МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

Кафедра програмної інженерії

МАГІСТЕРСЬКИЙ ПРОЄКТ
на тему:
«Математичне та програмне забезпечення для автоматизованого проєктування конструкцій придверних решіток»

Керівник магістерського проєкту
к.ф.м.н. Бойко І. В.
_____
“___”_____2020 р.

Розробив студент гр. СПд -2
Кузишин А. І.

Тернопіль 2020

РЕЗЮМЕ

Магістерський проєкт на тему “Математичне та програмне забезпечення для автоматизованого проектування конструкцій придверних решіток” на здобуття освітньо ступеня “Магістр” 121 – Інженерія програмного забезпечення написана обсягом 100 сторінок і містить 7 ілюстрацій, 7 таблиць, 2 додатки та 41 джерело за переліком посилань.

Метою проєкту є розробка алгоритму автоматизованого проектування конструкцій придверних решіток.

Потреба застосування шаблонів проектування виникла, коли перед розробниками веб-додатків постало безліч проблем під час розробки складних систем. Програмісти, які займаються проектуванням комплексних графічних систем почали стикатися з проблемами поєднання дизайну та реалізації програмного коду в рамках ресурсу. Використовуючи стандартні підходи і методи реалізації бізнес-додатків, коли код програми розміщується безпосередньо в тілі програми, розробникам стало досить складно підтримувати внутрішній інтерфейс складних систем.

Результати проєкту можуть використані при автоматизованому проектуванні конструкцій придверних решіток та підтверджені довідкою про використання.

Можливими напрямками подальших досліджень є дослідження роботи автоматизованих систем проектування спрямованих на інші види виробництва та побудови їхніх трьох вимірних моделей.

КЛЮЧОВІ СЛОВА: ПРИДВЕРНІ РЕШІТКИ, АВТОМАТИЗОВАНІ СИСТЕМИ ПРОЕКТУВАННЯ, ВЕБ-ОРІЄНТОВАНІ СИСТЕМИ.

RESUME

Master's project on the topic "Mathematical and software for automated designing of lattice constructions" for obtaining an educational degree "Master" 121 - Software engineering is written in volume of 82 pages and contains 7 illustrations, 7 tables, 1 application and 30 sources under the list of references.

The aim of the project is to develop an algorithm for automated design of gateway gate structures.

The need to use design templates arose when web application developers faced many challenges during the development of complex systems. Programmers involved in the design of integrated graphics systems have begun to face the challenges of combining design and implementation of the code within the resource. Using standard approaches and methods of implementing business applications, when the program code is placed directly in the body of the program, it became difficult for developers to maintain the internal interface of complex systems.

The results of the project can be used in the automated designing of lattice constructions and confirmed by the use certificate.

Possible areas of further research are the study of the work of automated design systems aimed at other types of production and the construction of their three dimensional models.

KEYWORDS: PIPE CUTTINGS, AUTOMATED DESIGN SYSTEMS, WEB-ORIENTED SYSTEMS.KEYWORDS: PIPE CUTTINGS, AUTOMATED DESIGN SYSTEMS, WEB-ORIENTED SYSTEMS.

ЗМІСТ

ВСТУП

Web-ресурси представляються сукупністю Web-сторінок, що містять необхідну користувачеві інформацію та елементи управління, а процес роботи з ними - як перехід між сторінками, керований даними користувача, що задаються у відповідь на запит.

При збільшенні кількості сторінок у веб-додатку, що зберігаються на сервері, ускладнюється сам Web-ресурс. При цьому, він може утворити цілий програмний комплекс, який дозволить не лише переглядати інформацію, а й взаємодіяти з ним.

Актуальність роботи. Потреба застосування шаблонів проектування виникла, коли перед розробниками веб-додатків постало безліч проблем під час розробки складних систем. Програмісти, які займаються проектуванням комплексних графічних систем почали стикатися з проблемами поєднання дизайну та реалізації програмного коду в рамках ресурсу. Використовуючи стандартні підходи і методи реалізації бізнес-додатків, коли код програми розміщується безпосередньо в тілі програми, розробникам стало досить складно підтримувати внутрішній інтерфейс складних систем.

MVC (Model – View - Controller) – це архітектурний шаблон, який описує спосіб побудови структури веб-додатку, сфери відповідальності і взаємодії кожної з частин в даній структурі.

Основна ідея архітектурного шаблону MVC полягає в чіткому розподілі відповідальності за різні функціональні частини. Система ділиться на основні три компоненти, кожен з яких відповідає за певний тип поставлених задач. Їх буде досліджено та реалізовано у даному дипломному проекті.

Для кращого розуміння архітектури, що буде представлено в даній роботі, вона буде реалізована на прикладі використання веб-ресурсу для автоматизованого проектування конструкцій придверних решіток .

Зв'язок роботи з науковими програмами, планами та темами.

Напрямок виконаних досліджень безпосередньо пов'язаний з науково-дослідним напрямком кафедри «Комп’ютерної інженерії» Тернопільського національного технічного університету.

Мета і задачі дослідження. Метою проєкту є розробка алгоритму автоматизованого проектування конструкцій придверних решіток.

Основними **задачами** дослідження є:

- проведення аналізу існуючих рішень та архітектур веб-орієнтованих систем;
- розробка та дослідження алгоритму автоматизованого проектування конструкцій придверних решіток;
- проектування архітектури системи автоматизованого проектування конструкцій придверних решіток;
- розробка системи автоматизованого проектування конструкцій придверних решіток.

Об'єкт дослідження – системи автоматизованого проектування конструкцій та виробів.

Предмет дослідження – вирішення задачі побудови алгоритму автоматизованого проектування конструкцій придверних решіток.

Методи дослідження. Теоретичні дослідження по розробці моделі системи автоматизованого проектування ґрунтуються на застосуванні системного аналізу, методології функціонального моделювання, інженерії знань, семантичних мереж, теорії множин, теорії графів, теорії нечіткого виведення, теорії алгоритмів та об’єктно-орієнтованого проектування.

Наукова новизна одержаних результатів визначається наступним чином:

– розроблено алгоритми для автоматизованого проектування конструкцій придверних решіток, що дозволило розробити архітектуру системи;

– досліджено та проаналізовано існуючі можливості щодо веб орієнтованих систем автоматизованого проектування конструкцій, що обгрунтовує доцільність розробки;

– проаналізовано математичні особливості процесу розробки системи автоматизованого проектування конструкцій придверних решіток.

Практична цінність одержаних результатів полягає в тому, що:

– обґрунтовано підхід та розроблено алгоритм автоматизованого проектування конструкцій придверних решіток, що дозволило застосувати відповідне програмне та апаратне забезпечення для реалізації запропонованого рішення;

– реалізовано програмне забезпечення для автоматизованої побудови конструкцій придверних решіток засобами PHP, MySQL.

Особистий внесок магістранта. Всі результати отримані автором самостійно.

Публікації та апробація МП. За результатами наукових досліджень, проведених у магістерському проєкті, підготовлено тези доповіді «Автоматизація проектування конструкцій придверних решіток» обсягом 1 сторінка на Всеукраїнській школі-семінарі «Сучасні комп'ютерні інформаційні технології»

Для реалізації проекту використано веб-сервер Apache, який включає в себе основні функціональні можливості, такі як обробка конфігураційних файлів, протокол HTTP і система завантаження модулів. Також було задіяно інтерпретатор PHP (гіпертекстовий препроцесор) і систему управління базою даних MySQL з підтримкою транзакцій, що являє собою MySQL — компактний багатопотоковий сервер баз даних та характеризується високою швидкістю, стійкістю і простотою використання.

1 СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Аналіз існуючих архітектур проектування веб-ресурсів

Традиційний термін «сайт» відноситься до цієї користувальницької сторони Web-ресурсів. Однак сучасні Web-ресурси, як комп'ютерний продукт, характеризуються колосальною розбіжністю між тим, що пред'являється користувачеві, і тим, як це формується і пред'являється, і які засоби при цьому використовуються. При цьому ускладнення структури Web-ресурсів розвивається від сукупності HTML-сторінок, розміщених на сервері, які видаються користувачеві, до величезних програмних комплексів, коли переважна частина пропонованих сторінок формується програмним шляхом у процесі функціонування ресурсу, які диктуються насамперед величезним діапазоном завдань користувача.

Діапазон можливостей Web-ресурсів, з точки зору використання, визначає їх головну характеристику, а з точки зору внутрішньої структури і технології розробки - двоїстий характер [2]. Це призводить до необхідності вести розробку в двох дуже різних, але одночасно взаємопов'язаних аспектах.

З одного боку, основним змістом Web-ресурсів є дані, заради представлення яких ресурс створюється, або контент. Звідси – завдання проектування контенту та організації взаємодії з користувачем, що з'явилися одночасно з народженням Інтернету. Це основний (і сформований перший) клас задач проектування у випадку, якщо Web-ресурс представлений набором статичних, заздалегідь створених і збережених на сервері Web-сторінок, пов'язаних посиланнями. Спочатку цей клас задач був єдиним.

З іншого боку, сучасні Web-ресурси являють собою розвинені програмні комплекси, що використовують різноманітні програмні засоби та інструменти для створення коду та системи управління базами даних (СКБД) для зберігання інформації та управління нею. Інформація, що пред'являється користувачеві, плюс можливості інтерактивної взаємодії з ресурсом (сайтом) є не що інше як результат роботи відповідних програм (точніше, сукупності програмних сценаріїв), основна частина яких функціонує на стороні сервера. Звідси -

завдання проектування Web-ресурсів як програмних комплексів.

Наведені положення ілюструються на рисунку 1.1.

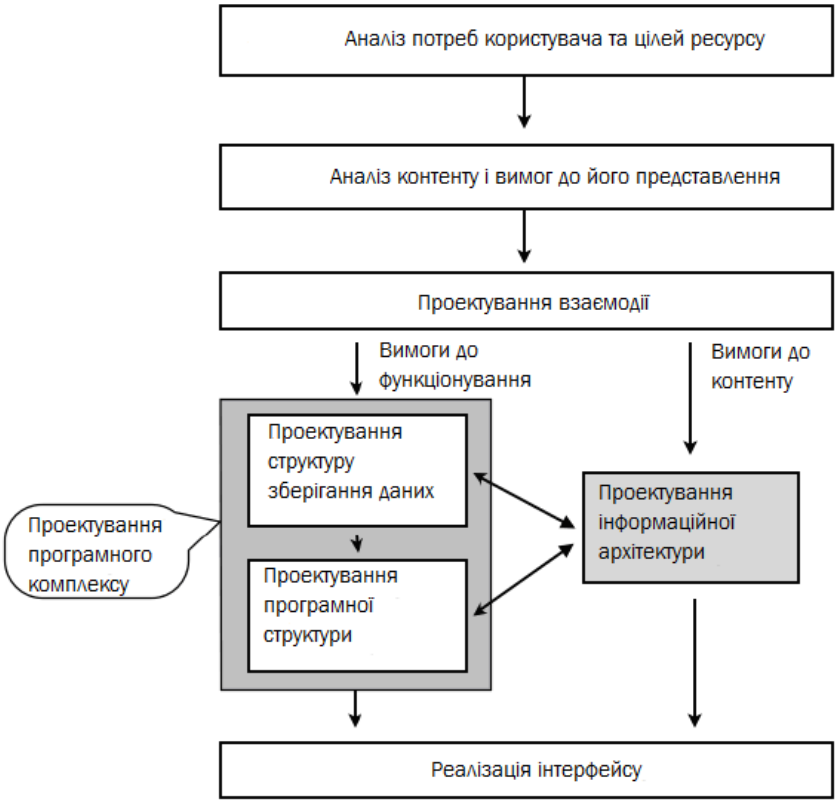


Рисунок 1.1 – Два аспекти проектування Web-ресурсу

Схема досить умовна. Зв'язки, що йдуть зверху вниз, означають передусім не тимчасову послідовність розробки, а факт реалізації верхніх блоків в підлеглих. Проте вона в загальних рисах дозволяє відобразити подвійність об'єкту, що розглядають.

Насправді проектування взаємодії, носієм якого є інтерфейс комп'ютерного продукту, як сукупність засобів, більша частина яких представлена візуальними елементами управління, пронизує (але, на жаль, далеко не завжди належним чином реалізується) практично всі смислові моменти проектування. Однак для користувача продукт представлений тільки своїм інтерфейсом.

Головне, що повинно бути закладено у проектування взаємодії - цілі користувачів, для яких розробляється продукт, і класи розв'язуваних ними

завдань. Теоретичною базою тут служать основні принципи когнітивної психології, які перетинаються з взаємодією людини з комп'ютером.

Розрізняють трирівневу, файлову та клієнт-серверну архітектуру проектування веб-ресурсів.

Трирівнева архітектура (англ. Three-tier) - архітектурна модель програмного комплексу, що припускає наявність у ньому трьох компонентів: клієнта, сервера додатків (до якого підключено клієнтський додаток) і сервера баз даних (з яким працює сервер додатків).

Клієнт – це графічний (інтерфейсний) компонент комплексу, який відображається кінцевому користувачеві. Даний рівень не повинен містити прямих зв'язків з базою даних (згідно вимог безпеки та масштабованості), має бути навантаженим основною бізнес-логікою (згідно вимог масштабованості) та зберігати стан додатку (згідно вимог надійності). Зазвичай, для даного рівня виноситься тільки найбільш проста бізнес-логіка: алгоритми шифрування, інтерфейсний вигляд авторизації, перевірка введених значень на відповідність формату та допустимість, прості операції із даними (групування, сортування, підрахунок значень), із завантаженими вже на термінал.

