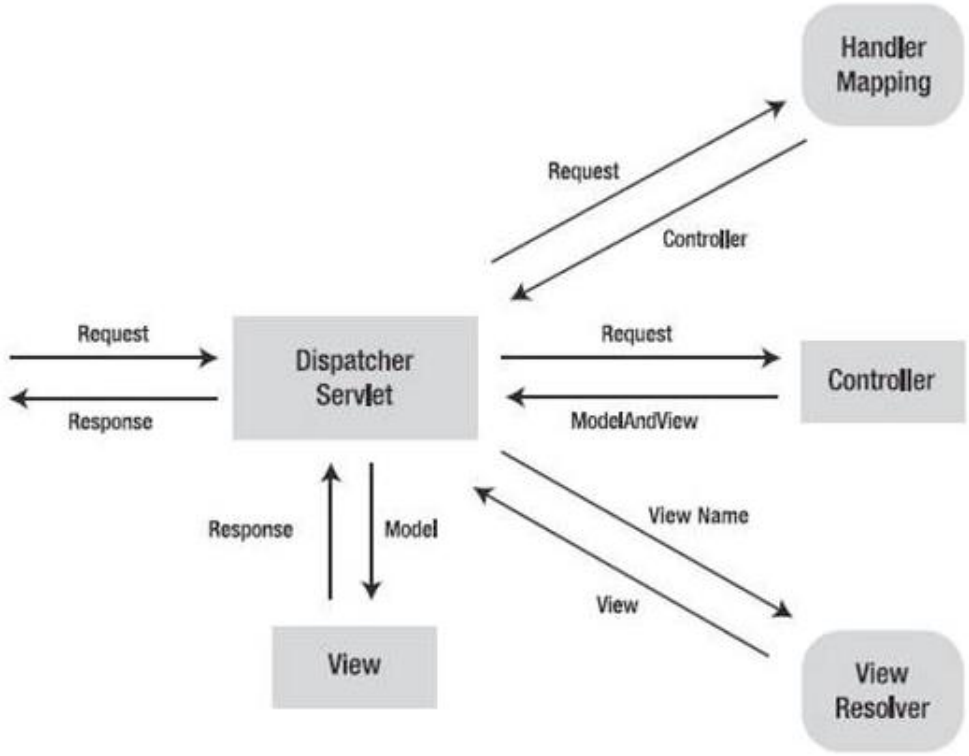

Рисунок 3.5 — Схема роботи Dispatcher Servlet

У ситуації коли буде знайдено два контролера, що мають необхідний функціонал обробки запиту неможлива, адже, це на етапі запуску проекту програмісту буде видана помилка. Після отримання імені контролера запит передається у нього. В контролері відбувається обробка запиту та назад відправляється об’єкт ModelAndView, що містить в собі дані та як їх потрібно відображати. Dispatcher Servlet на основі цього об’єкту шукає відповідне представлення (ім’я об’єкт типу View) за допомогою View Resolver. Завдяки цьому диспетчер сервлетів знає у яке представлення відправити модель (об’єкт типу Model), що містить дані. Від модуля представлення диспетчер сервлетів отримує відповідь, яка потім відсилається на веб-частину застосунку, у випадку коли вона необхідна.

Spring організовує архітектуру проекту шляхом зобов’язування програміста вказувати через анотації до якого типу класів відноситься даний, якщо ж не вказати тип, то фреймворк не буду поміщати даний клас у свій контейнер, а отже в ньому не будуть працювати методи з бібліотек, що надає Spring. Існує 4 анотації, що

утворюють архітектуру проекту:

— @Component позначається приналежність класу до фреймворку;

— @Controller позначається клас, в якому Dispatcher Servlet буде шукати метод, що оброблює запит;

— @Service позначається клас, в якому реалізується бізнес логіка, нічим кардинально не відрізняється від @Component, але дозволяє розробнику вказати смислове навантаження класу;

— @Repository позначається клас, що буде використовуватися для роботи з пошуком, отриманням та зберіганням даних. Здебільшого дані класи використовуються для організації взаємодії з базою даних та реалізації відповідних шаблонів.

У результаті використання даного фреймворку отримаємо архітектуру проекту як показано на рисунку 3.6.

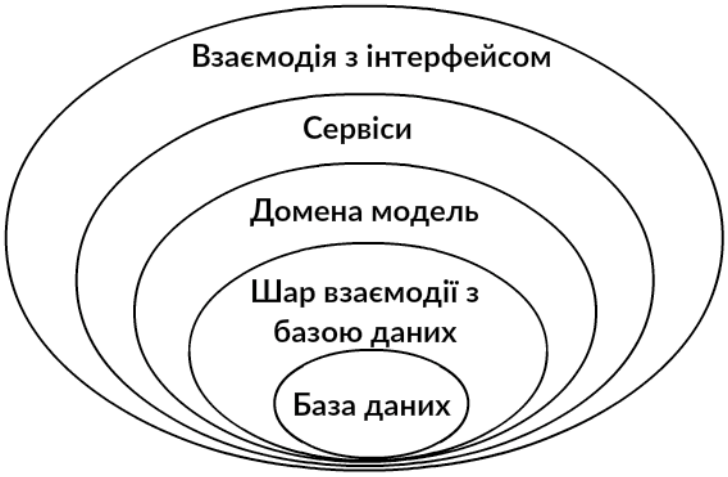

Рисунок 3.6 — Архітектура проекту з використання Spring фреймворку

3.3. Фреймворк Spring Data JPA

Фреймворк Spring Data JPA реалізує шар доступу до даних, який може бути досить громіздким. Занадто багато шаблонного коду пишеться для реалізації таких

завдань, як розбивка на сторінки і аудит. Spring Data JPA покликане значно поліпшити реалізацію шару доступу до даних, скоротивши зусилля на те, що дійсно необхідно. Як розробник, ви пишете інтерфейс сховища, включаючи власні методи пошуку, а Spring забезпечує їх автоматичну реалізацію [10]. Spring Data JPA надає ряд інтерфейсів, в яких уже описуються найбільш часто використовувані методи роботи з базою даних. Роздивимося на прикладі одного з основних інтерфейсів — JPARepository (рисунок 3.7).

Наслідуючи цей інтерфейс програміст отримує готову реалізацію методів збереження , оновлення та видалення даних з бази, а також методи пошуку всіх елементів з талички та за ід. Для того, щоб власний створений метод у інтерфейсі, що наслідую інтерфейс Spring Data JPA, виконував запити в базу, потрібно щоб назва методу будувалася зі слів, які зрозумілі декомпілятору даного фреймворку, або напряму через анотацію над методом вказувати SQL запит.

```
@NoRepositoryBean
public interface JpaRepository<T, ID extends Serializable> extends PagingAndSortingRepository<T, ID>
    List<T> findAll();

    List<T> findAll(Sort var1);

    List<T> findAll(Iterable<ID> var1);

    <S extends T> List<S> save(Iterable<S> var1);

    void flush();

    <S extends T> S saveAndFlush(S var1);

    void deleteInBatch(Iterable<T> var1);

    void deleteAllInBatch();

    T getOne(ID var1);

    <S extends T> List<S> findAll(Example<S> var1);

    <S extends T> List<S> findAll(Example<S> var1, Sort var2);
}
```

Рисунок 3.7 — Інтерфейс Spring Data JPA

Наприклад, метод з іменем: findAllByField(FieldType fieldValue) виконає запит до бази аналогічний до: “SELECE * FROM TABLE WHERE field = fieldValue” [11].

3.4. Фреймворк Hibernate

Hibernate є інструментом об'єктно-реляційного відображення для Java з відкритим вихідним кодом. Він надає фреймворк для відображення об'єктно-орієнтованої моделі даних в традиційні реляційні бази даних [12]. Hibernate не тільки дбає про відображення Java класів в таблиці БД (і типів даних Java в типи даних SQL), але також забезпечує запит даних і пошукові засоби і може значно скоротити час розробки який витрачається на написання SQL і JDBC коду вручну.

Зіставлення Java-класів з таблицями БД здійснюється за допомогою конфігураційних XML файлів, або за допомогою Java анотацій. Забезпечуються можливості по організації відносини між класами один-до-багатьох та багато-до-багатьох. В доповення до управління асоціацію між об'єктами, Hibernate може також управляти рефлексивними відносинами, де об'єкт має зв'язок один-ко-многим з іншими екземплярами свого типу.

Цей фреймворк не зобов’язує реалізовувати ніякі інтерфейси для своєї роботи, вся приналежність до фреймворку організовується через анотації, що вказують Hibernate як їх використовувати при зіставленні з базою даних. Під час виконання програми він зчитує ці анотації і використовує цю інформацію для того, щоб добудувати запити для відправки в деяку реляційну БД.

Hibernate являється однією з реалізацій JPA (Java Persistence API) — специфікація API Java EE, котра надає можливість зберігати об’єкти. Так, згідно неї, сутності — POJO-класи зв’язуються з БД за допомогою анотації @Entity, або через xml файл. Такий клас має задовольняти деяким умовам:

— повинна мати пустий конструктор;

— не може бути вкладена, інтерфейсом чи enum (перерахуванням);

— не може бути з final (константою) та мати final поля;

— мати хоча б одне поле помічене анотацією @Id.

— Даний фреймворк складається з таких компонентів:

— EntityManagerFactory — фабрика EntityManager, що забезпечує

екземпляри диспетчера об'єктів, всі екземпляри виконані з можливістю підключення до тієї же бази даних, щоб використовувати одні і ті ж параметри за замовчуванням, як це визначено в конкретної реалізації, і т.д. Ви можете підготувати кілька фабрик entity manager отримати доступ до кількох сховищ даних;

— EntityManager використовується для доступу до бази даних в конкретній одиниці роботи. Він використовується для створення та видалення стійких сутностей, щоб знайти об'єкти по їх первинному ключу ідентичності, і для запиту по всім організаціям;

— Persistence context являє собою набір екземплярів об'єкта, в якому для будь-якої постійної ідентичності об'єкта є унікальний екземпляр об'єкта. У persistence context, екземпляри сутностей і їх життєвий цикл управляється конкретним менеджером сутностей. Обсяг цього контексту може бути або транзакція, або розширена одиниця роботи.

3.5. Фреймворк Apache Maven

Apache Maven — це фреймворк для автоматизації збірки проектів на основі опису їх структури в файлах на мові POM (англ. Plain Object Model). Maven забезпечує декларативну, а не імперативну (на відміну від засобу автоматизації збирання Apache Ant) збірку проекту. У файлах опису проекту міститься його специфікація, а не окремі команди виконання. Всі завдання по обробці файлів, описані в специфікації, Maven виконує за допомогою їх обробки послідовністю вбудованих і зовнішніх плагінів. Використання maven зобов’язує використовувати стандартну структуру каталогів, що добре при спільній розробці. Так, у кореневому каталогу лежать тільки папки src, в якій знаходиться увесь код; папка target, в якій лежать всі створені під час роботи Maven файли та pom.xml файл у якому настроюється взаємодія Maven з проектом. В папці src лежать ще дві папки — main, в якій лежить реалізація програмного продукту та test — в якому лежать модульні та інтеграційні тести.

Maven керує життєвим циклом проекту. Він має набір фаз, що визначають

порядок дій при його побудові [13]. Maven містить три незалежних порядка виконання:

—clean — життєвий цикл для очищення проекту;

—default — основний життєвий цикл, що включає в себе такі фази як:

- validate, що виконує перевірку, чи є структура повною та правильною;
- compile, що компілює код програми;
- test, що запускає усі тести за папки test;
- install — установка програмного забезпечення в локальний Maven-репозиторій, щоб зробити його доступним для інших проектів поточного користувача.
- deploy — стабільна версія програмного забезпечення поширюється на віддалений Maven-репозиторій, щоб зробити його доступним для інших користувачів.

—site — життєвий цикл генерації проектної документації.

3.6. Мова розмітки HTML

Для відображення результату роботи методів, що реалізують алгоритм системи обліку та оперативного зберігання проектно-технологічної документації енергетичних споруд було створено Web-сторінку, написану на HTML (англ. HyperText Markup Language — мова розмітки гіпертекстових документів) — стандартна мова розмітки веб-сторінок в Інтернеті [14].

Більшість веб-сторінок створюються за допомогою мови HTML (або XHTML). Документ HTML оброблюється браузером та відтворюється на екрані у звичному для користувача вигляді.

Разом із каскадними таблицями стилів та вбудованими скриптами, HTML являється одною з основних технологій для побудови веб-сторінок.

Мова розмітки HTML впроваджує засоби для:

— створення структурованого документа шляхом позначення структурного

складу тексту: заголовки, абзаци, списки, таблиці, цитати та інше [15];

— отримання інформації з інтернету через гіперпосилання;

— створення інтерактивних форм;

— включення зображень, звуку, відео, та інших об’єктів до тексту.

Розмітка в HTML складається з чотирьох основних компонентів: елементів (та їхніх атрибутів), базових типів даних, символьних мнемонік та декларації типу документа. Елементи являють собою базові компоненти розмітки HTML. Кожен елемент має дві основні властивості: атрибути та зміст (контент). Існують певні настанови щодо кожного атрибута та контента елемента, які треба виконувати задля того, щоб HTML-документ був визнаний валідним.

Правильний семантичний HTML також покращує доступність веб-документів. Наприклад, коли браузер або аудіо-браузер може правильно встановити структуру документа, він не буде витрачати час користувачів з вадами зору, на прочитання повторюваної або неактуальної інформації, якщо вона була розмічена правильно.

Для перегляду HTML-розмітки документа можна використовувати будь-який текстовий редактор. Для перегляду документу, відтвореного за правилами HTML-розмітки, використовується браузер.

3.7. Мова розмітки CSS

Каскадні таблиці стилів (англ. Cascading Style Sheets, або скорочено CSS) — спеціальна мова, що використовується для опису сторінок, написаних мовами розмітки даних.

Найчастіше CSS використовують для візуальної презентації сторінок, написаних HTML та XHTML, але формат CSS може застосовуватися до інших видів XML-документів. CSS це мова стилів, що визначає відображення HTML-документів. Наприклад, CSS працює з шрифтами, кольором, полями, рядками, висотою, шириною, фоновими зображеннями, позиціонуванням елементів і багатьма іншими речами.

Специфікації CSS були створені та розвиваються Консорціумом Всесвітньої мережі.

CSS використовується авторами та відвідувачами веб-сторінок, щоб визначити кольори, шрифти, верстку та інші аспекти вигляду сторінки. Одна з головних переваг — можливість розділити зміст сторінки (або контент, наповнення, зазвичай HTML, XML або подібна мова розмітки) від вигляду документа (що описується в CSS) [16].

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо.

Один і той самий HTML або XML документ може бути відображений по-різному залежно від використаного CSS.

Основними перевагами мови CSS є:

— інформація про стиль для усього сайту або його частин може міститися в одному .css-файлі, що дозволяє швидко робити зміни в дизайні та презентації сторінок;

— різна інформація про стилі для різних типів користувачів: наприклад великий розмір шрифту для користувачів з послабленим зором, стилі для виводу сторінки на принтер, стиль для мобільних пристроїв;

— сторінки зменшуються в об’ємі та стають більш структурованими, оскільки інформація про стилі відділена від тексту та має певні правила застосування і сторінка побудована з урахуванням їх

— прискорення завантаження сторінок і зменшення обсягів інформації, що передається, навантаження на сервер та канал передачі. Ця перевага досягається за рахунок того, що сучасні браузери здатні кешувати (запам’ятовувати) інформацію про стилі і використовувати для всіх сторінок, а не завантажувати для кожної;

— управління відображенням безлічі документів за допомогою однієї таблиці стилів;

— більш точний контроль над зовнішнім виглядом сторінок;

— поданням для різних носіїв інформації (екран, друк, і т. д.);

— складна і пророблена техніка дизайну.

Для пришвидшення розробки дизайну було використано вже готовий набір стилів Bootstrap 3 [17].

3.8. Менеджер сервлетів Apache Tomcat

Контейнер сервлетів - це програма, що представляє собою сервер, який займається системною підтримкою сервлетів і забезпечує їх життєвий цикл відповідно до правил, зазначених в їх специфікаціях. Може працювати як повноцінний самостійний веб-сервер, бути постачальником сторінок для іншого веб-сервера, наприклад Apache, або інтегруватися в Java EE сервер додатків. Забезпечує обмін даними між сервлетом і клієнтами, бере на себе виконання таких функцій, як створення програмного середовища для функціонуючого сервлету, ідентифікацію та авторизацію клієнтів, організацію сесії для кожного з них.

Apache Tomcat (також називається Tomcat Server) реалізує кілька специфікацій Java EE, включаючи Java Servlet, JavaServer Pages (JSP), Java EL і WebSocket, і забезпечує середовище HTTP "чистого Java", в якому може працювати код Java.

Tomcat розробляється і підтримується відкритою спільнотою розробників під егідою Apache Software Foundation, що випускається під ліцензією Apache License 2.0 і є відкритим програмним забезпеченням.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ

Для моніторингу за теплонасосною установкою необхідно змоделювати процесс її роботи за деяких температурних умов, з вибором конкретного теплового насоса. Окрім цього необхідно мати уяву про об’єкт у якому буде працювати теплонасосна установка, а саме деякі її паспортні характеристики, такі як: площа об’єкту, величина питомої тепловтрати будівлі, необхідний температурний режим.

Вибір клієнт-серверної архітектури обумовлений тим, що дана архітектура дозволяє організовувати систему, яка дозволить централізовано змінювати асортимент можливих теплових насосів у системі, здійснювати двох-факторний контроль над данними користувача, відокремити до швидкодіючого та продуктивного серверу навантажену частину фізичних та математичних розрахунків. Для того, щоб сервер можна було розгорнути на будь-якій операційній системі було запропоновано обрати кроссплатформену мову програмування — Java.

Факт, що клієнт частина відокремлена від основної логіки програми на сервері, дає змогу розробнику репрезентаційної частини не витрачати час на додаткові обчислення чи перевірки у виглядовій частині, а приділити більше часу саме на зовнішній вигляд та фізичний інтерфейс майбутнього користувача. Для того аби кінцевий продукт роботи виглядав не тільки інформативно, але й лаконічно та інтерактивно були обрані графічні бібліотеки мови JavaScript та заздалегідь створені каскадні CSS стилі фреймворку Bootstrap.

Вибір реляційної бази даних обумовлений тим, що предметна область диктує використання об’єктів звичного для людини простору. Так за допомогою реляційних відносин досить легко реалізувати зв’язки між майбутнім звітом про роботу теплонасосної установки та потенційним клієнтом. Тим паче що, використані фреймворки Spring та Hibernate сприяють реляційно-об’єктному підходу вирішення проблеми дипломної роботи.

Для організації архітектури програмного продукту було вирішено використати фреймворк Spring та його модулі, адже такий підхід дозволяє розробити гнучкий

програмний продукт з чітко виділеними шарами, реалізацію яких можна змінювати без зміни функціональності застосунку.

4.1. Структура програмного забезпечення

Для реалізації задачі моніторингу та управління теплонасосною установкою у складі мультиагентної системи був розроблений клієнт-серверний агент, сервер якого може бути розгорнутий у будь-якому інформаційному середовищі, а у якості клієнтського інтерфейсу виступає браузер користувача. На рисунку 4.1 представлений інтерфейс головної сторінки для користувача, що цікавиться у використанні теплонасосної установки.

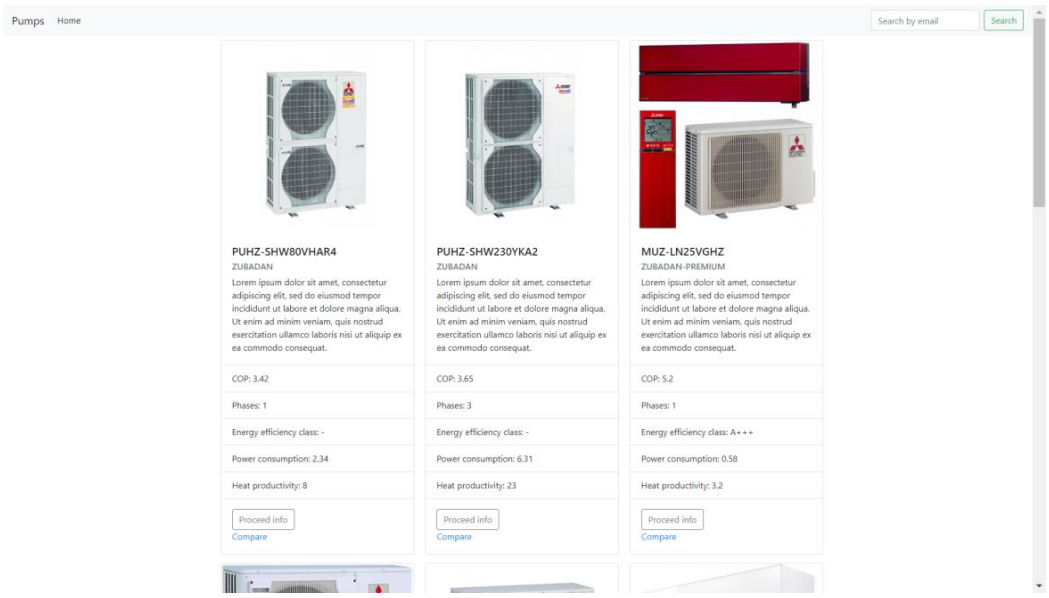

Рисунок 4.1 — Головна сторінка для користувача

Програмний продукт було реалізовано з використанням Spring фреймворку, це означає, що він має чітко розшаровану архітектуру, де кожен компонент може бути змінений або модернізований, а також згідно Maven фреймворку організація архітектури проекту має структуру директорій, що показано на рисунку 4.2.

Структура проекту складається з наступних директорій-модулей: src, target, .idea, External Libraries та файлів .gitignore, gs-serving-web-content.iml та pom.xml.