Сервер додатків (середній шар, сполучний шар) розташовується на другому рівні, на ньому зосереджена велика частина бізнес-логіки. Поза ним залишаються тільки фрагменти, що експортуються на клієнта (термінали), а також елементи логіки, занурені в базу даних (збережені процедури і тригери). Реалізація даного компонента забезпечується зв’язним програмним забезпеченням. Сервери додатків проектуються таким чином, щоб додавання до них додаткових примірників забезпечувало горизонтальне масштабування продуктивності програмного комплексу і не вимагало внесення змін в програмний код додатку.

Сервер баз даних (шар даних) забезпечує зберігання даних і виноситься на окремий рівень, який реалізується, як правило, засобами систем управління базами даних. Підключення до цього компоненту забезпечується тільки з рівня сервера додатків.

У найпростіших конфігураціях всі компоненти або частина з них можуть бути суміщені на одному обчислювальному вузлі. У продуктивних конфігураціях, як правило, використовується виділений обчислювальний вузол для сервера баз даних або кластер серверів баз даних, для серверів додатків – виділена група обчислювальних вузлів, до яких безпосередньо підключаються клієнти (термінали).

Файл-сервер - це окремо виділений сервер, який використовується для виконання файлових операцій введення-виведення та збереження файлів будь-якого формату.

Файл-серверні додатки – це додатки, які структурно схожі з локальними програмами і використовують мережевий ресурс для збереження даних окремими файлами. В такому випадку, зазвичай, функції сервера є обмеженими в зберіганні даних (також можливе зберігання виконуваних файлів), а обробка даних виконується лише на стороні клієнта. Зважаючи на те, що одночасний доступу на запис до одного файлу є неможливим, то кількість клієнтів є обмежена. Проте, якщо клієнти звертаються до файлів виключно в режимі читання, то їх може бути в рази більше.

Перевагами файл-серверів є:

– невисока вартість розробки;

– висока швидкість розробки;

– низька вартість оновлення і зміни ПЗ [1].

Недоліками є:

– при зростанні числа клієнтів обсяг трафіку різко збільшується і навантаження на мережі передачі даних;

– великі витрати на модернізацію і супровід сервісів бізнес-логіки на кожній клієнтській робочій станції;

– мала надійність системи [1]

Основним принципом технології "клієнт-сервер" є поділ функцій додатка на три групи:

– введення і відображення даних (взаємодія з користувачем);

– прикладні функції, які є характерними для предметної області;

– функції для управління ресурсами (файловою системою, базою даних і т.д.) [2].

Тому, для будь-якого додатку виокремлюють такі компоненти:

– компонент подання даннях;

– прикладний компонент;

– компонент для управління ресурсом [2].

Для з’єднання компонентів використовують правила, які називають "протокол взаємодії".

Логічний поділ додатку на дві і більше частин є характерною ознакою таких додатків, при цьому кожна з яких може виконуватися на окремому комп'ютері.

Окремі частини даного додатку функціонують один з одним (рисунок 1.2), здійснюючи обмін повідомленнями в заздалегідь визначеному форматі.

Рисунок 1.2 – Функціонування клієнт-серверної архітектури додатку

У цьому випадку, двохрівнева архітектура клієнт-сервер стає триланковою, а в деяких випадках, вона може складатися і з більшої кількості ланок.

1.2 Модель-Представлення-Контролер

MVC (Model-view-controller, «Модель-представлення-поведінка», «Модель-представлення-контролер») - це шаблон проектування додатків, при якому керуюча логіка поділена на три окремих компонента таким чином, що модифікування одного з них дає мінімальний вплив на інші.

Призначення моделі представлення зрозуміло і без пояснень – ця модель спеціально розроблена для використання в перегляді веб-ресурсу. Вона надає спрощений інтерфейс над доменною моделлю, який зводить прийняття рішень в уявленні до мінімуму.

Шаблон MVC добре застосовувати при створенні складних проектів, де необхідно відокремити роботу php програміста (або розділити групу програмістів на відділи), дизайнера, верстальника, і т.д.

Шаблон MVC розділяє уявлення, дані, і обробку дій користувача на три окремих компоненти:

Для наочності схеми дії шаблону MVC, нижче подана ілюстрація (рисунок 1.3), де суцільними лініями показані прямі зв’язки, а переривчастими лініями зображені непрямі.

MVC Модель (Model) – це модель, яка надає дані (зазвичай для View), а також реагує на запити (зазвичай від контролера), змінюючи свій стан.

MVC Представлення (View) відповідає за відображення інформації (користувальницький інтерфейс).

MVC Контролер (Controller) інтерпретує дані, введені користувачем, та інформує модель і уявлення про необхідність відповідної реакції (рисунок 1.3).

Такі компоненти, як уявлення і поведінка, залежать від моделі, але ніяк не впливають на неї. Модель може мати кілька варіантів представлень. Можливо, концепція MVC складна для розуміння, але якщо її осмислити, вона стає незамінною при розробці додатків на PHP.

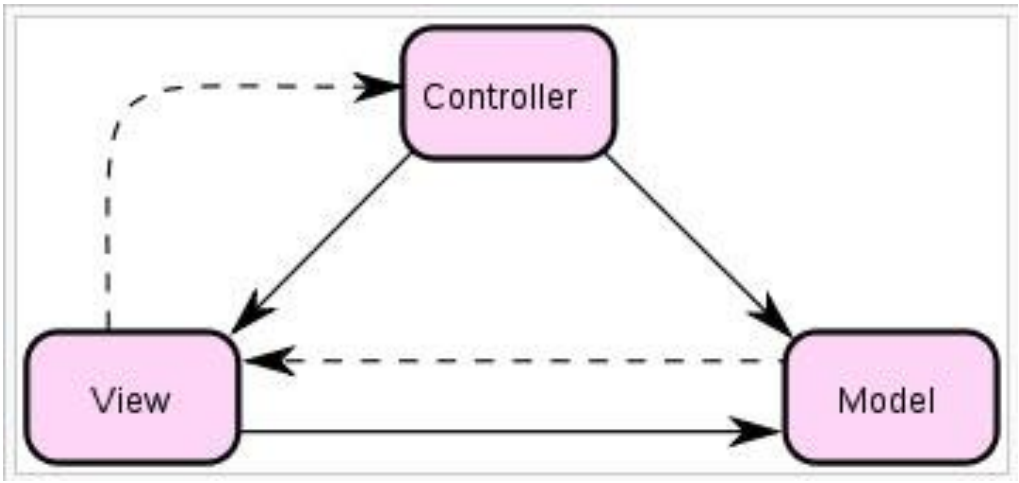


Рисунок 1.3 – Концепція Model-view-controller.

Особливістю при використанні MVC в PHP, є те, що існує одна точка входу в php додаток, яка, наприклад, досягається шляхом створення index.php через який будуть оброблятися всі запити.

Спочатку перевіряється існування файлу, на який роблять запит, і якщо його немає, то йде перенаправлення на index.php, інакше навіть запити картинок сайту будуть перенаправлятися на індекс. Якщо немає можливості використовувати ModRewrite в своєму додатку, то доведеться робити переадресацію вручну.

Модель відповідає за інформацію (дані) програми та правила управління цими даними. Моделі в основному використовується для керування відповідною таблицею бази даних і правил взаємодії з нею. У більшості випадків, кожна таблиця в базі даних буде відповідати одній моделі у вашому додатку. Основна частина бізнес-логіки додатка буде зосереджена в моделях.

Контролер керує запитами користувача (отримуються у вигляді запитів HTTP GET або POST, коли користувач натискає на елементи інтерфейсу для виконання різних дій). PHP контролери отримують запити користувачів, які прямують через index.php, і відповідно до них, коректують роботу моделі. Його основна функція - викликати і координувати дію необхідних ресурсів та об'єктів, потрібних для виконання дій, що задаються користувачем. Зазвичай контролер викликає відповідну модель для задачі і вибирає підходящий вид. Тобто контролер відстежує зміну в моделі і створює або змінює інтерфейс php

програми.

Вид забезпечує різні способи подання даних, які отримані з моделі. Він може бути шаблоном, який заповнюється даними. Може бути кілька різних видів, і контролер вибирає, який підходить найкраще для поточної ситуації. Тобто, представлення відповідає за користувальницький інтерфейс програми. Найчастіше це HTML файли з вставками PHP коду виключно виведення даних. Цей шар відповідає за виведення даних у веб-браузер або інший інструмент який звертається до додатку.

Веб додаток зазвичай складається з набору контролерів, моделей і видів.

Працює даний PHP MVC шаблон таким чином. При зверненні користувачем за потрібною url вибирається відповідний контролер, який звертається до представлення і моделі, і виводиться інформація. Іншими словами контролер в mvc є сполучною ланкою моделі і представлення.

Як згадувалося вище MVC шаблон, перш за все диференціація розробників php сайту на відділи. Також збільшується швидкість роботи php програми, якщо створюється великий проект. Ну і те, що стосується безпосередньо самого php розробника, це правильна структуризація php коду (все на своїх місцях, так легше для розуміння).

Найочевидніше перевагою є те, що розробник отримує від використання концепції MVC – це чіткий поділ логіки подання (інтерфейсу користувача) і логіки програми.

Підтримка різних типів користувачів, які використовують різні типи пристроїв є спільною проблемою наших днів. Інтерфейс, що надається повинен відрізнятися, якщо запит приходить з персонального комп'ютера або з мобільного телефону. Модель повертає однакові дані, єдина відмінність полягає в тому, що контролер вибирає різні види для виведення даних.

Крім ізолювання видів від логіки додатка, концепція MVC істотно зменшує складність великих додатків. Код виходить набагато більш структурованим, і, тим самим, полегшується підтримка, тестування та повторне використання рішень.

В загальному виділяють такі два варіанти взаємодії моделі, представлення та контролера.

У першому варіанті (рисунок 1.4) модель і представлення нічого не знають один про одного. Відповідальність за коректне відображення моделі повністю покладається на контролер.

Рисунок 1.4 – Схема взаємодії між модулями для першого варіанту

Найпростіша схема взаємодії між трьома модулями для другого варіанту представлена на рисунку 1.5. При отриманні вхідних даних контролер повинен повідомити про це моделі. Модель обробляє дані. Отримавши результат, модель в свою чергу повинна проінформувати про це представлення.

Іноді також можливий зв'язок між контролером і поданням (наприклад, для того, щоб повідомити користувачеві про помилку вводу).

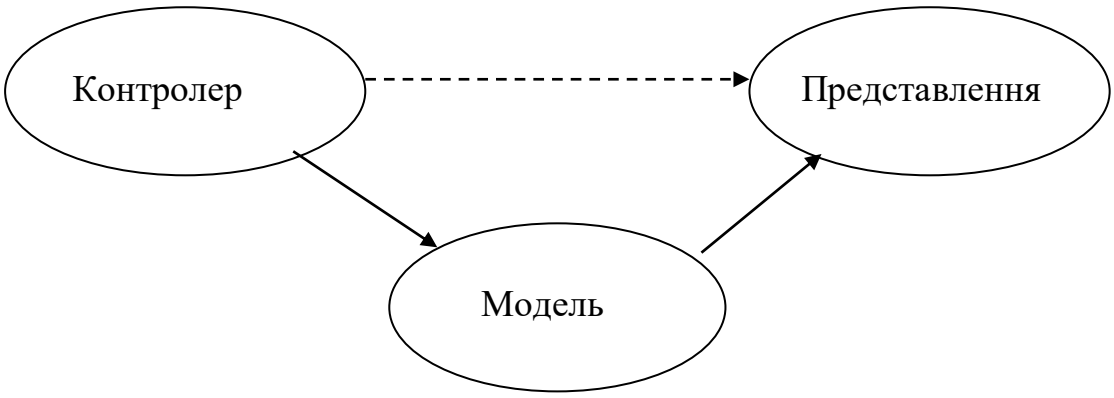


Рисунок 1.5 – Схема взаємодії між модулями для другого варіанту

З однією моделлю може бути пов'язано кілька виглядів. Наприклад, одні

й ті ж дані можуть бути представлені у вигляді тексту, графіка та діаграми.

При цьому можна збільшувати кількість представлень, не змінюючи саму модель.

Основна перевага організації додатків у відповідності зі схемою Model-View-Controller полягає в тому, що такий додаток надалі легше модифікувати. Наприклад, у разі зміни критеріїв виведення даних достатньо змінити модуль View, що відповідає за виведення інформації. При цьому модулі Controller і Model залишаться недоторканими.

1.3 Аналіз області застосування

Найоптимальнішим способом зберегти покриття підлоги від забруднення і передчасного зносу, це - установка брудозахисних покриттів. Одним з видів брудозахисних покриттів є, брудозахисні алюмінієві решітки.

Фахівці, рекомендують встановлювати брудозахисні алюмінієві решітки в місцях з максимальною прохідністю: вокзали, аеропорти, супермаркети, банки, готелі, бізнес і торгові центри тощо. Алюмінієві брудозахисні решітки встановлюються, як правило, при вході, або в тамбурному зоні приміщення в приямок або на поверхні підлоги застосовуючи при це зовнішнє обрамлення.