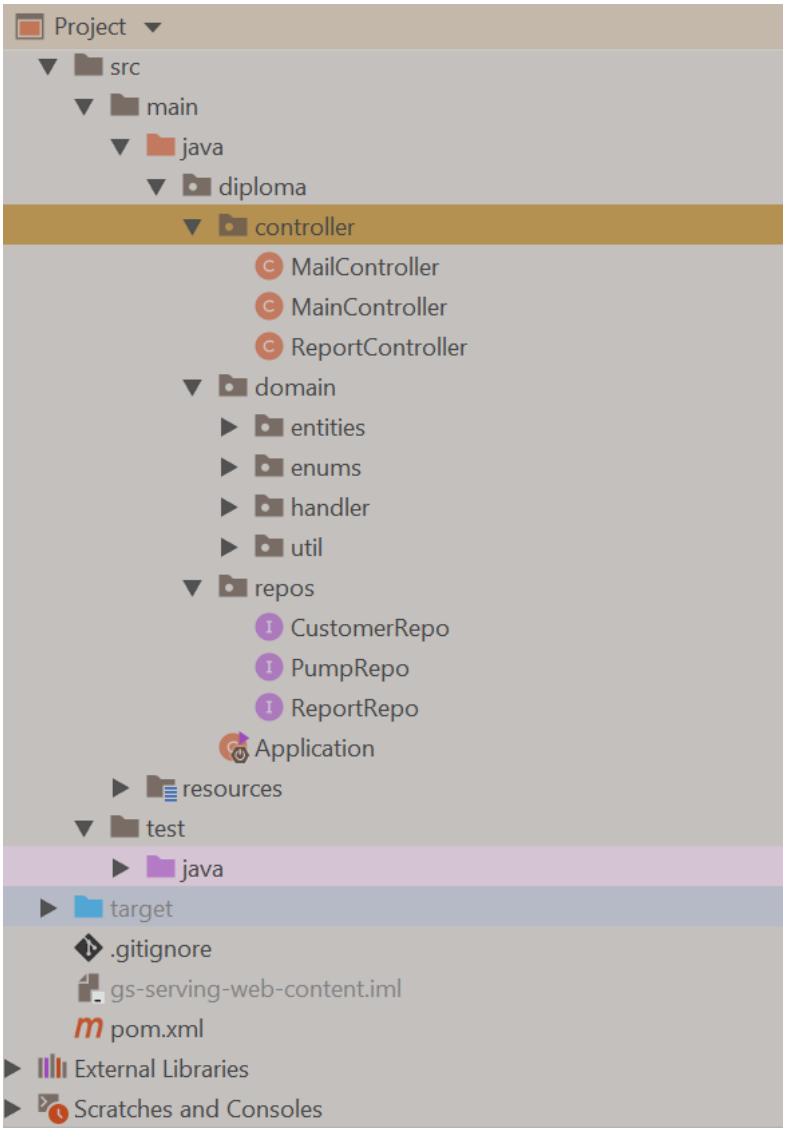

Рисунок 4.2 — Структура файлів проекту

Папка src поділяється на main та test, а main у свою чергу на java, та resources. Модуль src містить усю логіку застосунку.

Модуль target містить скомпільований проект, що містить у собі скомпільовані .class файли проекту та веб-архів з розширенням файлу .war, що передається на керування менеджеру сервлетів Apache Tomcat, який у свою чергу розгортає та встановлює його.

Файл pom.xml — це основний файл проекту, що використовує Maven фреймворк, у ньому знаходиться основна інформація про проект, підключаються бібліотеки, підключаються плагіни необхідні та визначається порядок компіляції модулів.

Директорія .idea — зберігає в собі усі налаштування відкритого проекту при розрозбці. До таких налаштувань можуть належати комбінації гарячих клавіш, плагіни для розробки у середовищі, налаштування зовнішнього вигляду середовища розробки і т.д [19].

Файл з розширенням .iml створюється автоматично при утворені нового проекту. Він зберігає інформацію про модуль розробки, який може бути компонентом Java, Plugin, Android або Maven; зберігає шляхи модуля, залежності та інші параметри.

External Libraries — це модуль, що містить набір підключених сторонніх бібліотек. У випадку даної реалізації до цього модулю потрапляють файли бібліотек, які були підключені до проекту у файлі pom.xml. Саме фреймворк Maven займається скачуванням цих бібліотек та їх розміщенням у структурі проекту.

Файл .gitignore — це файл плагіну, який додається розробником. Він слугує допоміжним механізмом підтримки системи контролю версій, а саме якщо програміст вважає, що стан деяких файлів проекту впродовж етапу розробки не змінюється, або зміни цих файлів не впливають на кінцеву роботу програми – такі файли додаються у список ігнорованих. Якщо файл зазначений флагом «ігнорувати» система контролю версій припине його версіоність.

Як вже було сказано вище, модуль src містить модуль main, який поділяється ще на два модулі:

— папка java містить код серверної частини, який написано на мові програмування Java. В свою чергу папка java поділена ще на 3 папки;

— папка resources містить основний конфігураційний файл проекту application.properties – цей файл містить в собі налаштування бази данних, конфігурацію шарів фреймворку Spring та інші важливі параметри системи. Окрім цього resources включає в себе статичні та динамічні файли репрезентаційної частини фінального продукту, так у папці data зберігається файл temperature.xlsx – що є необхідною частиною математичних розрахунків, папка static містить у собі зображення та JavaScript файли, папка templates містить у собі ієрархію папок та файлів .ftl формату, що у свою чергу є продуцентами Freemarker фреймворку, що

компілює та комбінує веб-сторінки програмного агента.

Відповідно до архітектури Spring MVC фреймворку сервера частина коду програмного агенту поділяється на такі компоненти: controller, domain та repos. У директорії controller знаходяться всі контролери системи з відповідними анотаціями при оголошені класу. Саме за допомогою цих аннотацій Spring MVC фреймворк розуміє як розрізняти компоненти всього програмного агенту.

Так само у директорії domain, а саме у дочірній директорії entities містяться класи, що описують собою сутності системи, так само як і контролери, вони позначені спеціальними анотаціями Spring MVC фреймворку.

Оскільки об’єктно-реляційну модель програмного агента пободовано на базі фреймворку Spring Data JPA компонент з назвою repos є вхідною точкою для створення відповідних репозиторіїв для кожної з сутностей системи. З технічної документації відомо, що фреймворк Spring Data JPA підтримує 2 механізми оголошеня репозиторіїв – xml розмітка та java класи, якщо бути точнішим, то java інтерфейси у випадку програмного агента моніторингу та управління теплонасосною установкою.

Отже, серверна частина повністю написана об’єктно-орієнтованою строготипізованою мовою програмування високого рівня Java, а інтерфейс користувача — виконано з використанням таких технологій: HTML мова гіпертекстової розмітки, CSS – каскадних стилів цієї розмітки, JavaScript мови програмування, що допомагає оживити сайт з точки зору інтерфейсу користувача.

Схема взаємодії модулів у системі зображена на рисунку 4.3.

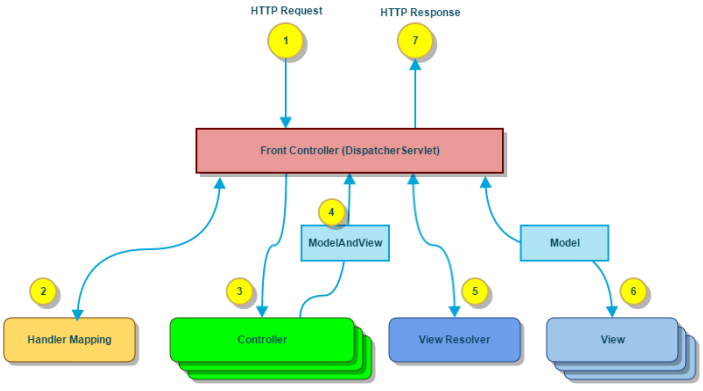

Рисунок 4.3 — Схема взаємодії модулів

Як видно зі схеми, взаємодія клієнта з сервером відбувається на основі HTTP протоколу.

4.2. Опис структури бази даних

Як було сказано у попередніх розділах відповідальність за створення моделі бази даних було перекладено на фреймворки Spring Data JPA та Hibernate. Так наприклад для генерації таблиці report, що відповідає за збереження даних сутності звіту достатньо було оформити конфігурацію класу Report.java як зображено на рисунку 4.4:

```
@Entity
public class Report {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "customer_id")
    private Customer owner;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "pump_id")
    private Pump pump;

    @Enumerated(EnumType.STRING)
    private TemperatureMode temperatureMode;

    @ElementCollection
    @CollectionTable(
            name = "Norm_Heat_Prod",
            joinColumns = @JoinColumn(name = "normHeatProd_id")
    )
    private List<Double> normalizedHeatProductivity = new ArrayList<>();

    @ElementCollection
    @CollectionTable(
            name = "Coef_Heat_Prod",
            joinColumns = @JoinColumn(name = "coefHeatProd_id")
    )
    private List<Double> heatProductivityCoefficients = new ArrayList<>();
```

Рисунок 4.4 — Класс Report.java з конфігурацією

Ділянки коду, що починаються с символу «@» називають анотаціями. Java-анотація – в мові Java спеціальна форма синтаксичних метаданих, яка може бути додана в вихідний код. Анотаціями можна позначити пакети, класи, методи, змінні і параметри. Анотації використані у даному класі Report.java належать до пакету javax.persistence, що у свою чергу є складовою частиною Spring Data JPA.

Кожна з використаних анотацій несе в собі ключові метадані для компілятора jvm, який у свою чергу допомагає фреймворку Hibernate з роботою над створеними таблицями.

Наприклад анотація @ManyToOne відповідно до її назви встановлює реляційні зв’язки між таблицями, а точніше зв’язок типу багато до одного.

Анотація @Enumerated з параметром EnumType.String вказує що тип колонки з назвою temperatureMode у таблиці report матиме строковий тип даних у майбутній таблиці базі даних.

Використання у парі параметризованих анотацій @ElementCollection та @CollectionTable з указаними параметрами приведе до створення таблиці з указаною назвою та з реляційним зв’язком багато до одного.

Таким чином, після недовгих маніпуляцій та не написавши жодного рядка SQL коду за допомогою вище названих фреймворків Spring Data JPA та Hibernate маємо структуру бази даних, що зображена на рисунку 4.5:

Такий самий підхід було використано для генерації інших таблиць програмного агента. Слід приділити увагу налаштуванням бази даних системи. А саме JPA має функції для генерації DDL, і вони можуть бути налаштовані для ініціалізації під час запуску програми усупереч з базою даних. Як було сказано у попередніх розділах налаштування бази даних системи розміщено у файлі application.properties, а саме конфігурація двох вказаних параметрів:

— spring.jpa.hibernate.ddl-auto (enum) - це функція Hibernate, яка керує поведінкою більш дрібним способом.

— spring.jpa.generate-ddl (boolean) вмикає та вимикає цю функцію і не залежить від постачальника.

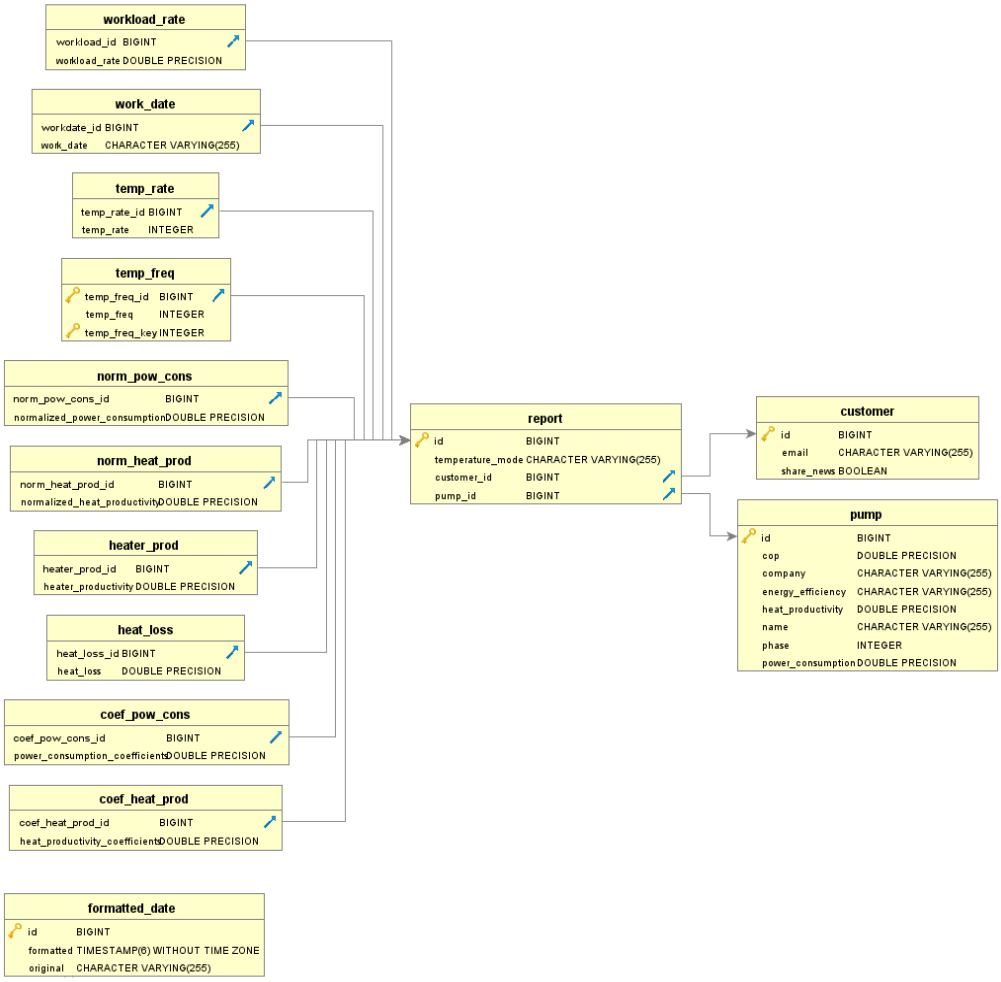

Рисунок 4.5 — Структура бази даних програмного агента

4.3. Опис алгоритму розрахунку потенціалу генерування теплової енергії в реальних умовах

Теплова енергія - форма енергії, пов'язана з рухом атомів, молекул або інших частинок, з яких складається тіло. Теплова енергія - неточний термін. Теплота, як і робота є не видом енергії, а лише способом її передачі. По суті, теплова енергія - це сумарна кінетична енергія структурних елементів речовини (будь то атоми, молекули або заряджені частинки). Теплова енергія системи плюс потенційна енергія міжатомних взаємодій називається внутрішньою енергією сістеми. Тепловая енергія вимірюється в джоулях-ДЖ (в СІ).

Теплова енергія може виділятися завдяки хімічним реакціям (горіння), ядерних реакцій (поділ ядра, ядерний синтез), механічним взаємодій (тертя). Теплота може передаватися між тілами за допомогою теплопровідності, конвекції або випромінювання.

Земля, зокрема, володіє гігантськими запасами енергії. У декількох метрах нижче її поверхні вона зберігає сонячне тепло. З ядра Землі температури величиною 6500 градусів Цельсія випромінюються в її зовнішні шари. Теплові насоси використовують геотермальне тепло або тепло грунтових вод в залежності від технології. Енергія, накопичена в навколишньому повітрі, також підходить для обігріву приміщень і виробництва гарячої води. Теплові насоси можуть використовувати ці ресурси і, таким чином, істотно знижують витрати на виробництво теплової енергії.

Не залежно від того, яка технологія використовується, теплові насоси ефективно працюють навіть при низьких температурах навколишнього середовища. До 75 відсотків ваших потреб в тепловій енергії можуть бути отримані безпосередньо з навколишнього середовища і безкоштовно. Тільки 25 відсотків повинні бути додані у вигляді електричної енергії.

Як правило, виробники теплових насосів у характеристиках установок указають лише приблизні значення виробляємої теплової енергії при температурі 17 градусів за Цельсієм, оскільки така температура вважається середньою за рік для більшості країн світу, в яких використання теплових насосів є релевантним. Окрім теплової енергії також вказуються приблизні значення споживаної електричної енергії, коефіцієнти продуктивності та коефіцієнти завантаженості теплового насосу.

Існують методи для розрахунків усіх вище названих характеристик теплонасосних установок для конкретних кліматичних умов. У розробленому програмному агенті користувачу надається можливість обрати період часу для моніторингу за роботою теплового насоса. Керівником дипломної роботи була надана статистична інформація температурного клімату на території України у вигляді Microsoft Excel файлу. За допомогою цієї інформації та періодом часу моделювання,

що задає користувач, агент може визначити температурні умови для моніторингу за роботаю теплового насоса в указаний проміжок часу.

Для розрахунку актуальної величини сгенерованої теплової енергії теплонасосною установкою використовуються нормалізуючі коефіцієнти, які в свою чергу мають функціональну залежність від температури навколишнього середовища, цю функціональну залежність зображено на рисунку 4.6:

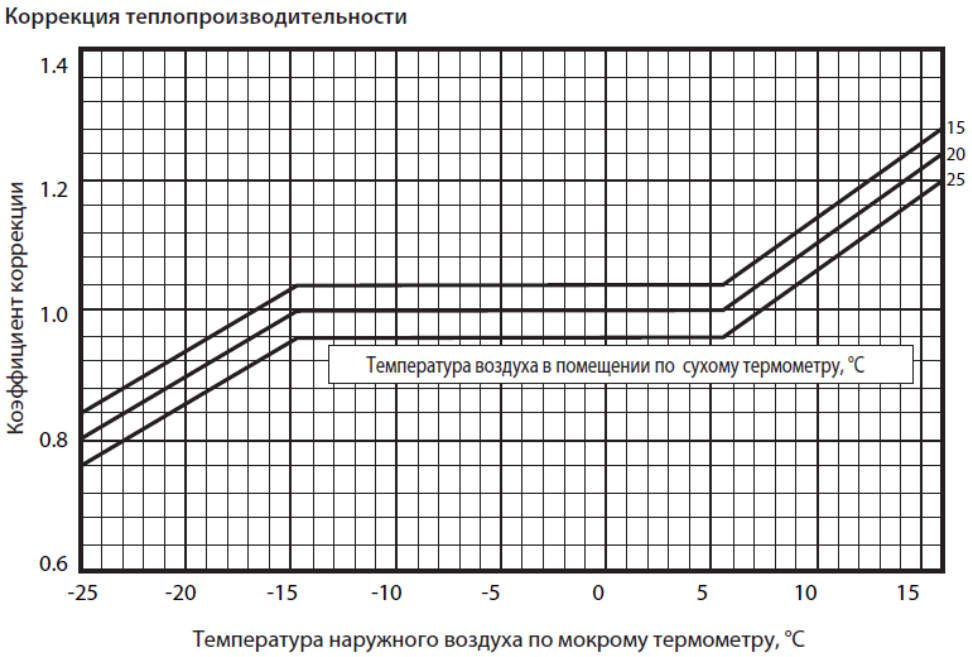

Рисунок 4.6 – Графік залежності коефіцієнту корекції від температури

Як видно на рисунку 4.6, графік залежності представляє собою комбінацію лінійних залежностей коефіцієнту корекції від температури. Числа 15, 20 та 25 у свою чергу визначають величини температурного режиму теплового насоса і позначають необхідні температури повітря у досліджуваному приміщені. Таким чином, маючи значення корекційного коефіцієнту та паспортне значення теплової продуктивності теплового насосу за формулою 4.1 визначається сгенероване значення теплової енергії за конкретної температури:

$$Q^{i}_{\text{роб.ТН}} = Q_{\text{ном.ТН}} \cdot K^{i}_{\text{кор.}Q_{\text{ТН}}}, \tag{4.1}$$

Де $Q^{i}_{\text{роб.ТН}}$ – значення сгенерованої теплової енергії за температури i, $Q_{\text{ном.ТН}}$ – номінальне значення продуктивності теплонасосної установки, $K^{i}_{\text{кор.}Q_{\text{ТН}}}$ – коефіцієнт

корекції теплового насосу за температури i.

4.4. Опис алгоритму розрахунку споживаної енергії теплонасосною установкою

Так само як і потенціал сгенерованої енергії кількість споживаної енергії функціонально залежить від температурного режиму навколишнього середовища. Окрім цього, слід зауважити, що у випадку генерації продуцентом енергії є тепловий насос, він є основним та єдиним джерелом створення теплової енергії, що не можна сказати про її споживання. Так наприклад у системі теплонасосної установки можуть існувати додаткові догрівачі, фанкойли та інші консументи електричної енергії.

Додаткові догрівачі використовуються у системі теплонасосною установки у тому випадку, коли кількості сгенерованої теплової енергії недостатньо для обігріву заданої площі об’єкту, при цьому важливими є два фактори:

— коефіцієнт завантаженості системи дорівнює 1, тобто система завантажена на 100 відсотків;

— використання додаткового теплового насосу не навантажує систему хоча б на 75 відсотків.

Якщо обидві умови було дотримано, з’являється необхідність включити до системи теплонасосної установки додаткових споживачів електричної енергії.

Як було сказано раніше, кількість спожитої енергії тепловим насосом має функціональну залежність від температури навколишнього середовища, ця залежність зображена на рисунку 4.7:

Відповідно до рисунку, так само як і у випадку з сгенерованою енергією, основним чинником змін у кількості спожитої електроенергії є коефіцієнт коррекції, що у свою чергу залежить від температури.

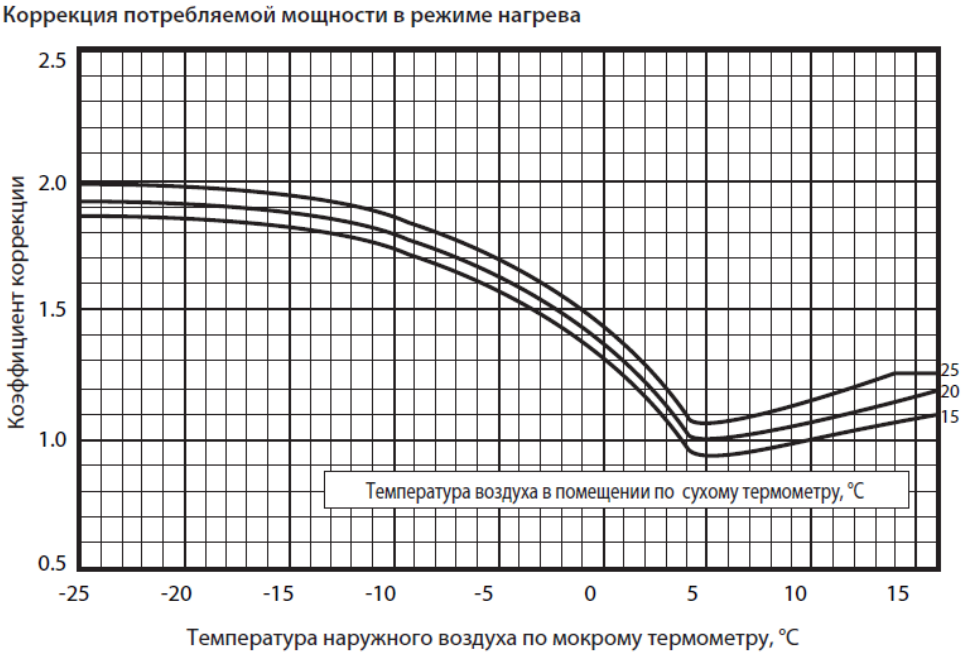

Рисунок 4.7 – Залежність корекційного коефіцієнту спожитої енергії від температури

Для розрахунку спожитої електроенергії тепловим насосом за певної температури слід скористатися формулою 4.2:

$$P^{i}_{\text{спож.ТН}} = P_{\text{ном.ТН}} \cdot K^{i}_{\text{кор.}P_{\text{ТН}}}, \tag{4.1}$$

Де $P^{i}_{\text{спож.ТН}}$ – значення спожитої електроенергії за температури i, $P_{\text{ном.ТН}}$ – номінальне значення споживання електроенергії тепловою установкою, $K^{i}_{\text{кор.}Q_{\text{ТН}}}$ – коефіцієнт корекції спожитої енергії тепловим насосом за температури i.