Необхідно відзначити, що брудозахисні алюмінієві решітки крім своєї зносостійкості, витримують періодичну навантаження до 1500 кг і володіють протиковзким ефектом, так само висока стійкість до корозії, яка забезпечує нормальну експлуатацію в будь-яку погоду, будь то сніг або дощ.

Невід'ємним перевагою брудозахисних алюмінієвих решіток, це заміна її чистячих елементів без подальшого ремонту або заміни основного алюмінієвого каркаса. Що в свою чергу, скоротить витрати на обслуговування і заощадить гроші.

Придверні решітки бувають стандартного розміру, а також виготовляються за розмірами і формою замовника.

На рисунку 1.6 наведені придверні решітки з використанням обрамлень

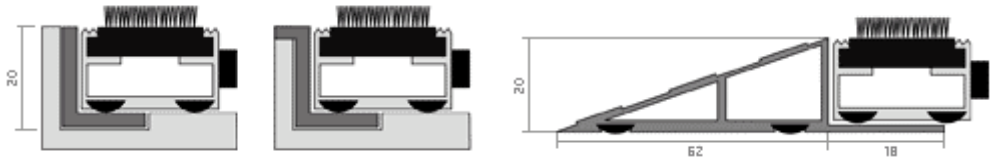


Рисунок 1.6 - Придверні решітки з використанням обрамлень

На рисунку 1.7 наведені придверні решітки з миючим елементом з гуми.

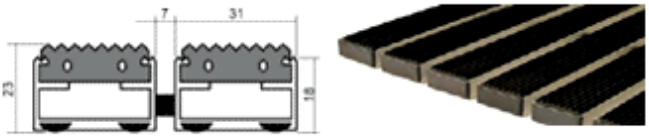


Рисунок 1.7 - Придверні решітки з миючим елементом з гуми

На рисунку 1.8 наведені вид придверних решіток з миючим елементом з гуми, оснащена скребком для очищення підошов взуття від грудок налиплого бруду.

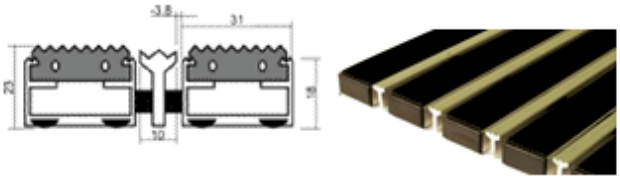


Рисунок 1.8 - Придверні решітки з миючим елементом з гуми, оснащена скребком для очищення підошов взуття від грудок налиплого бруду.

На рисунку 1.9 наведений вигляд придверних решіток з миючим елементом текстилю. Рисунки 1.10 – 1.13 ілюструють ще декілька різновидів придверних решіток.

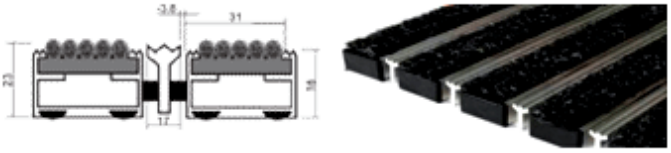


Рисунок 1.9 - Придверні решітки з миючим елементом текстилю. Оснащена скребком для очищення підошов взуття від налиплого бруду.

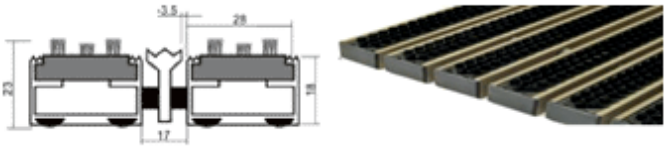


Рисунок 1.10 - Придверні решітки з миючим елементом щетини. Оснащена скребком для очищення підошов взуття від налиплого бруду.

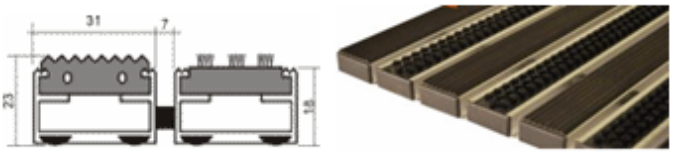


Рисунок 1.11 - Придверні решітки з очисними елементами з гуми і щетини.

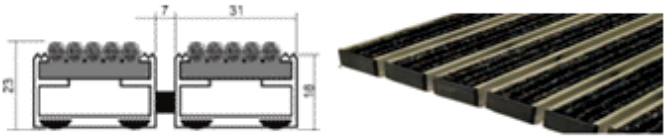


Рисунок 1.12 - Придверні решітки з очисним елементом з текстилю. М'яко очищає підошви взуття.

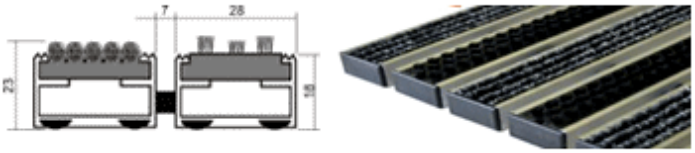


Рисунок 1.13 - Придверні решітки з очисними елементами з текстилю та щетини.

Для замовлених виробів, максимальна ширина (довжина профілю) становить 3 метри, довжина не обмежена. Рекомендується установка виробу в облаштований приямок.

Брудозахисні решітки мають високу стійкість до корозії і перепадів температури від -40 до + 70°С.

1.4 Постановка задачі дослідження

В даному магістерському проекті повинна бути досліджена та розроблена система для автоматизації проектування конструкцій придверних решіток. Виходячи з назви системи, зрозуміло, що вона структурно відображатиме архітектуру такої системи і складатиметься з трьох частин, а саме: моделей, тобто окремих вузлів виконуваної програми, виду, який відображатиме інтерфейс системи та контролера, який зв’яже моделі та відповідні їм інтерфейси.

Дана система буде розроблена для керування вмістом веб-орієнтованої системи інтернет-ресурсу. Таке програмне забезпечення дозволяє оперувати даними незалежно від місця знаходження користувача, оскільки завантажене на сервері. Використовують такі системи для створення інтернет-магазинів, форумів, соціальних мереж, тощо. Хоч звичайні користувачі не бачать внутрішньої структури роботи веб-сайтів, проте такі програми набули широкого поширення.

Архітектура веб-орієнтованої системи інтернет-ресурсу повинна забезпечити безпечне відділення функціоналу, представлення та користувацьких дій. Тобто, дій користувачів повинні лише впливати на вибір інформації а не на хід роботи системи. Функціонал системи може бути змінений лише компетентними працівниками чи самими розробниками. Представлення має обиратися програмою самостійно відповідно до вибраної користувачем інформації.

Одним з головних етапів розробки програмного забезпечення є робота з базами даних. Необхідно здійснити розробку відповідної бази даних, згідно поставленому завданню. ЇЇ структура повинна вміщати всі розділи, які потрібні для відображення інформації. Табличні дані відображатимуть зв’язки за схемою: один до одного, один до багатьох і багато до багатьох, і які в свою чергу задаються відповідно до первинного ключа таблиці.

Як згадано вище, дане програмне забезпечення міститиме блок модель.

Для кожної такої моделі і вибраної користувачем інформації повинен бути розроблений відповідний інтерфейс, тобто блок представлення. Вигляд відображення даних повинен обиратися самою програмою в залежності від запиту користувача. За це відповідатиме окремий блок програми – контролер, який пов’язує роботу блоків моделі і представлення.

Користувацька URL-адреса – це коротка веб-адреса, яка легко запам’ятовується та спрямовує відвідувачів просто на сторінку відповідного веб-ресурсу. Щоб отримати такий адрес потрібно або купити, або використати безкоштовний домен. Проте, якщо обрати другий варіант, то URL-адреса міститиме назву спершу цього сайту, який надає послугу безкоштовного домену, а це, як правило, зменшує довіру користувачів.

Основою для розробки є завдання на магістерський проект, затверджений кафедрою комп’ютерної інженерії факультету інженерії програмного забезпечення Тернопільського національного технічного університету.

Метою розробки та дослідження системи є розробка алгоритму, модулів та файлів для побудови архітектури веб-орієнтованої системи інтернет-ресурсу для автоматизованого проектування конструкцій придверних решіток.

Програмне забезпечення, яке буде розроблене в даному магістерськомуу проєкті базуватиметься на понятті об’єктно-орієнтованого програмування. Тому однією з функціональних вимог програми є те, що розроблені класи повинні забезпечувати можливість виконання таких функцій, як класи для роботи з базою даних, робота з системними та користувацькими таблицями, розробка класів для моделей, представлення і для контролерів.

Розглянемо ці класи детальніше. Перший клас для роботи з базою даних повинен відповідати за її з’єднання з програмою. Даний клас матиме захищений модифікатор доступу. Тобто, звернення до об’єкта класу можливі тільки з методів того класу, в котрому цей об’єкт є визначеним. Будь-які нащадки класу вже не можуть отримати доступ до його об’єктів. Наслідування за типом private не дозволяє доступ з дочірнього класу до об’єктів батьківського класу без виключень, в тому числі навіть public-об’єкти.

Ще один клас, який повинна містити програма, це клас для роботи з системними таблицями. В таких таблицях розміщується інформація про структуру самої бази даних (назва, тип, розміри колонок таблиці, тощо). Проте, вони використовують не лише для внутрішніх потреб СУБД, а й звичайні користувачі можуть мати до них доступ. Але, в цілях безпеки, вносити зміни в системні таблиці можна лише адміністраторам, тоді як користувачам модифікація даного розділу недоступна.

Наступний клас – це клас для роботи з користувацькими таблицями. Судячи з назви, стає зрозумілим, що в таких таблицях зберігаються дані, котрі були введені самим користувачем. Проте задавати параметри таблиці, такі як додаткові колонки, міняти їх назви, типи, кількість введених символів в комірку, зазвичай, можуть лише адміністратори. І як було описано вище, ці дані записуються в системні таблиці.

Необхідно розробити клас моделі, представлення (виду, вигляду) та контролеррів. Модель - містить бізнес-логіку програми і включає методи вибірки, обробки (наприклад, правила валідації) і надання конкретних даних, що часто робить її дуже об’ємною, що цілком нормально.

Модель не повинна безпосередньо взаємодіяти з користувачем. Всі змінні, що відносяться до запиту користувача повинні оброблятися в контролері.

Модель не повинна генерувати HTML або інший код відображення, який може змінюватися в залежності від потреб користувача. Такий код повинен оброблятися в видах.

Одна і та ж модель, наприклад: модель аутентифікації користувачів може використовуватися як в користувальницькій, так і в адміністративній частині програми. У такому випадку можна винести загальний код в окремий клас і успадковувати від нього нащадків, в яких визначатимуться специфічні для підпрограм методи.

Представлення має використовуватися для завдання зовнішнього відображення даних, отриманих з контролера і моделі.

Клас представлення повинен містити HTML-розмітку і невеликі вставки PHP-коду для обходу, форматування і відображення даних. Даний клас не повинен безпосередньо звертатися до бази даних. Цим повинні займатися моделі. Також, не має працювати з даними, отриманими із запиту користувача. Це завдання має виконувати контролер.

Проте, може безпосередньо звертатися до властивостей і методів контролера або моделей, для отримання готових до висновку даних.

Клас представлення, для зручності буде поділений на загальний шаблон, що містить розмітку, загальну для всіх сторінок (наприклад, шапку і підвал) і частини шаблону, які використовують для відображення даних виведених з моделі або відображення форм введення даних.

Контролер повинен виступати сполучною ланкою, що з'єднує моделі, види та інші компоненти в робочий додаток. Контролер відповідає за обробку запитів користувача. Контролер не повинен містити SQL-запитів. Їх краще тримати в моделях. Контролер не повинен містити HTML і інший розмітки. Її варто виносити в представлення.

Логіка контролера досить типова, тому більша її частина має виноситися в базові класи. Моделі, навпаки, дуже об’ємні і містять велику частину коду, пов'язану з обробкою даних, тому структура даних і бізнес-логіка, що міститься в них, звичайно досить специфічна для конкретного додатка.

Додаткових вимог до вихідних кодів та мов програмування не пред’являються.

Передбачається захист інформації шляхом аутентифікації користувача. Щоб увійти в систему, він повинен ввести логін та пароль. Нового користувача може додати лише адміністратор, шляхом введення в базу даних його дані. З метою забезпечення даних, усі паролі в базі даних повинні бути зашифровані. Для цього можна застосувати метод шифрування SHA2. Процес проектування в вигляді алгоритму наведено на рисунку 1.14.

Щодо спеціальних вимог, то система повинна мати інтуїтивно-зрозумілий та простий інтерфейс.

Рисунок 1.14 – Алгоритм проектування придверних решіток

Процес проектування придверних решіток повинен складатись з наступних кроків:

1. Створення на онлайн-порталі проекту решітки на основі попереднього замовлення.

2. Підготовка зображення контуру майбутнього виробу в будь-якому графічному редакторі у форматі JPEG.

3. Завантаження підготовленого контуру в проект.

4. Аналіз контуру на сервері, виділення його основних характеристик та збереження їх в базі даних.

5. Проведення розрахунків.