4.5. Опис алгоритму розрахунку споживання теплової енергії досліджуваною будівлею

Існує досить багато алгоритмів розрахунку та аналізу втрат тепла жилими та не жилими приміщенями. У ході проведених досліджень предметної області, було прийнято рішення скористатися методом тепловтрат за значеннями питомої тепловтрати будівлі та площі досліджуваного об’єкта. Як правило значення тепловтрат будівлі значиться у паспорті приміщення, те саме можна сказати і про

площу досліджуваної будівлі.

Як і попередньо розраховані величини прийнято вважати, що тепловтрати будівлі напряму залежать від температури середовища, точніше кажучи залежать лінійно, цю залежність зображено на рисунку 4.8:

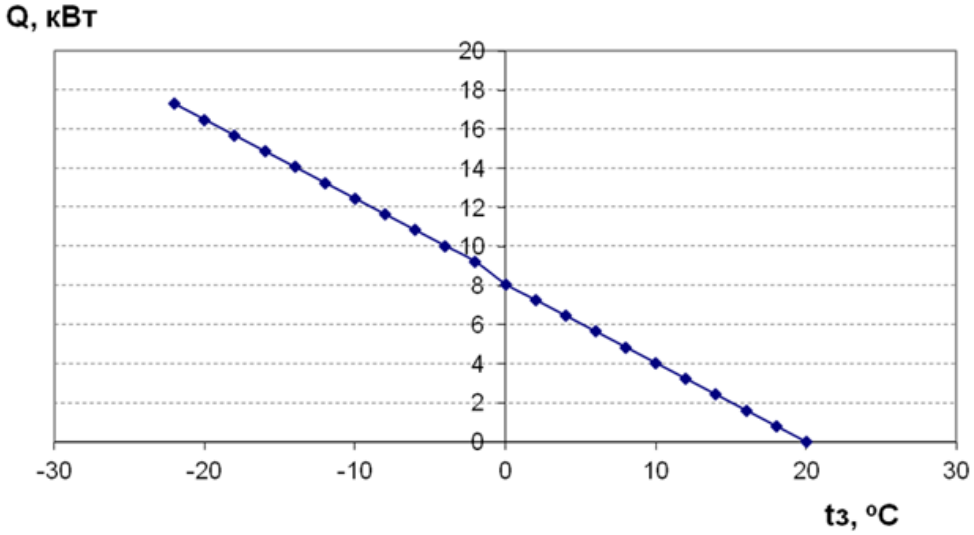

Рисунок 4.8 – Графік залежності тепловтрат будівлі від температури

Прийнято вважати, що за температури 20 градусів за Цельсієм будь яка будівля припиняє обмінюватись теплом з навколишнім середовищем. Знаючи мінімальну температуру за період, що ввів користувач для процесу моніторингу, можна скласти рівняння прямої, що проходить через 2 точки формула 4.3:

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1}, \tag{4.3}$$

Таким чином, маючи графік прямої, можна знайти питому тепловтрату будівлі для кожної з температур у заданому періоді дослідження. Оскільки площа будівлі є одним з обов’язкових параметрів, що вводить користувач, залишається скористуватися формулою 4.4, що зображена нижче:

$$Q^i_{\text{втрат}} = Q^i_{\text{пит.втрат}} \cdot S_{\text{буд.}}, \tag{4.4}$$

Де $Q^i_{\text{втрат}}$ – втрати теплової енергії будівлі за температури i, $Q^i_{\text{пит.втрат}}$ – питома тепловтрата енергії будівлі за за температури i, $S_{\text{буд.}}$ – площа досліджуваної будівлі.

Отже, втрати теплової енергії будівлі лінійно залежать від температурних умов навколишнього середовища та площі досліджуваного об’єкта опалення.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Виконаний програмний агент належить до класу клієнт-серверних застосунків. Через це системні вимоги до системи поділені на дві складових: вимоги до користувача та вимоги до сервера, на якому буде розгорнутий сервер агента. Слід зауважити, що вимоги до сервера включають у себе не тільки апаратну частину, але й програмну. Оскільки усі математичні розрахунки відбуваються на сервері, користувачеві не слід піклуватися про встановлення додаткового програмного забезпечення.

Великою перевагою розробленого програмного продукту є використання Apache Tomcat, який дозволяє розгортати програмний агент у локальній мережі, або на хостинг сервері. Якщо систему було розгорноту у локальній мережі, користувачу достатньо перейти за адресою http://localhost:8080 і він потрапить на головну сторінку агенту. У випадку використання хостингу все залежить від його конфігурацій.

5.1. Системні вимоги

Для роботи користувача з розробленою програмною системою користувачу потрібні мінімальні потужності апаратного забезпечення (вимоги наведені в таблиці 5.1).

Таблиця 5.1. Вимоги до апаратного забезпечення клієнта

Пристрій	Характеристика
Процесор	Intel ® Core ™ 2 / 2 Duo / Pentium ® / Celeron ® / Xeon™ / i3 / i5 / i7 чи AMD 6 / Turion ™ / Athlon ™ / Duron ™ / Sempron ™ з тактовою частотою не нижче 1.5 GHz.

Оперативна пам’ять (RAM − Random Access Memory)	Рекомендовано не менше 1 RAM
Швидкість з’єднання з інтернет	Рекомендовано не менше 128 кб/сек

Підтримувані апаратні архітектури:

- 32-розрядна (x86);
- 64-розрядна (x64)

Нижче розглянемо вимоги до апаратного та програмного забезпечення комп’ютера, який буде виконувати функції сервера, на якому буде розгорнуто програмний застосунок (вимоги наведено в таблиці 5.2 та в таблиці 5.3 відповідно).

Таблиця 5.2. Вимоги до апаратного забезпечення сервера

Пристрій	Характеристика
Процесор	Intel ® Core ™ / i3 / i5 / i7 чи AMD K5 ™ / K6 ™/ K7 ™ з тактовою частотою не нижче 3.0 GHz.
Оперативна пам’ять (RAM − Random Access Memory)	Рекомендовано не менше 8 RAM
Швидкість з’єднання з інтернет	Рекомендовано не менше 1024 кб/сек
Вільне місце на жорсткому диску	Рекомендовано 200 гб

Таблиця 5.3. Вимоги до програмного забезпечення сервера

Розділ	Назва
Операційна система	Windows Vista, 7, 8, 8.1, 10, Mac OS 10.10, 10.11, 10.12, Linux
Передустановлені програми та компоненти	Maven
	Java JDK 8
	PostgreSQL Server

5.2. Сценарії роботи користувача в розробленій системі

Головна сторінка предсталяє собою набір карток с короткою інформацією про кожен з теплових насосів. Для пошуку збережених звітів необхіжно звернутись до пошукового поля та ввести електронну адресу. Для переходу у меню конфігурації процесу монітиронгу необхідно обрати теплову установку та натиснути на кнопку Proceed info рисунок 5.1:

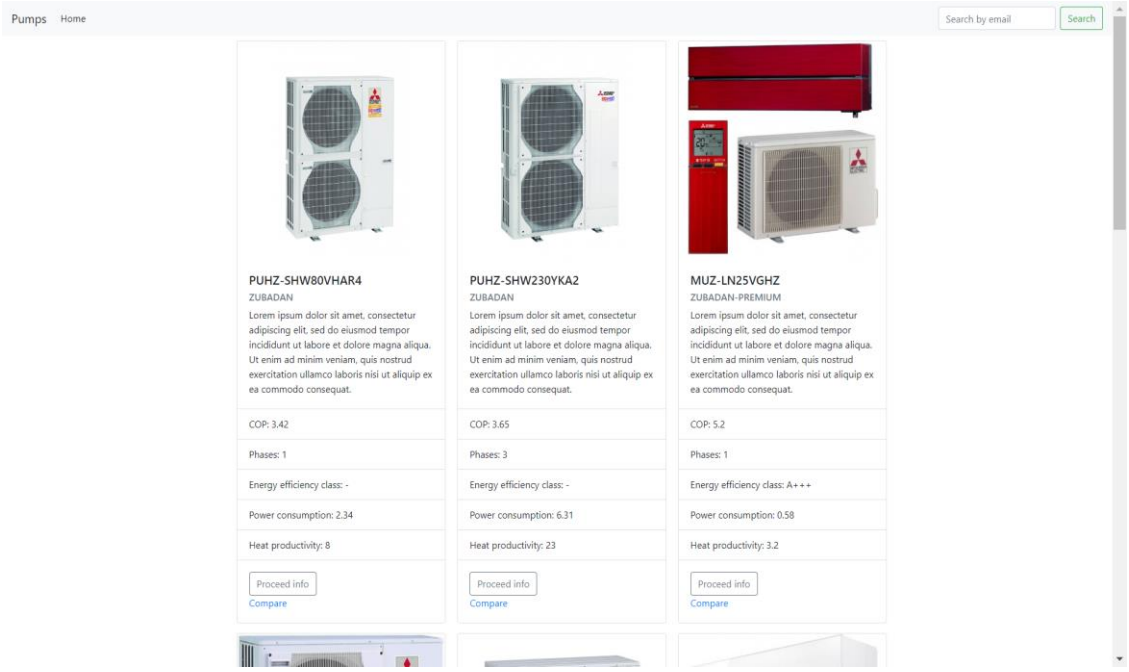

Рисунок 5.1 — Головна сторінка агенту

Після натиску на кнопку Proceed info, користувач потрапляє на сторінку

детальної інформації про тепловий насос та водночас форми конфігураціх параметрів моделювання процесу моніторингу за роботою теплового насоса рисунок 5.2:

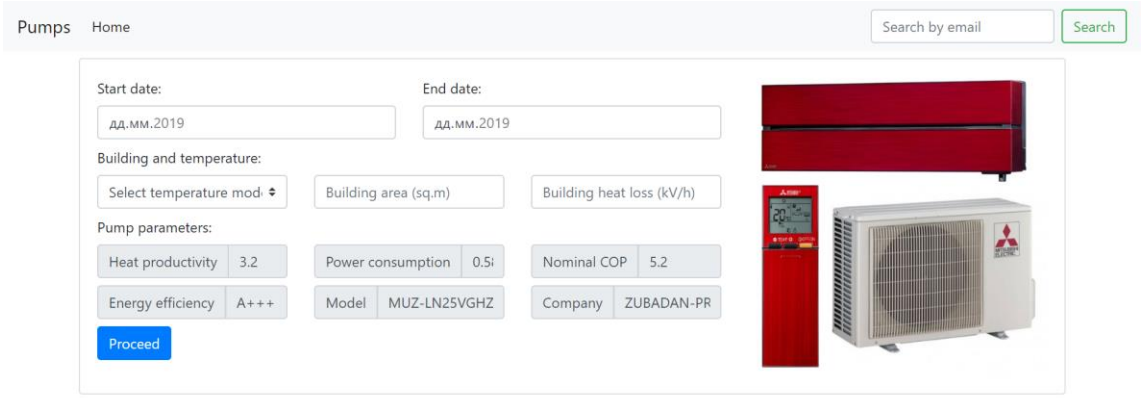

Рисунок 5.2 – Сторінка інформації та конфігурації процесу моделювання

На цій сторінці користувачу пропонується ввести необхідні параметри для запуску процесу моніторингу за роботою теплового насоса. Як було сказно раніше програмний агент обладнений двух-факторную верифікацією даних, тобто спочатку введені користувачем дані перевіряються на стороні клієнта, потім на сервері. У разі введених некоректних даних система відповість помилкою під відповідними полями. Механізм валідування зображено на рисунку 5.3:

Оскільки всі параматри, що вводяться користувачем, є конче для подальших розрахунків, окрім стандартної валідації на стороні клієнта, вони також проходять повторну валідацію на стороні серверу, так наприклад, якщо у процесі аналізу вхідних даних сервер не зможе створити об’єкти на основі переданих даних старт процесу моделювання буде відкладено.

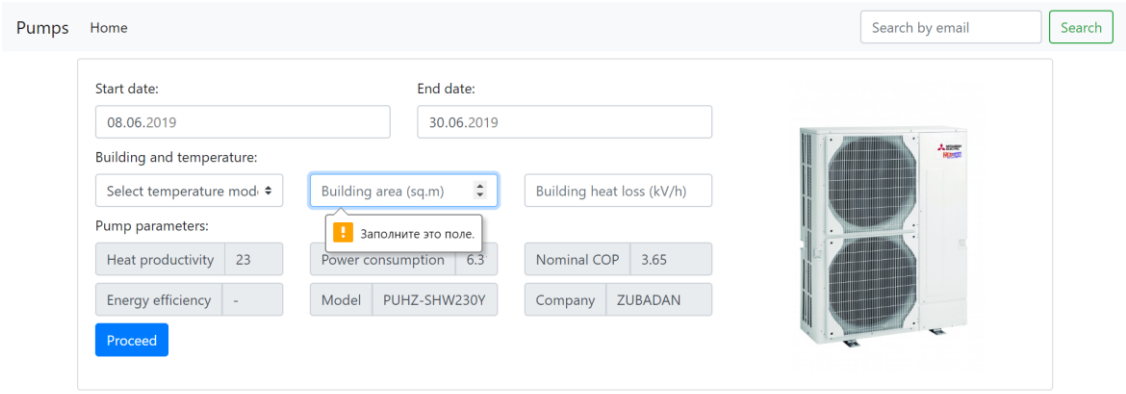

Рисунок 5.3 – Валідація даних, що вводяться користувачем

Якщо валідація даних пройшла успішно два кроки. Користувач потрапляє на сторінку згенерованого звіту з моніторингу роботи теплового насосу у заданий період часу рисунок 5.4:

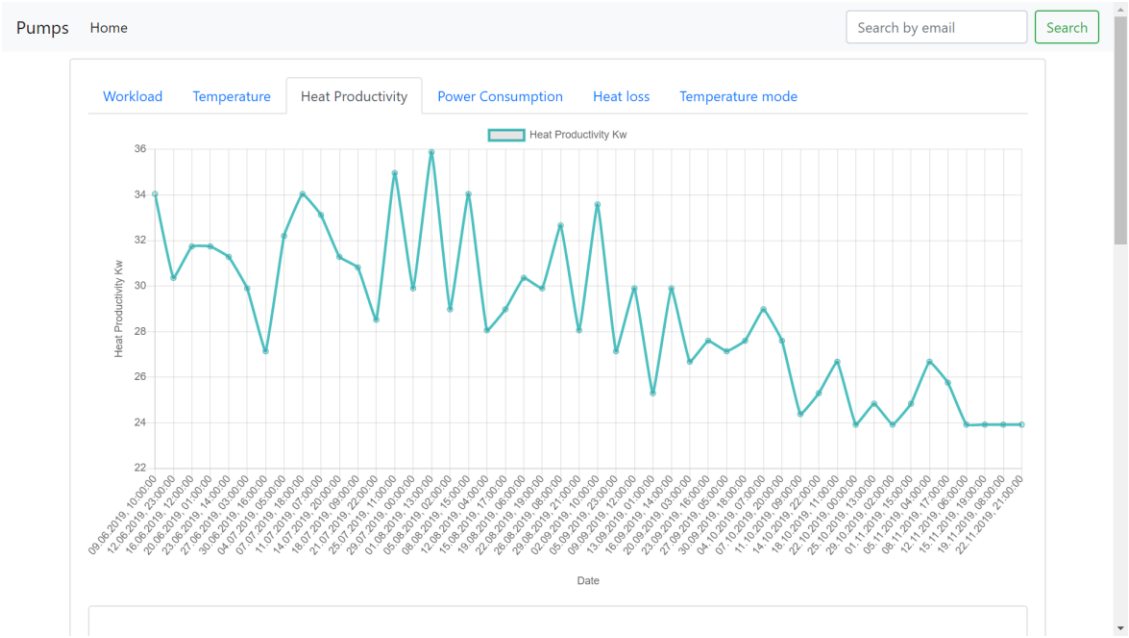

Рисунок 5.4 – Початок сторінки звіту з моделювання процесу роботи теплового насосу

Сторінка звіту з роботи починається з панелі інтерактивних графіків, що репрезентують характеристику роботу теплонасосної установки впродовж заданого періоду часу та з заданими характеристиками будівлі. Користувач може

переключатись між необхідними графіками для детального аналізу роботи теплонасосною установки.

Загалом інтерактивна панель графіків скадається з 6 графіків, 2 з них – стовпчикові діаграми, інші 4 – лінійні графіки залежностей. Механізм переходу зображено на рисунку 5.5:

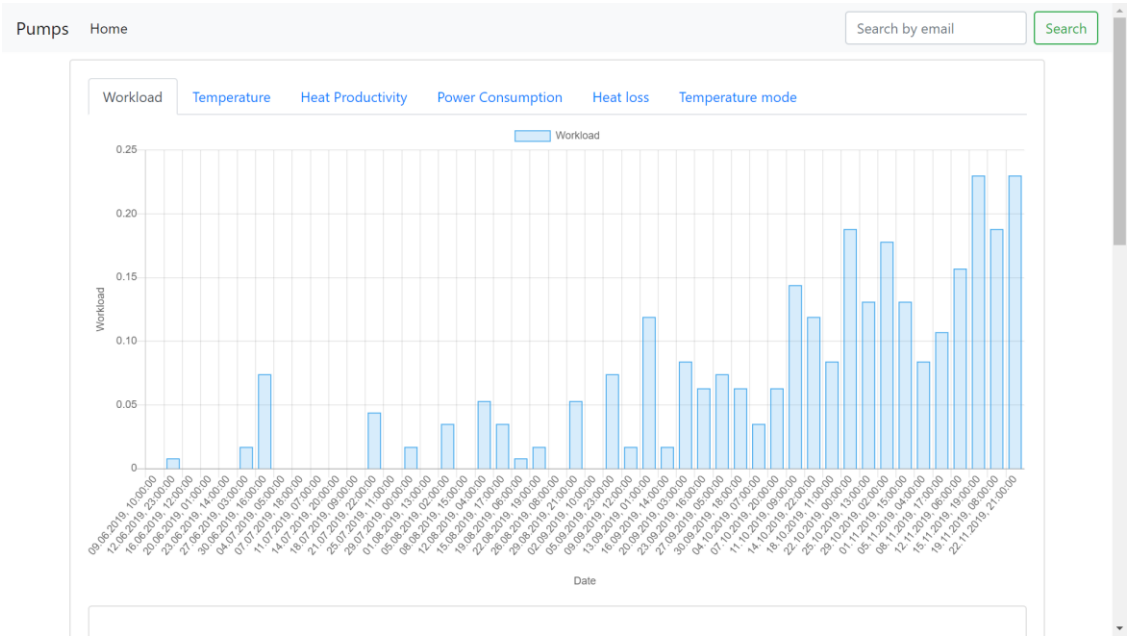

Рисунок 5.5 – Перехід до стовбачстої діаграми завантаженості теплового насоса

Нижче знаходиться дублюча інформація про досліджуваний тепловий насос, рисунок 5.6.

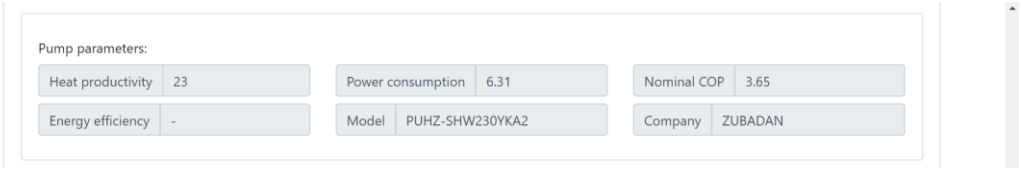

Рисунок 5.6 – Інформація про тепловий насос

Найважливіша частина звіту знаходиться одразу після інформації про тепловий насос – таблиця розрахованих значень роботи теплонасосної установки – рисунок 5.7:

Show: 10 Search:

Temperature	Heat loss	Heat productivity normalization coefficient	Normalized heat Productivity	Power consumption normalization coefficient	Normalized power consumption	Heat productivity of additional heater	Workload coefficient
27	0	1.48	34.04	1.15	7.256	0	0
19	0.25	1.32	30.36	1.15	7.256	0	0.008
22	0	1.38	31.74	1.15	7.256	0	0
22	0	1.38	31.74	1.15	7.256	0	0
21	0	1.36	31.28	1.15	7.256	0	0
18	0.5	1.3	29.9	1.15	7.256	0	0.017
12	2	1.18	27.14	1.096	6.916	0	0.074
23	0	1.4	32.2	1.15	7.256	0	0
27	0	1.48	34.04	1.15	7.256	0	0
25	0	1.44	33.12	1.15	7.256	0	0

Showing 1 to 10 of 48 records Pages: Previous 1 2 3 ... 5 Next

Рисунок 5.7 – Таблиця розрахованих значень

Слід зауважити, що таблиця володіє широким спектром інтерактивних функцій. Так наприклад користувач має можливість сортувати записи таблиці відносно вибраної колонки. Сортування підтримується обох типів – від більшого до меншого та навпаки. Приклад сортування зображенно на рисунку 5.8.