6. Формування документації для працівників цеху, для списання на склад та формування файлу у форматі CSV для імпорту в систему 1С-Бухгалтерія.

Розмір зображення контуру може будь-яким, але потрібно врахувати, що при зменшенні його розмірів буде погіршуватися точність розрахунків. Оптимальним є співвідношення один до одного: 1 мм виробу до 1 пікселя на зображенні. Головною вимогою до контуру майбутнього виробу є збереження пропорцій між всіма його відрізками.

Найбільш ресурсоємною є операція аналізу контуру і виділення його основних характеристик. Тому вона здійснюється лише один раз для кожного контуру. В подальшому це дозволяє швидко здійснювати повторні розрахунки для проекту при зміні другорядних характеристик виробу (кількість ліній стяжки, зазор між елементами конструкції тощо).

Розрахунки і формування документації здійснюються в режимі реального часу на основі параметрів проекту та особливостей контуру виробу. Результати відображаються в браузері для попереднього перегляду з можливістю завантаження на комп'ютер користувача.

Загалом, впровадження проекту повинне привести до наступних результатів:

- час проектування майбутніх виробів зменшився в середньому в 10 разів, що збільшило швидкість виконання замовлень та зменшило простої виробництва;
- зникла необхідність у використанні спеціалізованого інженерного продукту Компас-3D, що дало можливість здійснювати проектування не лише інженерам, а й просто впевненим користувачам комп’ютера, які мають навички роботи з будь-яким графічним редактором;

- прямий економічний ефект пов’язаний з усуненням необхідності використання САПР Компас-3D та щорічним придбанням ліцензії.

Висновки до розділу 1

1. Проведено аналіз існуючих архітектур проектування веб-ресурсів, що дає змогу обрати найкращу для поставлених цілей архітектуру.

2. Проаналізовано обрану архітектуру веб – ресурсу, що дозволяє виділити її сильні та слабкі сторони.

3. Проведено постановку цілей дослідження та задач проектування системи для автоматизації проектування придверних решіток.

2 ДОСЛІДЖЕННЯ ТА РОЗРОБКА АРХІТЕКТУРИ ІНТЕРНЕТ РЕСУРСУ

2.1 Дослідження інструментарію розробки

Для реалізації даного проекту будемо використовувати стек технологій PHP, JavaScript, MySQL. Такий вибір зумовлений тим, що у клієнта вже є онлайн-рішення для зберігання/супроводу вхідних замовлень та база даних номенклатури продукції на цьому стеку.

Головним фактором мови PHP є практичність. PHP повинен надати програмісту кошти для швидкого і ефективного вирішення поставлених завдань. Практичний характер PHP обумовлений п'ятьма важливими характеристиками:

- традиційністю;
- простотою;
- ефективністю;
- безпекою;
- гнучкістю.

Багато конструкцій мови запозичені з Сі, Perl. Код PHP дуже схожий на той, який зустрічається в типових програмах на C або Pascal. Це помітно знижує початкові зусилля при вивченні PHP. PHP - мова, що поєднує переваги Perl і Сі і спеціально націлена на роботу в Інтернеті, мова з універсальним і зрозумілим синтаксисом.

І хоча PHP є досить молодою мовою, вона знайшла таку популярність серед web-програмістів, що на даний момент є мало не найпопулярнішою мовою для створення web-додатків (скриптів).

Сценарій PHP може складатися з 10 000 рядків або з одного рядка - все залежить від специфіки завдання. Не доведеться довантажувати бібліотеки, вказувати спеціальні параметри компіляції або що-небудь в цьому роді. Механізм PHP просто починає виконувати код після першої екрануючої послідовності (<?). І продовжує виконання до того моменту, коли він зустріне парну екранує послідовність (?>). Якщо код має правильний синтаксис, він виконується в точності так, як вказано.

PHP - мова, яка може бути вбудована безпосередньо в html-код сторінок, які, в свою чергу будуть коректно оброблятися PHP -інтерпретатором. Можна використовувати PHP для написання CGI-сценаріїв і позбутися великої кількості незручних операторів виведення тексту. Можна залучати PHP для формування HTML-документів, позбувшись від безлічі викликів зовнішніх сценаріїв.

Велика розмаїтість функцій PHP позбавлять від написання багаторядкових призначених для користувача функцій на C або Pascal.

Ефективність є виключно важливим фактором при програмуванні для багатокористувацьких середовищ, до числа яких належить і web.

Важлива перевага PHP полягає в «движку». «Движок» PHP не є ні компілятором, ні інтерпретатором. Така архітектура «движка» PHP дозволяє обробляти сценарії з достатньо високою швидкістю.

За деякими оцінками, більшість PHP-сценаріїв (особливо не дуже великих розмірів) обробляються швидше аналогічних їм програм, написаних на Perl.

Але продуктивність PHP цілком достатня для створення цілком серйозних web-додатків.

PHP надає в розпорядження розробників і адміністраторів гнучкі і ефективні засоби безпеки, які умовно поділяються на дві категорії: засоби системного рівня і засоби рівня додатків.

У PHP реалізовані механізми безпеки, що знаходяться під управлінням адміністраторів; при правильному налаштуванні PHP це забезпечує максимальну свободу дій і безпеку. PHP може працювати в так званому безпечному режимі (safe mode), який обмежує можливості застосування PHP користувачами по ряду важливих показників. Наприклад, можна обмежити максимальний час виконання і використання пам'яті (неконтрольована витрата пам'яті негативно впливає на швидкодію сервера). За аналогією з cgi-bin адміністратор також можна встановлювати обмеження на каталоги, в яких користувач може переглядати і виконувати сценарії PHP, а також

використовувати сценарії PHP для перегляду конфіденційної інформації на сервері (наприклад, файлу passwd).

У стандартний набір функцій PHP входить ряд надійних механізмів шифрування. PHP також сумісний з багатьма додатками незалежних фірм, що дозволяє легко інтегрувати його з захищеними технологіями електронної комерції (e-commerce). Інша перевага полягає в тому, що вихідний текст сценаріїв PHP можна переглянути в браузері, оскільки сценарій компілюється до його відправки за запитом користувача. Реалізація PHP на стороні сервера запобігає викраденню нетривіальних сценаріїв користувачами, знань яких вистачає хоча б для виконання команди View Source.

Оскільки PHP є вбудовуваною (embedded) мовою, вона відрізняється винятковою гнучкістю по відношенню до потреб розробника. Хоча PHP зазвичай рекомендується використовувати в поєднанні з HTML, він з таким же успіхом інтегрується з JavaScript, WML, XML та іншими мовами.

При написанні веб-додатків, програмування на JavaScript використовується найбільш часто. Якщо коротко перерахувати ключові особливості даної мови, то слід виділити наступне:

- Об'єктно-орієнтованість. Виконання програми є взаємодія об'єктів;
- Приведення типів даних проводиться автоматично;
- Функції виступають об'єктами базового класу. Ця особливість робить JavaScript схожим на багато функціональні мови програмування, такі як Lisp і Haskell;
- Автоматичне очищення пам'яті. Так зване, прибирання сміття робить JavaScript схожим на C # або Java.

Якщо говорити про суть застосування JavaScript, то ця мова дозволяє «оживляти» нерухомі сторінки сайтів за допомогою коду, який можна запустити на виконання (так звані, скрипти).

Якщо говорити про синтаксис JavaScript, то йому притаманні такі особливості:

- Регістр важливий. Функції з назвами func () і Func () - абсолютно різні;

- Після операторів необхідно ставити крапку з комою;
- Вбудовані об'єкти і операції;
- Прогалини не враховуються. Можна використовувати скільки завгодно відступів, а також перекладів рядка, щоб оформити свій код.

Відзначимо недоліки. Необхідність забезпечувати кросбраузерність. Код повинен коректно виконуватися у всіх, або хоча б найпопулярніших, браузерах;

Система успадкування в мові викликає труднощі в розумінні того, що виконується. В JavaScript реалізовано успадкування, засноване на прототипах.

Відсутня стандартна бібліотека. JavaScript не надає ніяких можливостей для роботи з файлами, потоками введення-виведення і іншими речами;

Синтаксис в цілому ускладнює розуміння. Тепер варто відзначити деякі переваги

JavaScript надає велику кількість можливостей для вирішення найрізноманітніших завдань. Гнучкість мови дозволяє використовувати безліч шаблонів програмування стосовно до конкретних умов. Винахідливий розум отримає справжнє задоволення;

Популярність JavaScript відкриває перед програмістом чимала кількість готових бібліотек, які дозволяють значно спростити написання коду і нівелювати недосконалості синтаксису;

Застосування в багатьох областях. Широкі можливості JavaScript дають можливості використання коду в різних технологіях.

2.2 Проектування ієрархії класів для рівня Model

Як було згадано вище, компонент «Model» призначений для роботи з базою даних. В класах даної ієрархії будуть виконуватися запити на здійснення вибірки даних та їх редагування, а також видалення та додавання нових записів.

Для виконання даних запитів буде використовуватися синтаксис мови MySQL. Для цього будуть використовуватися такі основні команди, як select, update, insert into, delete, тощо. Для розширення можливостей вибірки будуть використовуватися додаткові оператори мови MySQL.

Як видно на рисунку 2.1, клас Model є батьківським класом. Основна його функція – це виклик функції для створення з'єднання з базою даних. Кожен клас, який буде його наслідувати буде автоматично під'єднаним до бази даних. Тобто не потрібно кожного разу описувати функцію з'єднання, що в свою чергу зменшить об'єм коду та підвищить ефективність роботи програми не витрачаючи час на обробку зайвих функцій.

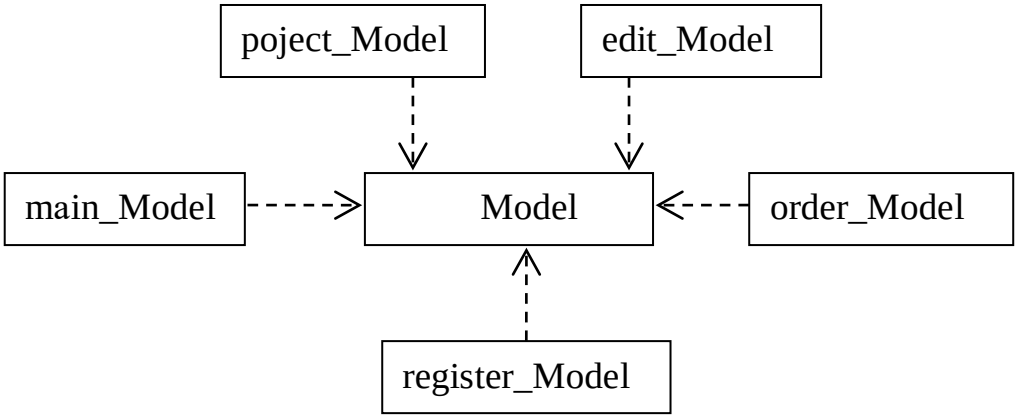


Рисунок 2.1 – Ієрархія класів компоненту «Model»

Клас main_Model буде призначений для виконання запитів на вибірку для головної сторінки. Тут повинне здійснюватися опрацювання замовлень та бізнес логіка.

Дані запити будуть виконуватися за допомогою команди select. Згідно синтаксису команди mysql, вибірка здійснюватиметься на основі заданих полів

вибраних таблиць. Також повинен вказуватися параметр, за яким буде здійснюватися вибірка, а саме: останні три записи, тобто відбуватиметься сортування даних від останніх доданих замовлень до перших і відбиратиметься обмежена кількість записів командою limit.

Клас article_Model є дочірнім класом для Model і успадковує функцію з'єднання з базою даних. Він буде призначений для роботи з таблицею order, в якій записані усі замовлені вироби. Цей же клас буде використовуватися для перегляду написаних статей з реалізацією посторінкової навігації.

Функція класу order_Model повинна використовуватися для одержання даних. В даному випадку, результатом повернення функції має бути значення назви заголовка замовлення та текст.

Ще одна команда, яка має бути реалізована в класі order _Model, це команда, яка дозволить отримати номер замовлення, на котрий користувач забажає перейти, та передаватиме даний параметр у запит MySQL.

Отже, в результаті реалізації цієї моделі, користувач зможе переглядати замовлення, тим самим полегшуючи пошук необхідної інформації.

Клас firm_Model успадковуватиме від класу Model певні, доступні методи, як дочірній клас. інша частина функцій, які будуть доступні в цьому класі будуть доступні лише для нього.

Даний клас повинен бути схожим до вищеописаного article_Model, проте призначений вже буде для роботи з таблицею firm. Стає зрозумілим, що цей клас повинен надавати можливість пошуку необхідних організацій, підприємств чи товарів. Для більшого комфорту, теж повинна бути передбачена посторінкова навігація.

Вищезгадана навігація реалізовуватиметься аналогічним чином, як і в класі article_Model, тобто сторінка, на яку користувач бажає перейти, буде передаватися через контролер і параметром передаватиметься в запит MySQL для вибору потрібного запису з таблиці firm.

В результат реалізації даного класу, користувач отримає можливість ознайомитися з даними щодо фірм і товарів, які можна буде рекламувати в

даному веб-додатку.