Show: 10 Search:

Temperature ▲	Heat loss	Heat productivity normalization coefficient	Normalized heat Productivity	Power consumption normalization coefficient	Normalized power consumption	Heat productivity of additional heater	Workload coefficient
-2	5.5	1.04	23.92	1.412	8.913	0	0.23
-2	5.5	1.04	23.92	1.412	8.913	0	0.23
2	4.5	1.04	23.92	1.2	7.572	0	0.188
2	4.5	1.04	23.92	1.2	7.572	0	0.188
3	4.25	1.04	23.92	1.13	7.13	0	0.178
5	3.75	1.04	23.92	0.99	6.247	0	0.157
6	3.5	1.06	24.38	1.004	6.335	0	0.144
7	3.25	1.08	24.84	1.018	6.424	0	0.131
7	3.25	1.08	24.84	1.018	6.424	0	0.131
8	3	1.1	25.3	1.032	6.512	0	0.119

Showing 1 to 10 of 48 records Pages: Previous 1 2 3 ... 5 Next

Рисунок 5.8 – Приклад сортування за температурою

Окрім сортування користувач може шукати у таблиці конкретні записи за допомогою пошукового поля, оперувати пагінацією та кількістю одночасно відображених записів у таблиці, приклади роботи зображені на рисунках 5.9 та 5.10:

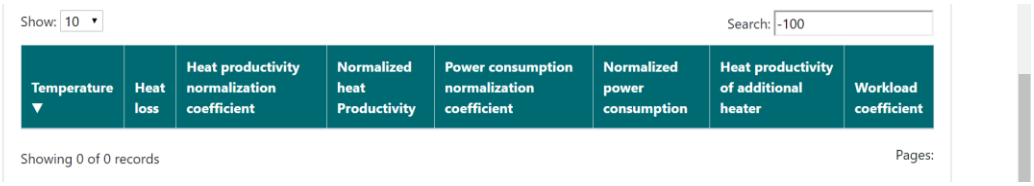

Рисунок 5.9 – Приклад пошуку запису з температурою -100 градусів

Рисунок 5.10 – Кількість одночасно відображених записів була збільшена до 20

У останній частині звіту користувачеві пропнується завантажети звіт на його комп’ютер, або зберегти звіт до бази даних агента з прив’язкою до електроної адреси користувача, даний процес зображено на рисунку 5.11:

У разі вибору збереження файлу локально на комп’ютер система підготує усі складені графіки та діаграми та представить їх у вигляді структурованого файлу з розширенням .pdf, що є дуже зручним та практичним з точки зору друкування.

Слід зауважити, що звіт, який не був збережений до бази даних з електроною адресою користувача ніколи не зможе бути відновленим.

Як було сказано раніше до списку збережених звітів користувач може доступитися за допомогою пошукого поля. Достатньо ввести адресу електроної пошти, що пов’язана зі звітом, якщо насправді такий звіт існував у системі –

користувач буде переведений до сторінки з збереженими звітами. Ця сторінка зображена на рисунку 5.12:

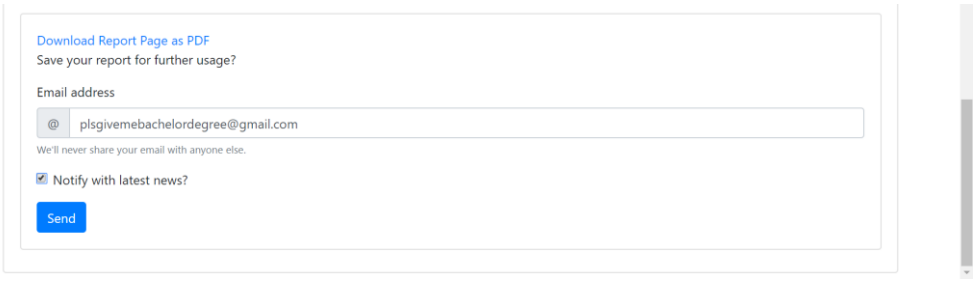

Рисунок 5.11 – Форма збереження звіту

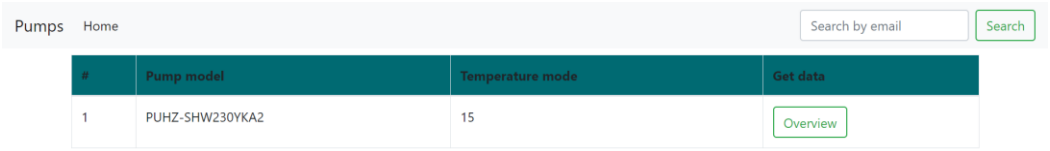

Рисунок 5.12 – Сторінка збережених звітів

Для перегляду звіту, користувачеві необхідно натиснути на кнопку Overview.

ВИСНОВКИ

В ході виконання даної роботи було розроблено програмний агент мультиагентної системи управління та моніторингу теплонасосною установкою.

Агент було написано на мові програмування Java з використанням графічного веб-інтерфейсу.

Програму можна використовувати для моделювання процессу використання теплового насоса у певний проміжок часу за певних температур та параметрів будівлі.

В ході роботи було проведено огляд та зроблено аналіз засобів, що були використані для створення даного програмного забезпечення (середовища розробки Intellij IDEA, Java Spring Framework та засобів створення веб-інтерфейсів: HTML5, CSS, JavaScript та Freemarker).

Сервер програмного агенту було розміщено на безкоштовному хостинг-сервісі Heroku.

Для роботи з даним програмним забезпеченням необхідний лише комп’ютер середньої потужності.

Користувач має змогу власноруч задавати параметри системи для моделювання процессу використання вибраного теплового насоса. Для нього виводяться графіки тепловтрат будівлі, споживання електроенергії, генерування теплової енергії, завантаженості теплонасосної установки та графік температурних умов впродовж заданого періоду.

Користувач може зберігати звіти на особистий комп’ютер, або зберігати звіт у базу данних та переглядати його у будь-який час.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Документація PostgreSQL [Електронний ресурс] — https://www.postgresql.org/ Дата доступу: 04.06.2017
2. Фаулер М. Рефакторинг. Улучшение существующего кода [Електронний ресурс] / М. Фаулер. — Пер. с англ. — СПб: Символ-Плюс, 2003. — 432 с. — Режим доступу: dolina-sun.ru/filemanager/download/77
3. И.Н. Блинов, В.С. Романчик Минск: издательство «Четыре четверти», 2013. — 896 с.
4. Офіційний сайт Spring Framework [Електронний ресурс] — Режим доступу: https://spring.io/ - Дата доступу: 01.06.2017
5. Собеседование по Java EE — Spring Framework [Електронний ресурс] — Режим доступу: http://javastudy.ru/interview/jee-spring-questions-answers/ - Дата доступу: 01.06.2017
6. Документація Spring Framework. 1. Introduction to Spring Framework [Електронний ресурс] — Режим доступу: https://docs.spring.io/spring/docs/3.0.0.M4/reference/html/ch01s02.html Дата доступу: 01.06.2017
7. Документація Spring Framework. 6. The Web [Електронний ресурс] — Режим доступу: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html Дата доступу: 01.06.2017
8. Документація Spring Data Framework [Електронний ресурс] — Режим доступу:https://docs.spring.io/spring-data/jpa/docs/current/reference/html/ Дата доступу: 01.06.2017
9. Офіційний сайт Spring Data Framework [Електронний ресурс] — Режим доступу: http://spring-projects.ru/projects/spring-data/jpa/ Дата доступу: 01.06.2017
10. Офіційний сайт Hibernate Framework [Електронний ресурс] — Режим доступу: http://hibernate.org/orm/ Дата доступу: 01.06.2017

11. Документація Maven Framework [Електронний ресурс] — Режим доступу: http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference Дата доступу: 01.06.2017
12. Пауэлл Т.А. Полное руководство по HTML / Т.А. Пауэлл. — М.: Мир, 2001. — 912 с.
13. Кисленко Н.П. HTML. Самое необходимое. / Н.П. Кисленко — СПб.: БХВ-Петербург, 2008. — 352 с.
14. Справочник CSS [Електронний ресурс] — Режим доступу: http://htmlbook.ru/css Дата доступу: 01.06.2017
15. Офіційний сайт Bootstrap 3 [Електронний ресурс] — Режим доступу: http://bootstrap-3.ru/index.php Дата доступу: 01.06.2017
16. К. Дж. Дейт. Введение в системы баз данных / An Introduction to Database Systems. — 7-е изд. — «Вильямс», 2001. — 1092 с.
17. Офіційний сайт Intelij Idea [Електронний ресурс] — Режим доступу: https://www.jetbrains.com/idea/Дата доступу: 01.06.2017
18. Брюс У. Перри. Java сервлеты и JSP. Сборник рецептов [Електронний ресурс]/ Брюс У. Перри — Пер. С англ. — КУДИЦ-Пресс, 2009 — 768 с. — Режим доступу: https://www.ozon.ru/context/detail/id/4434287
19. Multi-Agent Systems for Power Engineering Applications—Part 1: Concepts, Approaches, and Technical Challenges [Електронний ресурс] — IEEE TRANSACTIONS ON POWER SYSTEMS, VOL. 22, NO. 4, NOVEMBER 2007. — 11с. — Режим доступу: http://strathprints.strath.ac.uk/26472/ .
20. Modelling and Simulation of Electrical Energy Systems through a Complex Systems Approach using Agent-Based Models [Електронний ресурс] – 202с.- https://www.researchgate.net/publication/259739043_Modelling_and_Simulation_of_Electrical_Energy_Systems_through_a_Complex_Systems_Approach_using_Agent-Based_Models.
21. Energy Management of Distributed Resources in Power Systems Operations - Models [Електронний ресурс] – 193с.

22. Сайт GE Renewable Energy // Heat Pump Managament [Електронний ресурс] — Режим доступу: https://www.ge.com/renewableenergy
23. Сайт Power Electronics // Power Managament Chapter 16: Heat Power. [Електронний ресурс] — Режим доступу: https://www.powerelectronics.com/power-management

Додаток 1

Програмний агент управління та моніторингу теплонасосної установки

Специфікація

УКР.НТУУ“КПІ”.ТМ5199_19Б

Аркушів 2

2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ«КПІ ім. Ігоря Сікорського».ТМ5199_19Б 81-1	Записка	Пояснювальна записка
Компоненти		
УКР.НТУУ«КПІ». ТМ5199_19Б 12-1	Текст програмного модулю	
УКР.НТУУ«КПІ». ТМ5199_19Б 13-1	Опис програми	