Для того, щоб звичайні користувачі сайту могли писати статті, відгуки, рекламувати організації, фірми, товари та послуги, вони повинні бути зареєстрованими. Відповідно до режиму доступу, кожен такий клієнт зможе отримати набір деяких доступних йому функції. Так, наприклад, блогер зможе писати статті про певні товари, послуги чи фірми і підприємства. Або просто зможе писати відгуки. Якщо ж, користувач хоче прорекламувати свою фірму чи товари, то він зможе зареєструватися, як рекламодавець. Це надасть йому можливість роботи з таблицею firm, але він не матиме можливість залишати відгуки, задля уникнення непорозумінь, таких як залишення псевдо відгуків про власні послуги та товари або уникнення неправдивого обмовлення конкурентів.

Отже, для реєстрації користувачів буде створено клас register_Model. Цей клас міститиме методи, які дозволять отримати дані, що передав користувач для його реєстрації та здійснити відповідний запис в таблицю users, для надання ширшого спектру можливостей для клієнта, відповідно до його прав доступу.

Для написання та редагування текстів, а саме статей чи інформації про фірми та товари буде використовуватися клас edit_Model. Використовуючи функції батьківського класу Model, він вже буде з’єднаним з базою даних.

У відповідних полях, користувач введе необхідну інформацію і по натиску кнопки здійсниться запис чи редагування у відповідній таблиці. Дана команда буде описана в методі класу edit_Model. Для цього використається синтаксис мови MySQL. Всі необхідні дані для запису та умови для редагування кортежів будуть передаватися від класу компоненту «Controller».

Вищеописані класи дозволять веб-додатку взаємодіяти з базою даних, оскільки вміщуватимуть усі потрібні для цього запити. Всі дані, які користувач вводитиме у форму будуть отримуватися від компоненту «Controller». Відповідно, він теж отримає доступ до даних, отриманих в результаті виконання запиту.

Таким чином, як було передбачено для компонента «Model», в ньому виконуватимуться лише запити MySQL для роботи з базою даних. Функції для обробки цих даних чи для пошуку необхідних параметрів не передбачені для класів моделі.

2.3 Проектування ієрархії класів для рівня View

Як було описано у першому розділі, компонент View (Представлення) дозволить організувати простий та доступний інтерфейс для користувача. Основним батьківським класом буде клас «View». Він надасть можливість генерації будь-якої сторінки необхідної контролеру.

Ієрархію класів компонента представлення зображено на рисунку 2.2.

Клас «View» надасть можливість включати чи відключати потрібні шаблони в залежності від подання інформації. Також в ньому за замовченням будуть підключені так звані дефолтні значення. Тобто, якщо не буде обраного якого-небудь іншого вигляду, то він підключатиметься автоматично. Якщо і його не потрібно, то повинна бути передбачена функція відключення будь-якого шаблону.

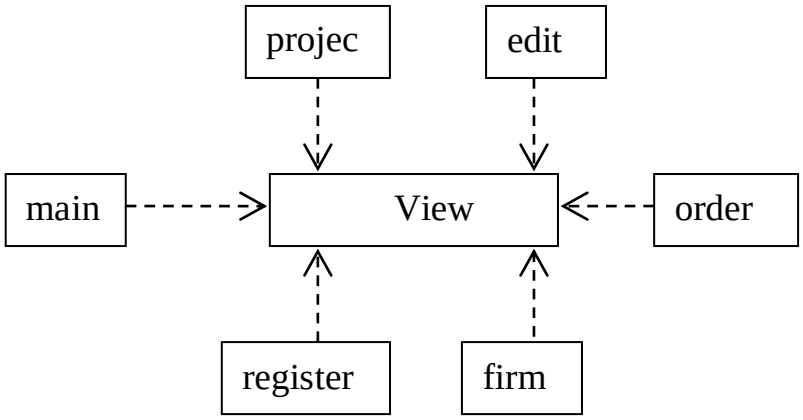


Рисунок 2.2 – Ієрархія класів компоненту «View»

За зовнішній вигляд головної сторінки сайту відповідатиме «main». Дана сторінка вміщуватиме блоки з найновішими статтями, блоки реклами, вікно аутентифікації користувача. Також міститиметься посилання на сторінку реєстрації, якщо клієнт захоче надавати будь-яку реклами чи писати свої публікації. Ці елементи займатимуть більшу частину головного вікна.

Буде розміщене меню сайту, де користувач зможе обрати послуги за

рубриками, які його цікавлять або почитати про проект.

Буде розташоване поле у правому верхньому куті сторінки, в якому користувач зможе задати пошук необхідної йому статті чи фірми за назвою і для запуску пошуку буде розташовано кнопка «Пошук».

Шаблон «register» міститиме форму для реєстрації користувача. Він складатиметься з полів: ім’я, дата народження, логін, пароль, підтвердження паролю. Обов’язково будуть виводитися підказки для введення даних. Так наприклад в полі «Ім’я» можуть бути лише літери, для поля «Дата народження» також буде встановлено формат введення даних. Поля «Пароль» та «Підтвердження паролю» будуть проходити перевірку на ідентичність введених даних. Заголовок сайту буде зберігатися.

Після того як всі поля будуть заповнені, лише тоді користувач зможе зареєструватися натиснувши кнопку «Зареєструватися» або очистити всі поля однією кнопкою «Очистити».

Шаблон «orderController» відображатиме перелік замовлень. Зверху зберігатиметься заголовок з усіма закладками рубрик та можливістю пошуку замовлень і фірм за назвою.

З правої сторони будуть розташовані вузькі смужки реклами фірм та товарів.

Самі замовлення займатимуть 80% сторінки. Буде розміщена її назва і короткий початок, нижче буде розташоване посилання «Читати далі», яке розгортатиме всю інформацію.

Розташовуватиметься по 10 статей на сторінку. Для перегляду наступної десятки буде розташована навігація по сторінках.

Шаблон «firmController» буде мати дещо схожий інтерфейс як і в «orderController», проте інформація про фірми буде розташована блоками і відображатиме всю інформацію про них загалом без додаткових навігацій.

Також з правої сторони замість реклами будуть розміщені публікації.

Для створення нового замовлення чи редагування вже існуючого буде використовуватися шаблон «edit». Він складатиметься з таких полів: назва,

поле для введення тексту самої публікації, рубрика, яка буде оформлена у вигляді випадаючого списку і реклами фірми.

Усі поля крім поля «Реклама фірми» будуть обов’язковими для заповнення.

Шаблон «project» міститиме простий текст про розроблений веб-ресурс. З правої сторони буде розташована реклама. Як і для будь якої сторінки зберігатиметься заголовок сайту з меню-навігацією по сайту та полем пошуку.

2.4 Проектування ієрархії класів для рівня Controller

Клас «Request» призначений для отримання даних з адресної стрічки будь-яким методом чи то Get, чи то Post. Функції даного класу повинні обробляти дані передані в url-адресі.

Одна функція повинна отримувати цілу адресну строку та ділити її за обраним параметром для виділення окремо адреси сторінки і окремо переданих даних. В залежності від потреби, в програмі повинно бути передбачено функції як для отримання окремого параметру, так і всі передані параметри для подальшої їх обробки.

Функція для передачі окремо вибраного параметру буде призначена для такого випадку, як, наприклад, для фільтрування даних при їх виборі з бази даних.

Для отримання коректних даних, повинна бути передбачена функція для очищення даних переданих раніше користувачем. Окрім цього, для обробки даних потрібні функції для визначення яким саме способом було передано ці дані.

Клас «Route» працюватиме безпосередньо з адресою сторінки. Отримавши повну url-адресу одна функція повинна відділити її від параметрів і розділити на каталоги. Це необхідне для вибору необхідних контролерів.

Ще одна функція передбачена цим класом повинна з попередньо отриманих даних, обрати необхідний модуль, контролер та потрібну дію. Відповідно до зазначених каталогів буде здійснюватися обробка і передача результатів від отриманих параметрів з класу «Request».

Клас «Route» і «Request» отримуватимуть всі необхідні дані з якими користувачу потрібно буде працювати і зберігатимуть в своїх новостворених об’єктах. Таким чином користувачеві не потрібно буде постійно оновлювати дані. Кожному класі, які будуть наслідувати головний контролер не потрібно буде робити окремі запити на дані, що в свою чергу підвищить швидкодію роботи сайту і зменшить обсяги передачі даних, таким чином звільнюючи

додаткові канали обміну інформацією.

Клас «FrontController» буде залежним від попередньо описаних класів «Route» і «Request».

Залежність проявляється в тому, що при зміні особливостей одного класу автоматично проявляються зміни в поведінці залежного класу. Тобто при зміні даних в об’єктах вищеописаних класів призводить до зміни поведінки об’єктів класу «FrontController».Таким чином параметри в залежному класі виступають об’єктами класів «Route» і «Request».

Даний клас призначений для створення об’єктів класу «Route» і «Request» та подальшої передачі значень для їх обробки головним контролером.

Як видно на рисунку 2.3, клас «Controller» залежить від класу «FrontController» та має ряд дочірніх класів, які будуть описані нижче. Даний клас призначений для об’єднання отриманих параметрів та виклику відповідних контролерів-обробників.

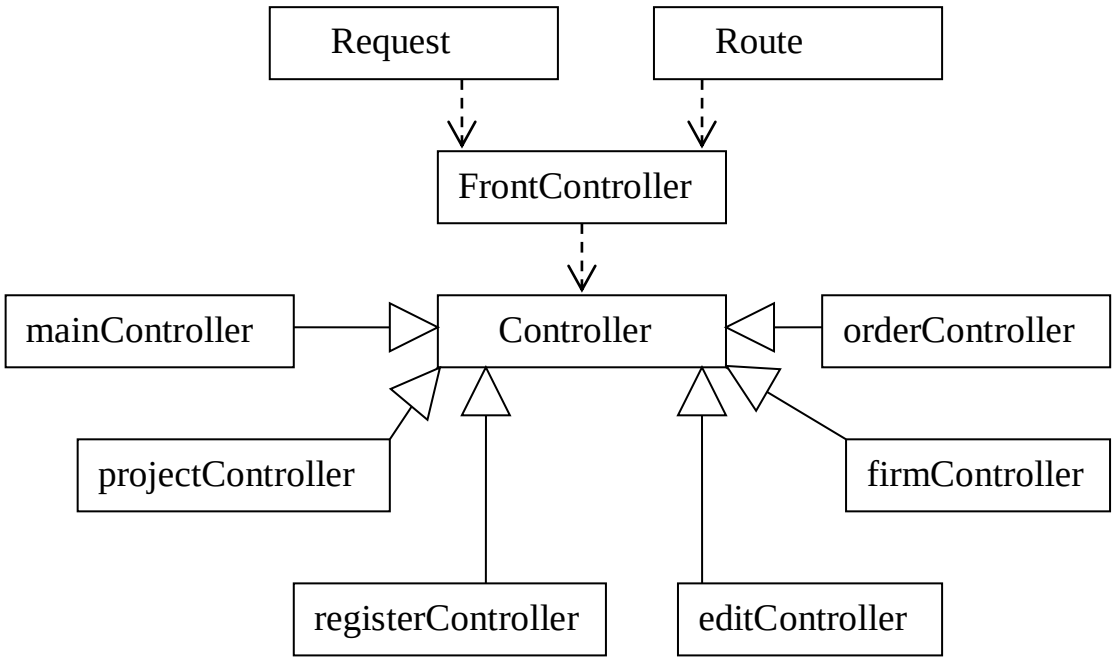


Рисунок 2.3 – Ієрархія класів компоненту «Controller»

Відповідно до заданого фільтру основний контролер вибирає контролер назва якого співпадає з заданим фільтром з приставкою Controller

Клас «mainController» призначений для генерування головної сторінки веб-додатку. Даний клас наслідує властивості класу «Controller», оскільки є дочірнім класом. Відповідно до отриманих параметрів і даних, основним контролером запуститься контролер «mainController» для подальшої обробки даних. Він, в свою чергу отримає дані з бази даних, які повинні відображатися на головній сторінці, використовуючи клас «Model» та «Select».

Одночасно, даний клас підключить необхідний інтерфейс веб-сторінки, який буде потрібний в даному випадку. Всі необхідні файли, які міститимуть опис сторінок, будуть керуватися в свою чергу класом «View», який взаємодіятиме з контролерами.

Клас «registerController», аналогічно попередньому класові виступає дочірнім класом і наслідуватиме доступні йому функції від «Controller». Він буде призначений для генерації даних при реєстрації нового користувача. Підключивши необхідний інтерфейс, в якому користувач, який бажатиме зареєструватися, зможе ввести всі свої необхідні дані для реєстрації. Після цього, отримавши дані з форми за допомогою методів описаних в класі «Request», контролер викличе клас «Insert», який виконає операції для запису даних в базу даних.

Клас «orderController» призначений для обробки даних при роботі користувача з різноманітними замовленнями. В веб-додатку повинна бути передбачена можливість вибірки публікацій як за свіжими статтями, що і буде визначатися автоматично по-замовчуванні, так і користувач матиме можливість обирати і впорядковувати статті по назві, по популярності, по категоріях і, навіть, по авторах.

Окрім цього, буде підключений відповідний зовнішній вигляд сторінки для зручного відображення інформації, запит на яку задасть користувач.

Даний клас буде викликатися через головний контролер, відповідно до поданого запиту і знайденого по фільтру класу-обробника, що описаний в класі «Request».