Додаток 2

Програмний агент управління та моніторингу теплонасосної установки

Текст програмного модулю

УКР.НТУУ“КПІ”.ТМ5199_19Б 12-1

Аркушів 7

2019

```
package diploma.controller;

import diploma.domain.entities.Customer;
import diploma.domain.entities.Pump;
import diploma.domain.entities.Report;
import diploma.repos.CustomerRepo;
import diploma.repos.PumpRepo;
import diploma.repos.ReportRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

@Controller
public class MainController {

    @Autowired
    private PumpRepo pumpRepo;

    @Autowired
    private ReportRepo reportRepo;

    @Autowired
    private CustomerRepo customerRepo;

    @GetMapping("/greeting")
    public String greeting(@RequestParam(name = "name", required = false,
defaultValue = "World") String name, Map<String, Object> model) {
        model.put("name", name);
        return "proceed";
    }

    @GetMapping("/")
    public String main(Model model) {
        Iterable<Pump> allPumps = pumpRepo.findAll();
        model.addAttribute("pumps", allPumps);
        return "main";
    }

    @GetMapping("/filter")
    public String filter(Model model, @RequestParam String filter) {
        Customer customer = customerRepo.findByEmail(filter);
        if (customer != null) {
            List<Report> reports = reportRepo.findByOwner(customer);
            reports = reports == null ? new ArrayList<>() : reports;
            model.addAttribute("reports", reports);
            return "filter";
        } else {
            Iterable<Pump> allPumps = pumpRepo.findAll();
            model.addAttribute("pumps", allPumps);
            return "main";
        }
    }
}


package diploma.controller;
```

```
import diploma.domain.entities.FormattedDate;
import diploma.domain.entities.Pump;
import diploma.domain.entities.Report;
import diploma.domain.util.TemperatureData;
import diploma.repos.PumpRepo;
import diploma.repos.ReportRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.Map;
import java.util.Optional;

@Controller
public class ReportController {
    @Autowired
    private PumpRepo pumpRepo;

    @Autowired
    private ReportRepo reportRepo;

    @GetMapping("/proceed")
    public String proceedPump(Model model,
                              @RequestParam String pumpId) {
        Optional<Pump> pumpOption = pumpRepo.findById(Long.parseLong(pumpId));
        pumpOption.ifPresent(pump -> model.addAttribute("pump", pump));
        return "proceed";
    }

    @PostMapping("/data")
    public String getData(Model model,
                          @RequestParam String area,
                          @RequestParam String loss,
                          @RequestParam String tempMode,
                          @RequestParam String pumpId,
                          @RequestParam String dateFrom,
                          @RequestParam String dateTo) {
        Optional<Pump> pumpOption = pumpRepo.findById(Long.parseLong(pumpId));
        dateFrom = dateFrom + "T00:00";
        dateTo = dateTo + "T00:00";
        Map<FormattedDate, Integer> temperatureForPeriod =
TemperatureData.getTemperatureForPeriod(dateFrom, dateTo);
        Map<Integer, Integer> temperatureFrequency =
TemperatureData.getTemperatureFrequency(dateFrom, dateTo);
        if (pumpOption.isPresent()) {
            Pump pump = pumpOption.get();
            Double heatLoss = Double.parseDouble(area) * Double.parseDouble(loss);
            Report report = new Report(temperatureForPeriod, temperatureFrequency,
Integer.parseInt(tempMode), pump, heatLoss);
            reportRepo.save(report);
            model.addAttribute("keys", temperatureFrequency.keySet());
            model.addAttribute("values", temperatureFrequency.values());
            model.addAttribute("report", report);
            model.addAttribute("pump", pump);
        }
        return "report";
    }

    @GetMapping("/loadData")
    public String loadData(Model model,
```

```
                            @RequestParam String reportId,
                            @RequestParam String pumpId) {
        Optional<Pump> pumpOption = pumpRepo.findById(Long.parseLong(pumpId));
        Optional<Report> reportOption =
reportRepo.findById(Long.parseLong(reportId));
        if (pumpOption.isPresent() && reportOption.isPresent()) {
            Pump pump = pumpOption.get();
            Report report = reportOption.get();
            model.addAttribute("keys", report.getTempFreq().keySet());
            model.addAttribute("values", report.getTempFreq().values());
            model.addAttribute("report", report);
            model.addAttribute("pump", pump);
        }
        return "report";
    }
}
package diploma.domain.entities;

import javax.persistence.*;

@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column
    private String email;

    @Column
    private Boolean shareNews;

    public Customer() {
    }

    public Customer(String email) {
        this.email = email;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Boolean getShareNews() {
        return shareNews;
    }

    public void setShareNews(Boolean shareNews) {
        this.shareNews = shareNews;
    }
}
```

```
package diploma.domain.util;

import diploma.domain.entities.FormattedDate;
import org.apache.poi.openxml4j.exceptions.InvalidFormatException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.util.*;
import java.util.stream.Collectors;

public final class TemperatureData {
    private static final int MAX_CAPACITY = 48;

    private static final Logger logger =
LoggerFactory.getLogger(TemperatureData.class);
    private static final Map<FormattedDate, Integer> hourTemperature = new
TreeMap<>();

    static {
        try {
            XLSXUtil.readXLSXTo(hourTemperature,
"src/main/resources/data/temperature.xlsx");
        } catch (IOException | InvalidFormatException e) {
            logger.error(e.getMessage());
        }
    }

    private TemperatureData() {
    }

    public static Map<FormattedDate, Integer> getTemperatureForPeriod(final String
dateFrom, final String dateTo) {
        Map<FormattedDate, Integer> collect = getFormattedDateInPeriod(dateFrom,
dateTo);
        List<FormattedDate> keys = new ArrayList<>(hourTemperature.keySet());
        int keysIncrement = collect.size() / MAX_CAPACITY;
        return collect.entrySet().stream()
                .filter((Map.Entry<FormattedDate, Integer> x) -> {
                    FormattedDate key = x.getKey();
                    int index = keys.indexOf(key);
                    return index % keysIncrement == 0;
                })
                .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
(oldValue, newValue) -> oldValue, LinkedHashMap::new));
    }

    public static Map<Integer, Integer> getTemperatureFrequency(final String
dateFrom, final String dateTo) {
        Map<Integer, Integer> temperatureFrequency = new TreeMap<>();
        Map<FormattedDate, Integer> formattedDateInPeriod =
getFormattedDateInPeriod(dateFrom, dateTo);
        for (Integer temperature : formattedDateInPeriod.values()) {
            Integer count = temperatureFrequency.get(temperature);
            if (count == null) {
                temperatureFrequency.put(temperature, 1);
            } else {
                temperatureFrequency.put(temperature, ++count);
            }
        }
        return temperatureFrequency;
    }
```

```
    private static Map<FormattedDate, Integer> getFormattedDateInPeriod(String
dateFrom, String dateTo) {
        FormattedDate formDateTo = new FormattedDate(dateTo);
        FormattedDate formDateFrom = new FormattedDate(dateFrom);
        return new TreeMap<>(hourTemperature).entrySet().stream()
                .filter((Map.Entry<FormattedDate, Integer> x) -> {
                    FormattedDate formattedDate = x.getKey();
                    return formDateTo.compareTo(formattedDate) > 0 &&
                            formDateFrom.compareTo(formattedDate) <= 0;
                })
                .collect(Collectors.toMap(Map.Entry::getKey, Map.Entry::getValue,
(oldValue, newValue) -> oldValue, LinkedHashMap::new));
    }
}
<#import "parts/common.ftl" as c>
<#import "parts/pump.ftl" as p>
<@c.page>
    <form action="/data" method="post" class="my-2 card card-body">
        <div class="row">
            <div class="col-8">
                <div class="row">
                    <div class="col">
                        <label for="start">Start date:</label>
                        <input type="date" id="start" name="dateFrom" required
class="form-control" min="2019-01-01"
                               max="2019-12-31">
                    </div>
                    <div class="col">
                        <label for="end">End date:</label>
                        <input type="date" id="end" name="dateTo" required
class="form-control" min="2019-01-01"
                               max="2019-12-31">
                    </div>
                </div>
                <p class="my-2">Building and temperature:</p>
                <div class="row my-2">
                    <div class="col">
                        <select name="tempMode" class="custom-select">
                            <option selected>Select temperature mode</option>
                            <option value="15">15</option>
                            <option value="20">20</option>
                            <option value="25">25</option>
                        </select>
                    </div>
                    <div class="col">
                        <input type="number" name="area" class="form-control"
required placeholder="Building area (sq.m)"
                               step="0.1"
                               min="0">
                    </div>
                    <div class="col">
                        <input type="number" name="loss" class="form-control"
required placeholder="Building heat loss (kV/h)"
                               step="0.1" min="0">
                    </div>
                </div>
                <@p.pumpParam></@p.pumpParam>
                <input type="hidden" name="pumpId" value="${pump.id}">
                <div class="form-group row">
                    <div class="col-sm-10">
                        <button type="submit" class="btn btn-
primary">Proceed</button>
                    </div>
```

```
                </div>
            </div>
            <div class="col-4 text-center">
                <img src="img/${pump.id}.jpg" class="img-fluid">
            </div>
        </div>
    </form>
</@c.page>
$('#downloadPdf').click(function (event) {
    var reportPageHeight = 1800;
    var reportPageWidth = 2200;

    var pdfCanvas = $('<canvas />').attr({
        id: "canvaspdf",
        width: reportPageWidth,
        height: reportPageHeight
    });
    var pdfctx = $(pdfCanvas)[0].getContext('2d');
    var pdfctxX = 0;
    var pdfctxY = 0;
    var buffer = 50;
    pdfctx.beginPath();
    pdfctx.rect(pdfctxX, pdfctxY, reportPageWidth, reportPageHeight);
    pdfctx.fillStyle = "white";
    pdfctx.fill();
    $("canvas").each(function (index) {
        var canvas = $(this);
        var canvasHeight = canvas.innerHeight();
        var canvasWidth = canvas.innerWidth();
        pdfctx.drawImage(canvas[0], pdfctxX, pdfctxY, canvasWidth, canvasHeight);
        pdfctxX += canvasWidth + buffer;

        if (index % 2 === 1) {
            pdfctxX = 0;
            pdfctxY += canvasHeight + buffer;
        }
    });

    var pdf = new jsPDF('l', 'pt', [reportPageWidth, reportPageHeight]);
    pdf.addImage($(pdfCanvas)[0], 'PNG', 0, 0);
    pdf.save();
});
```

Додаток 3

Програмний агент управління та моніторингу теплонасосної установки

Опис програмного модулю

УКР.НТУУ“КПІ”.ТМ5199_19Б 13-1

Аркушів 6

2019

АНОТАЦІЯ

Розроблений програмний агент моніторингу та управління теплонасосної станції має функції моделювання процесу генерації теплової енергії теплонасосною установкою впродовж довільного проміжку часу, функції збереження звіту процесу моделювання, можливість переглядати збережений звіт, функції конфігурування об’єкту моніторингу.

Користувачами можуть бути люди, які мають девайс з налаштуваннями інтернету.

ЗМІСТ

1. ВІДОМОСТІ ПРО ПРОГРАМНИЙ МОДУЛЬ

При створенні програмного забезпечення були використані такі засоби реалізації:

— середовище розробки IntelliJ IDEA, адже має найбільше функціоналу серед конкурентів;

— мову програмування Java для написання серверної частини;

— фреймворк Spring, а саме його модулі — Core Container, Web та Data Access/Integration, для організації архітектури та взаємодії серверної частини з інтерфейсом та базою даних;

— мову розмітки HTML;

— генератор шаблонів Freemarker, що дозволяє створювати динамічні веб-сторінки без дублікації коду;

— мову розмітки CSS з використання бібліотеки Bootstrap для розробки графічного інтерфейсу користувача;

— технологію Maven для підключення всіх модулів та бібліотек проекту;

— Git для впровадження системи контролю версій та легкої інтеграції з хостинг сервісом Heroku через створений репозиторій проекту;

— Tomcat контейнер сервлетів, що дозволяє зменшити час на розгортання проекту на хостингу або у локальній мережі;

реляційну базу даних PostgreSQL, вона є безкоштовную; має легкий та зрозумілий веб-інтерфейс та дуже просто інтегрується з середовищем розробки IntelliJ IDEA

1.1. Опис логічної структури

Для реалізації задачі моніторингу та управління теплонасосною установкою у складі мультиагентної системи був розроблений клієнт-серверний агент, сервер якого може бути розгорнутий у будь-якому інформаційному середовищі, а у якості клієнтського інтерфейсу виступає браузер користувача.

Інтерфейс користувача — виконано з використанням таких технологій: HTML мова гіпертекстової розмітки, CSS – каскадних стилів цієї розмітки, JavaScript мови програмування, що допомагає оживити сайт з точки зору інтерфейсу користувача.

1.2. Вхідні та вихідні дані

Вхідними даними для системи є температурні дані України за період 2018-2019 років, дані користувача, що характеризують досліджувану будівлю, та вибраний тепловий насос.

Вихідними даними є звіт з моделювання роботи теплонасосної установки, що несу у собі інформацію про потенціал сгенерованої енергії, кількість спожитої енергії, коефіцієнти корекції споживаня та генерування енергії, коефіцієнт завантаженості системи, графік температурного режиму впродовж заданого періоду.

2. ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Програмний модуль було протестовано в браузері Google Chrome 74.0.3729.169 на персональному комп’ютері, який працює на базі процесору x64 Intel Core i7 vPro (8th Gen) та має 16 Гб оперативної пам’яті. Розроблене програмне забезпечення є кросбраузереним та кросплатформенним, що дозволяє запускати його на комп’ютерах будь-якої потужності та в будь-яких сучасних браузерах.