Клас «firmController» дещо схожий з попередньо описаним класом, проте

включатиме ряд відмінностей, що в свою чергу пояснює додаткове створення ще одного класу. Він також має цей самий батьківський клас «Controller».

Аналогічно «articleController» даний клас виводитиме дані з бази даних, проте ці дані відрізняються від статей. Даний обробник буде виводити інформацію про фірми, організації, компанії, що рекламуватимуться в веб-додатку, що проектується.

Контролер «firmController» також використовуватиме функції класу «Select» з компоненту модель, проте підключатиме інший інтерфейс компоненту представлення в порівнянні з «articleController».

Клас «editController» використовуватиметься для взаємодії компонентів моделі і представлення у випадку, коли користувач писатиме чи редагуватиме статтю або інформацію про фірму.

Для взаємодії всіх компонентів структури mvc в даному випадку, цей клас буде приймати дані з форми описані в файлі, що підключатиметься через клас «View». Для оновлення цих даних використовуватиме методи описані в класі «Update». При чому перш ніж приступити до редагування, контролер забезпечить використання функцій компоненту модель для внесення у форму даних, що вже були записані у базі даних.

Клас «projectController» буде досить простим в проектуванні і міститиме відносно постійну інформацію – а саме інформацію про проект. Всі дані про нього вже будуть описані безпосереднього в файлі компоненту представлення, тому не потрібні будь-які операції пов’язані з роботою з базою даних. Проте для організації цього класу, все таки потрібний головний контролер для його запуску при запиті від користувача і виведенню відповідного інтерфейсу.

Веб-сайт складається із сторінок, а точніше з відповідей на запити. Тобто, існує маршрутизатор, який повинен визначити, котрий метод якого контролера потрібно викликати. Тому мають зазначатися два основні параметри, контролер та метод.

Розглянемо, як працює маршрутизація. Існує одна точка входу – це index.php. Туди направлятимуться всі запити. Тобто url-адреса матиме

приблизно такий вигляд: http://назва_сайту/контролер/метод/параметри. Параметри будуть зазначатися в кінці, якщо ж вони передаються методом post, то в url-адресі вони не відображатимуться.

Файл .htaccess використовується внесення змін до стандартного налаштування веб-сервера Apache і вбудованого інтерпретатора PHP. Даний файл міститиме наступні рядки коду:

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]
```

Ця сукупність команд запустить процес перезапису з трьома умовами. Якщо це не фізична директорія або файл, або посилання, то перенаправить URL-адресу у файл index.php, за що і відповідає остання стрічка коду.

Ще один необхідний файл для роботи mvc – це Bootstrap. Даний файл містить php-команди для розділення url-адреси. Тобто виділить, який контролер потрібно викликати, який метод і які при цьому є параметри.

У файлі index.php лише підключатимуться файл Bootstrap та головний Controller, Model та View.

2.5 Розробка класів та функцій для рівня Model

Як було описано у першому розділі, компонент «Модель» у шаблоні проектування MVC призначена для надання доступу до даних користувачеві, що зберігаються в базі даних. Даний компонент містить також правила управління даними, тобто містить реалізацію методів доступу до інформації.

Для управління таблицею в базі даних та визначення правил взаємодії з нею відповідає саме модель. Для полегшення реалізації і роботи з архітектурою MVC одна таблиця бази даних відповідає одній моделі.

Отже, для запису, вибірки та оброблення даних, що зберігаються, потрібна система управління базою даних. Для цього було використано MySQL. Сама ж база даних була реалізована за допомогою додатку phpMyAdmin. PhpMyAdmin – це веб-додаток, що реалізований на PHP. Він надає можливість повноцінної роботи з базою даних MySQL, в тому числі і віддалено через браузер. Даний додаток надає можливість більш зручної роботи з створенням структури бази даних та заповненням її інформацією.

В ході реалізації проекту було створено такі класи, як Model, main_Model, order_Model, firm_Model та register_Model. Кожний клас взаємодіє з реалізованою базою даних. Розглянемо кожний клас більш детально.

Клас Model є основним класом компоненту моделі. Він відповідає, перш за все, за під’єднання до бази даних. Це дозволяє уникнути постійного виконання однієї і тієї ж команди. Також цей клас викликає відповідний дочірній клас, що реалізує функції, необхідні для відповіді на запит користувача. Команди для встановлення з’єднання з базою даних описані в конструкторі класу, оскільки є необхідними для будь-якого запиту і повинні виконуватися обов’язково. Реалізується це з’єднання в класі Database, а клас Model лише створює об’єкт цього класу, таким чином встановлюючи з’єднання для виконання подальших запитів.

Опис команд з’єднання з базою даних, що описаний в класі Database представлено нижче:

```
mysql_connect('localhost', 'root', '');
mysql_select_db('web');
```

Як видно з вищеподаного коду, для з’єднання використовуються дві функції php, а саме функція під’єднання та функція вибору бази даних з якою потрібно працювати.

Перша функція містить три параметра – це сервер MySQL, ім’я користувача та пароль для доступу. В даному випадку пароль не було встановлено, тому цей параметр залишається пустим.

Друга функція вибору бази даних містить два параметри. Перший – це ім’я бази даних, котру обирає користувач і, другий – з’єднання з MySQL. Проте, його вказувати необов’язково, оскільки використовується останнє з’єднання, що було встановлено у випадку порожнього другого параметру даної функції. Для підключення відповідної моделі для відповіді на запит користувача в класі Model було реалізовано метод loadModel. До того ж, в ньому передбачено можливість уникнення помилки роботи програми в разі відсутності такої моделі. Функція виклику представлено нижче:

```
require 'models/'.$name.'_model.php';
```

Як видно, для цього використовується функція php – require.

Для одержання інформації, що розміщена на головній сторінці веб-додатку було реалізовано клас main_Model. В даному класі було реалізовано метод news для одержання даних із бази даних та їх обробки. Для цього було реалізовано запит на мові SQL.

*$query='SELECT * FROM article ORDER BY id DESC LIMIT 3';*

Оскільки блоків з новинами на головній сторінці 3, то і одержати потрібно було дані лише з трьох записів. Для цього використовувався параметр limit в запиті на вибірку. Щоб користувач міг бачити найсвіжіші новини, то і вибиратися у запиті мали останні записи в таблиці. Для цього було використано параметр ORDER BY з приставкою DESC, що дозволяло відсортувати дані від останнього доданого запису до першого, використовуючи при цьому поле id.

Проте для роботи з цими даними було використано властивість об'єкта класу. Ця змінна є багатовимірним асоціативним масивом, що полегшує роботу з змінними, оскільки запам'ятати назву індексу відповідно до назви поля легше, ніж цифру. Для запису одержаних даних в цей масив було використано функцію mysql_fetch_array, що відповідає за створення масиву, та вказаний параметр MYSQL_ASSOC. Він дозволяє вказати, що це має бути саме асоціативний масив.

mysql_fetch_array($res,MYSQL_ASSOC)

Як видно в даному коді, вказується ще один параметр, перший по списку. Він відповідає за запуск самого запиту до бази даних.

$res=mysql_query($query,$this->db);

В цьому синтаксисі використовується функція mysql_query, що використовується, як було написано раніше, для запуску запиту на виконання. Перший параметр тут вказується сам запит, що описаний вище, а другий – це параметр, що відповідає за з'єднання з базою даних. Для реалізації можливості перегляду написаних статей з реалізацією посторінкової навігації було розроблено клас article_Model. Даний клас містить два методи.

Перший метод спрямований на одержання даних з таблиці бази даних для виведення самої статті та інших її атрибутів. Для цього використовуються

вищеописані функції mysql_query та mysql_fetch_array. Проте, тут вже виникає одна особливість. Потрібно вивести не просто останні три записи, а вже реалізувати навігацію по сторінках. Для цього було реалізовано клас модуля «Контролер», який буде описаний нижче, проте для вибору необхідної сторінки потрібно одержати параметр «сторінка».

Цей параметр викликається шляхом звернення до конструктора класу компоненту «Контролер». Для звернення до нього використовується пара символу «:». Код представлено нижче.

```
article::page();
$id = article::$this->id;
```

Як бачимо, спершу відбувається ініціалізація функції page, а тоді звернення до змінної, яка була записана в ході виконання цієї функції.

Далі це значення передається в запит до бази даних, в результаті він виглядає так:

```
$query="SELECT * FROM article WHERE id='".$id."'";
```

Як бачимо, за допомогою інструкції where, було задано умову, яка саме стаття необхідна користувачеві. Таким чином, було реалізовано можливість посторінкової навігації в компоненті «Модель». Результат виконання частини модуля «Контролер» буде описано нижче. Код розроблених модулів приведений в додатку А.

Наступний метод, який було реалізовано в даному класі – це метод total. Він дозволяє визначити кількість записів в таблиці «order». Це необхідне для посторінкової навігації, адже користувач не може обрати «-1» сторінку чи номер сторінки, в якій не має жодних записів. В такому випадку помилки не виникне, проте інформації користувач не отримає, а натомість може зайти на велику кількість пустих сторінок вперед. Для уникнення цього і була

реалізована дана функція. Для звернення до бази даних використовуються ті ж функції, проте синтаксис запиту вже дещо інший.

$query='SELECT COUNT(*) FROM order;

Як видно з щойно поданого коду, використовується деяка команда count. Вона дозволяє підрахувати кількість записів у таблиці за полем id.

Для можливості перегляду даних про фірми, що розміщені в базі даних було реалізовано схожий клас firm_Model. Хоч і реалізація схожа з реалізацією класу order_Model, проте даний клас дозволяє працювати вже з таблицею firm. Для реалізації запитів до бази даних використовуються ті ж функції, проте синтаксис звернення змінюється в плані звернення до іншої таблиці та використанню вже її ж полів.

$query="SELECT * FROM firm WHERE id='".$id."'";

Отже, як бачимо для отримання даних про потрібну користувачеві фірму, було реалізовано з використання тієї ж команди select та інструкцію where, проте звернення вже відбувається до іншої таблиці, в даному випадку це таблиця firm.

Для уникнення тієї ж незручності, що і для перегляду статей, було аналогічно реалізовано функцію для підрахунку кількості записів.

Для того, щоб користувач зміг зареєструватися для подальшого використання ширшого спектру можливостей сайту було реалізовано модель register_Model. Даний клас з аналогічною назвою здійснює запис в базу даних. Для цього йому необхідні дані користувачі, які він збереже у базі. Ці дані він отримує від контролера шляхом звернення до відповідної функції потрібного класу. Одержання цих даних реалізовано шляхом виклику конструктора контролера та звернення до відповідних змінних цього класу.

register::type();

$name=register::$this->name;

Схожа конструкція, лише з викликом іншого класу та іншої функції, була описана вище, тому для додаткового розуміння використання даних звернень можна вернутися дещо назад.

Для уникнення такої помилки, як запис пустих значень, потрібно перевіряти, а чи справді дані є чи описані змінні є пустими. Для цього було використано оператор if та функції empty. Оператор дозволяє задати умову, що програма повинна виконати, якщо значення пусте, то запису даних в базу даних не відбувається, якщо ж значення змінної присутнє, то відбувається формування і надсилання запиту в базу даних. Для перевірки існування значення змінної і було використано функцію empty. Дана конструкція виглядає так:

if (!empty ($name))

Для того, щоб додати запис в базу формується запит, представлений нижче:

$query="INSERT INTO users (name, date, login, password, id_access) VALUES ('".$name."','".$birthd."','".$login."','".$password."','".$access."')";

Даний запит використовує команду insert into, призначену для здійснення запису в таблицю бази даних. Далі було зазначено в яку саме таблицю здійснюється запис та перераховані поля, в котрі значення мають бути записані. Тоді використовується оператор VALUES призначений для встановлення значень у відповідні поля. Дані значенні повинні бути записані в такому ж порядку, як і поля для яких вони призначенні. Алгоритми функцій, що дозволяють виконувати бізнес логіку наведені в додатку В.

В функції mysql_query, призначенні для передання запиту у базу, в даному випадку вказується лише один параметр, оскільки запит не повертає вибраних даних з таблиці і змінної для роботи з ними не потрібно.

В результаті цієї реалізації було описано чотири класи для роботи з базою даних.

Розробка класів та функцій для рівня View

Для зручної взаємодії користувача з даними одержаними із бази даних було реалізовано основний клас View. Даний клас призначений для підключення необхідних виглядів, що необхідні контролеру. Для цього в класі View було реалізовано функцію render. Дана функція підключає три частини вигляду сторінок. Це header, content та footer.

Спершу розглянемо header та footer. Дані частини представлення є фіксованими для всіх сторінок. Header містить стрічку навігації по сайті та поле для пошуку даних. Для цього було реалізовано стиль header засобами css. Цей стиль вказував на те, що даний заголовок повинен бути фіксованим зверху будь-якої сторінки. Опис ж самого вигляду навігації було реалізовано мовою розмітки сторінки html. Для задання стилів було використано Bootstrap із вже сформованими описаними візуальними компонентами. Для реалізації представлення сторінки було підключено даний файл.

<script src="public/js/bootstrap.js"></script>

<link href="public/css/bootstrap.css" rel="stylesheet">

Для формування заголовку сторінки та реалізації навігації було використано клас navbar navbar-default та navbar-collapse collapse із підключеного файлу.

Інша фіксована частина сторінки – це footer. Низ сторінки складаєтьсяз двох блоків для реклами. Для їх формування було використано класи panel panel-warning та panel-body.

Частину реалізації content розглянемо більш детально. Ця частина є змінною і для окремих сторінок має різний вигляд. Тому для реалізації зміни

інтерфейсу сторінки було реалізовано декілька контентів.

Перший контент – це контент головної сторінки. Дещо схожий з реалізацією блоків як для нижньої частини вікна, але має дещо змінений вигляд та призначений для відображення найновіших статей доданих користувачами.

Також створено вікно авторизації, яке в подальшому дозволить реалізувати та використовувати можливості аутентифікації користувачів для розширення можливостей використання сайту.

Наступний контент призначений для відображення даних, а саме для представлення статей. Окрім блоку, який передбачений для тексту, є додаткова навігація по сторінках для перегляду усіх статей, що записані у базі даних. Для цього використовуються кнопки-посилання, код опису яких представлено нижче.

* *

Дана конструкція дозволяє передати номер сторінки на котру потрібно перейти. Аналогічний вигляд має вікно для перегляду даних про фірму, відрізняється лише заголовком контенту.

Для передачі даних про користувача для реєстрації було створено контент register. Дана сторінка містить поля для вводу 5 параметрі, які користувач передає в контролер для подальшої їх обробки. Для цього було використано форму із зазначенням куди повинні відправлятися і дані і який метод для цього використовується. Код опису цієї форми наведений нижче.

<form class="navbar-form" action="register" method="post">

Як бачимо, використовується клас navbar-form для створення зовнішнього вигляду полів для введення даних. В параметрі action задано контролер, який повинен обробляти ці дані, а сааме контролер register. Метод для передачі неповинен відображати в адресній стрічці, які саме дані

передаються, тому для цього використовується метод post.

Для надсилання форми використовується кнопка «Зареєструватися».

<button type="submit" class="btn btn-danger">Зареєструватися</button>

Тип кнопки submit означає, що кнопка натискаєть і відправляє дані отримані з форми. Клас btn btn-danger лише описує її зовнішні вигляд. Таким чином, сформувавши header та footer, не потрібно зайвий раз описувати заголовок і нижню частину сторінки, що зменшує об'єми роботи та дозволяє підключати в контролері лише контент зменшуючи трафік передачі даних.

Розробка класів та функцій для рівня Controller

Для взаємодії моделі та представлення було реалізовано клас Controller. Він призначений для виклику основних класів View та Model. Для чого вони призначені було описано в питаннях 3.1 та 3.2 третього розділу.

Для виклику цих класів в контролері використовують ініціалізацію класу через змінну.

$this->view = new View();
$this->model = new Model();

Далі було реалізовано дочірні класи класу Controller. При виклику кожного класу запускається відповідний контролер-обробник. Розглянемо їх більш детально.

Клас контролера main призначений для формування головної сторінки веб-додатку. Цей клас створює об'єкт класу main_model для отримання даних з бази даних про найновіші статті. Також викликається функції для реалізації цього запиту. Для цього використовуються код:

$model = new main_model();

$model->news();

Проте для реалізації такої можливості потрібно підключити файл в якому описано клас article_model. Для цього використовується функція require, що описана в основному класі Model.

Отримавши доступ до даних через змінні об’єкту класу, методом post передаємо їх у представлення. Для цього використовується такий код:

$_POST['name0']=$model->row['0']['name'];

Даний метод при передачі даних приховує, не відображає в адресній стрічці. Це необхідно, оскільки стаття велика за розміром і запис її в url-адресу не є доцільним. Параметри зазначенні у квадратних дужках вказує на індекс масиву значення з якого повинне бути використане. Якщо масив двовимірний, то вказується два індекса по рядках і стовпцях, кожний параметр в окремих дужках, так як для змінної row.

Для підключення інтерфейсу сторінки, контролер використовує команду звернення до об’єкта класу View. Створювати його об’єкт через змінну вже не потрібно, бо його ініціалізація відбувається в батьківському класі. Таким чином це не потребує повторного використання функцій і не виникає надлишковості коду та виконання зайвих команд.

Для використання представлення сторінки використовується наступний код:

$this->view->render('main/index');

де функція render – функція класу View, а в дужках зазначається який саме файл з описом сторінки веб-додатку повинен підключатися.

Для реалізації можливості перегляду статей було створено клас article. Даний клас містить два методи. Перший метод page призначений для отримання номеру сторінки, що переданий користувачем. Для цього змінній

класу присвоюється значення отримане методом get.

Другий метод виконується одразу при створенні об’єкту класу, оскільки зазначений в його конструкторі. Для цього створюється об’єкт класу article_Model, та присвоєння значень глобальному масиву post.

```
$_POST['name']=$art->row['name'];
$_POST['article']=$art->row['article'];
```

Як видно з коду, використовується два параметри – це назва статті і сама стаття. Ці дані передаються у компонент представлення, який викликається функцією render классу View.

Аналогічно реалізований контролер для роботи з даними про фірму. Тут використовуються ті ж методи, що і для роботи із розділом для статей.

Клас register було розроблено для реалізації можливості реєстрації користувачів. В конструкторі даного класу відбувається підключення файлу з описом класу для роботи з базою даних. Для цього використовується команда:

```
require 'models/register_model.php';
```

Також підключається відповідне представлення, реалізація виклику функції якої описана вище.

В класі register реалізовано також ще два методи type та proof.

Метод type призначений для передачі даних моделі від користувача для запису його у базу даних. Для того, щоб дані, що передаються були приховані, було використано метод post.

```
$this->name=$_GET['name'];
```

Другий метод це метод proof, що використовується для підтвердження успішної реєстрації користувача. Він викликає вже інший контент, який не містить форми для введення даних, а лише видає інформацію про те, що

користувач зареєстрований.

Висновки до розділу 2

1. Проведено дослідження інструментарію розробки, що дало можливість переконатися в ефективності вибору.

2. Виконано проектування ієрархії класів для рівня Model, View, Controller, що надає можливість реалізації зпроектованої архітектури.

3. Виконано розробку класів та функцій для рівня Model, View, Controller, які забезпечують функціональність та зовнішній вигляд системи.

3 ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОЇ ПОБУДОВИ КОНСТРУКЦІЙ ПРИДВЕРНИХ РЕШІТОК

3.1 Математичне забезпечення для автоматизованого проектування конструкцій придверних решіток

При проектуванні були використанні методи комп'ютерної графіки до яких відносять методи перетворення графічних об'єктів, виведення (розгорнення) ліній у растровій формі, виділення вікна, видалення схованих ліній, проектування, зафарбування зображень.

Перетворення графічних об'єктів виконується за допомогою операцій переносу, масштабування, повороту.

Перенос точки з положення P в нове положення і можна виконувати по формулах типу:

$$C_{xi}=P_{xi}+\Delta_{xi} \tag{3.1}$$

де Δ_{xi} — приріст по координаті xi. Однак зручніше операції перетворення представляти в єдиній матричній формі:

$$C=PT \tag{3.2}$$

де T— перетворююча матриця. При цьому крапки C й P у двовимірному випадку зображують векторами-рядками 1×3, у яких крім значень двох координат, що мають назву при такому поданні однорідними, додатково вказують масштабний множник W. Тоді перенос для випадку 2D можна виразити у вигляді (3.2), де T представлене в таблиці 3.1, а W =1.

Таблиця 3.1

Представлення точок

1	0	0
0	1	0
Δ_x	Δ_x	1

Для операцій масштабування й повороту матриці T представлені в таблиці 3.2 і таблиці 3.3 відповідно, де m_x, m_y — масштабні множники, φ — кут повороту.

Зручність (3.2) пояснюється тим, що будь-яку комбінацію елементарних перетворень можна описати формулою (3.2).

Наприклад, вираження для переміщення з одночасним поворотом має вигляд:

$$C=PT_{сд}\,T_{пов}=PT,$$

де $T=T_{сд}\,T_{пов}$, $T_{сд}$ — матриця переміщення, $T_{пов}$ — матриця повороту.

Таблиця 3.2

Матриця масштабування

m_x	0	0
0	m_y	0
0	0	1

Таблиця 3.3

Матриця повороту

$\cos\varphi$	$\sin\varphi$	0
$-\sin\varphi$	$\cos\varphi$	0

0	0	1

Вивід графічних елементів у растровій формі потрібно для відображення цих елементів на бітову карту растрової системи. Нехай потрібно розгорнути відрізок *AB* прямої $y=ax+b$, причому $1 \geq a \geq 0$ (при інших значеннях *a* розглянутий нижче алгоритм залишається справедливим після певних модифікацій). Введемо позначення: $A=(xa,ya)$, $B=(xb,yb)$ за величину дискрета (пікселя) приймемо одиницю. В алгоритмі розгорнення номера рядків і стовпців карти, на перетинанні яких повинні перебувати точки відрізка, визначаються в такий спосіб:

1. $\Delta x:=xb-xa;\Delta y:=yb-ya;x:=xa;y:=ya;$
2. $d:=2\Delta y-\Delta x;$
3. якщо $d \geq 0$, то $\{y:=y+1;d:=d+2(\Delta y-\Delta x)\}$ інакше $d:=d+2\Delta y;$
4. $x=x+1;$
5. перехід до пункту 3, поки не досягнута точка *B*.

Економічність цього алгоритму обумовлюється відсутністю надлишково довгих арифметичних операцій типу множення.

Виділення вікна потрібно при визначенні тієї частини сцени, що повинна бути виведена.

Нехай вікно обмежене лініями $x=x_1$, $x=x_2$, $y=y_1$, $y=y_2$ (рисунок 3.1). По черзі для кожного багатокутника перевіряється розташування його вершин і ребер щодо границь вікна. Так, для багатокутника *ABCD* (рисунок 3.1) при відсіканні по границі $x=x_2$ проглядається множина вершин у порядку обходу по годинній стрілці. Можливі чотири ситуації для двох послідовних вершин P і R:

1. якщо $x_P > x_2$ й $x_R > x_2$, то обидві вершини й інцидентне їм ребро перебувають поза вікном і виключаються з подальшого аналізу;

2. якщо $x_P \leq x_2$ й $x_R \leq x_2$, то обидві вершини й інцидентне їм ребро залишаються для подальшого аналізу;

3. якщо $x_P \leq x_2$ й $x_R > x_2$, то вершина P залишається в списку вершин, а вершина R_3 міняється новою вершиною з координатами $x=x_2, y=y_P+(y_R-y_P)(x_2-x_P)/(x_R-x_P)$ в нашому прикладі такою новою вершиною буде E;

4. якщо $x_P > x_2$ й $x_R \leq x_2$, то вершина P заміняється новою вершиною з координатами $x=x_2, y=y_R+(y_P-y_R)(x_2-x_R)/(x_P-x_R)$, а вершина R залишається в списку вершин; у нашому прикладі новою вершиною буде F.

Після закінчення перегляду стосовно до всіх границь у вікні виявляються вершини, що залишилися в списку.

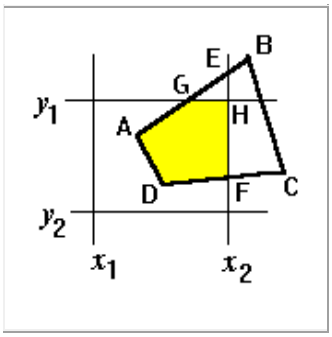


Рисунок 3.1 - Виділення вікна

Застосовуючи ці правила в нашому прикладі, одержуємо спочатку багатокутник *AEFD*, а після відсікання по верхній границі, $y=y_2$ — багатокутник *AGFD* (рисунок 3.1). Однак правильний результат трохи інший, а саме багатокутник *AGHFD* . Цей правильний результат виходить при подвійному обході вершин спочатку по годинній стрілці, потім проти із включенням у список нових вершин, що з'являються при кожному обході.

Застосовуються ряд алгоритмів видалення схованих ліній. Один з найбільш просто реалізованих алгоритмів — алгоритм *z*-буфера, де *z*-буфер — область пам'яті, число осередків у якій дорівнює числу пікселів у вікні виводу. Передбачається, що вісь *z* спрямована по нормалі до видової поверхні й спостерігач розташований у крапці $z=0$.

На початку виконання алгоритму всі пікселі відповідають максимальному значенню *z*, тобто максимальному видаленню від спостерігача, що приводить до приміщення в усі осередки *z*-буфера значень пікселів тіла креслення. Далі по

черзі для всіх точок граней розраховуються значення координати *z*. Серед точок, що відносяться до того самого пікселя (одному й тому ж осередку *z* - буфера *S*), вибирається точка з найменшим значенням z і її код (тобто кольори і яскравість) міститься в *S*. У підсумку *z*-буфер буде містити пікселі найбільш близьких до спостерігача граней.

Частково використовувались алгоритми побудови проекцій перетворюють тривимірні зображення у двовимірні. У випадку побудови центральної проекції кожна точка тривимірного зображення відображається на картинну поверхню шляхом перерахування координат *x* і *y* (рисунок. 3.2). Так, координата x'_a. точки A' обчислюється по наступній формулі: $x'_a = \frac{x_a d}{Z}$, аналогічно розраховується координата y'_a. точки A'.

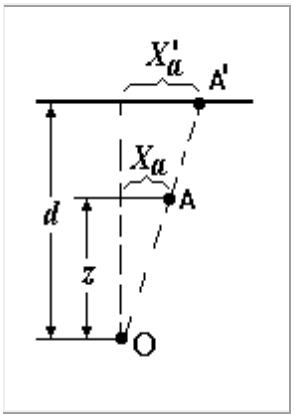


Рисунок 3.2 - Побудова центральної проекції точки *A*

У паралельних проекціях d→∞ і координати *x* й *y* точок A' і A збігаються. Тому побудова паралельних проекцій зводиться до виділення вікна, при необхідності до повороту зображення й можливо до видалення схованих ліній.

3.2 Проектування основних таблиць бази даних

Для створення бази даних використовується наступний DDL код:

```
CREATE DATABASE IF NOT EXISTS `order` DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
USE `order`;
```

Таблиця «order» використовується для збереження інформації про замовника, характеристики виробу, матеріали та особливості конструкції.

Для створення таблиці «order» використовується наступна процедура:

```
CREATE TABLE IF NOT EXISTS `order` (
  `id` int(10) unsigned NOT NULL,
  `oid` varchar(20) default '',
  `client_id` int(10) unsigned default NULL,
  `user_id` int(10) unsigned default NULL,
  `order_date` datetime default NULL,
  `worker_order_number` int(11) default NULL,
  `client_order_number` int(11) default NULL,
  `order_additional_code` varchar(50) default NULL,
  `product_id` int(10) unsigned default NULL,
  `product_name` varchar(100) default '',
  `product_type` tinyint(1) unsigned default '0',
  `profile_length` float(10,2) unsigned default '0.00',
  `product_length` float(10,2) unsigned default '0.00',
  `inside_radius` int(10) default '0',
  `outside_radius` int(10) default '0',
  `border` tinyint(1) unsigned default '0',
  `slats_amount` tinyint(1) unsigned default '0',
  `border_product_id` int(10) default NULL,
  `area` float(10,3) unsigned default '0.000',
  `amount` float(10,3) unsigned default '0.000',
```

```
`total_area` float(10,3) unsigned default '0.000',
`total_weight` float(10,3) unsigned default '0.000',
`border_length` float(10,3) unsigned default '0.000',
`date_ship_to` datetime default NULL,
`brigade` tinyint(1) default NULL,
`date_shipped` datetime default NULL,
`shipping_method` int(10) unsigned default NULL,
`shipping_address` varchar(255) default NULL,
`logo_variant` int(10) default NULL,
`is_active` tinyint(1) unsigned default '0',
`color_id` int(10) unsigned default '0',
`status_id` int(2) unsigned default '1',
`order_decline_reason` int(5) unsigned default '0',
`file_name` varchar(255) default NULL,
`status_updated` datetime default NULL,
`object_type` int(10) default NULL,
`sizes_source` int(10) unsigned default NULL,
`cargo_consignee` varchar(255) default NULL,
`courier_shipping_method` int(10) unsigned default NULL,
`courier_shipping_address` varchar(255) default NULL,
`color_version` int(10) unsigned default NULL,
`storage` int(10) unsigned default '0',
`laboriousness_coef` float unsigned default '1',
`scheme` longtext,
`cargo_payer` tinyint(1) unsigned default '0',
`contact_person_cargo` varchar(255) default NULL,
`send_sms_info` tinyint(1) NOT NULL,
`sms_phone` varchar(15) NOT NULL,
`piece_length` float(10,3) unsigned NOT NULL,
`piece_count` int(3) unsigned default '0',
```

```
`border_changed` tinyint(1) default '0',
`sketch_uploaded` tinyint(1) default '0',
`has_suborders` tinyint(1) default '0',
`parent_id` int(10) default NULL,
`cut_slats` tinyint(1) default '0',
`max_size` int(10) NOT NULL default '0',
`bright_pad` tinyint(1) NOT NULL default '0',
`bright_pad_stub` tinyint(1) NOT NULL default '0',
PRIMARY KEY  (`id`),
KEY `product_id` (`product_id`),
KEY `FK_order2` (`user_id`),
KEY `FK_order5` (`status_id`),
KEY `client_id` (`client_id`),
KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
ROW_FORMAT=DYNAMIC;
```

Проектування бази даних відбувалось в середовищі MySql WorkBench, як проілюстровано на рисунку 3.3.

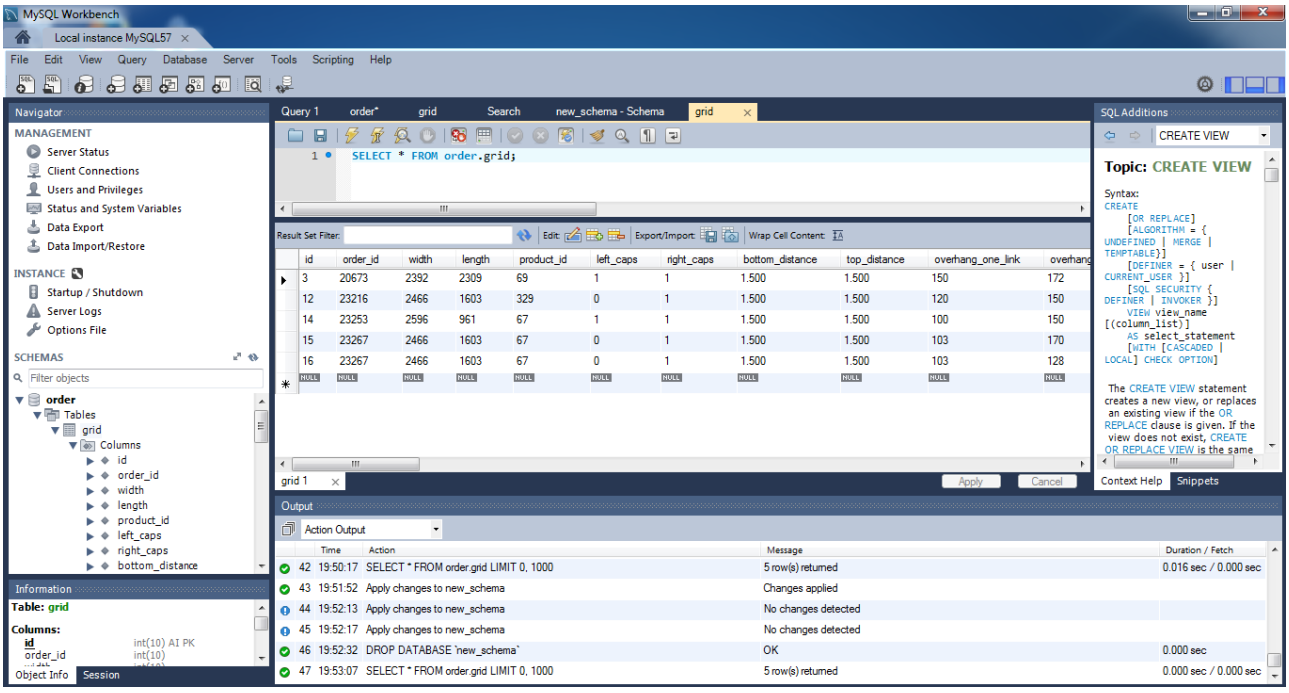


Рисунок 3.3 – Створена база даних відображена з використанням MySql Workbanch

Таблиця «grid» використовується для збереження параметрів замовленої решітки. Додаткові параметри вказані при проектуванні встановлені замовником також відносяться до таблиці «grid».

При створенні таблиці «grid» використаний наступний SQL код:

```
CREATE TABLE IF NOT EXISTS `grid` (
  `id` int(10) NOT NULL auto_increment,
  `order_id` int(10) NOT NULL,
  `width` int(10) NOT NULL,
  `length` int(10) NOT NULL,
  `product_id` int(10) NOT NULL,
  `left_caps` tinyint(1) default NULL,
  `right_caps` tinyint(1) default NULL,
  `bottom_distance` float(10,3) NOT NULL,
  `top_distance` float(10,3) NOT NULL,
  `overhang_one_link` int(10) NOT NULL,
  `overhang_two_links` int(10) NOT NULL,
  `filename` varchar(255) default NULL,
  `grid_group_id` int(10) default NULL,
  `image_cuts` longtext,
  `cuts_points` text,
  `pixel_width` int(10) default NULL,
  `pixel_length` int(10) default NULL,
  `area` float(10,2) default '0.00',
  `diagonal_cuts` int(3) default '0',
  `arc_cuts` int(3) default '0',
  `logo_id` int(10) default NULL,
  `links_distance` int(10) NOT NULL,
```

```
  `slats` tinyint(1) default NULL,
  `slats_position` tinyint(1) default NULL,
  `additional_limit` int(3) default '0',
  `left_limit` tinyint(1) default '0',
  `right_limit` tinyint(1) default '0',
  `use_for_main_links` tinyint(1) default '0',
  `small_caps_angles` tinyint(1) default '0',
  `custom_gap` tinyint(1) default '0',
  `gap` float(4,2) default '0.00',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=712
;
```

Для розрахунку самої грід конструкції в таблиці «grid_formula» внесена бізнес-логіка, що може змінюватись в залежності від використання розробленої системи. Для створення таблиці «grid_formula» використано наступний код:

```
CREATE TABLE IF NOT EXISTS `grid_formula` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `title` varchar(255) default NULL,
  `key` varchar(255) default NULL,
  `expression` text,
  `is_system` tinyint(1) unsigned default '0',
  `created` datetime default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=347;
```

Для групування шаблонів типів виробів використовується таблиця «grid_groups».

При створенні таблиці «grid_groups» використана наступна процедура:

```
CREATE TABLE IF NOT EXISTS `grid_groups` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `title` varchar(255) default NULL,
```

```
  `created` datetime default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=4 ;
```

На етапі рендерінгу шаблону дані про виріб у вигляді HTML коду містять змінні, які пізніше змінюються відносно конкретного замовлення.

При створенні таблиці «grid_group_templates» використана наступна процедура:

```
CREATE TABLE IF NOT EXISTS `grid_group_templates` (
  `grid_group_id` int(10) unsigned NOT NULL,
  `grid_template_id` int(10) unsigned NOT NULL,
  `position` int(3) NOT NULL,
  PRIMARY KEY  (`grid_group_id`,`grid_template_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Для збереження сю реалізованих HTML шаблонів з вставками формул згідно, яких будуть проводитись обчислення використано таблицю «grid_template».

При розробці бази даних було створено таблиці та встановлено зв’язки між ними, які детально описує ER діаграма бази даних, що відображена на рисунку 3.4.

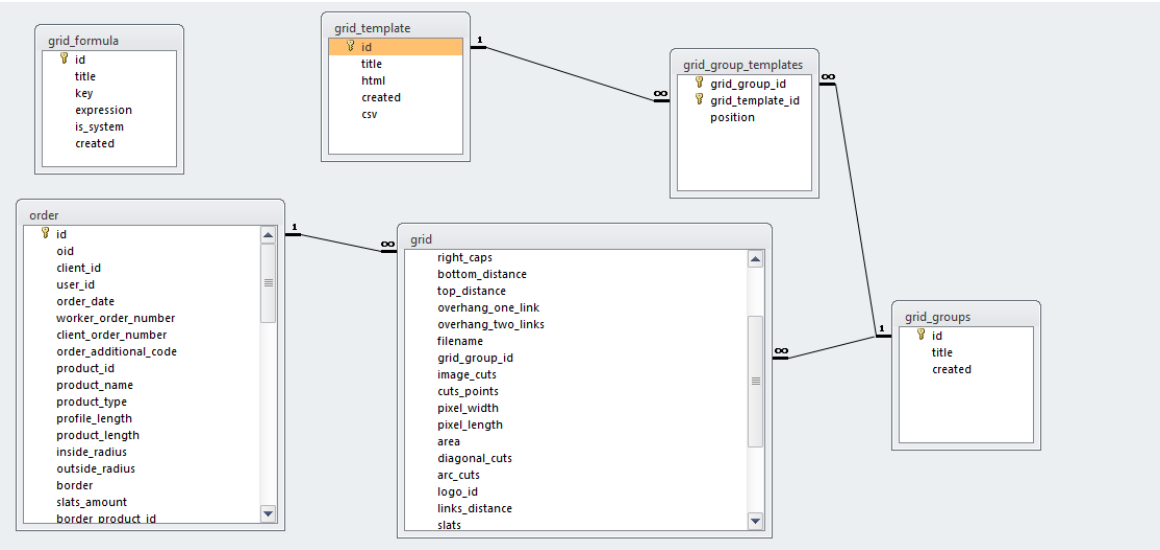


Рисунок 3.4 - ЕР діаграма проектованої бази даних

Основна частина спроектованої бази даних представлена на рисунку 3.2,

при побудові діаграми було використано лише основні таблиці, що забезпечують функціонування бізнес логіки.

При створенні таблиці «grid_template» використана наступна процедура:

```
CREATE TABLE IF NOT EXISTS `grid_template` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `title` varchar(255) default NULL,
  `html` longtext,
  `created` datetime default NULL,
  `csv` tinyint(1) unsigned default '0',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 AUTO_INCREMENT=13 ;
```

3.3 Тестування та відлагодження системи автоматизованого проектування конструкцій придверних решіток

При першому використанні системи автоматизованого проектування конструкцій придверних решіток, адміністраторам системи потрібно створити групи шаблонів, як це показано на рисунку 3.5.

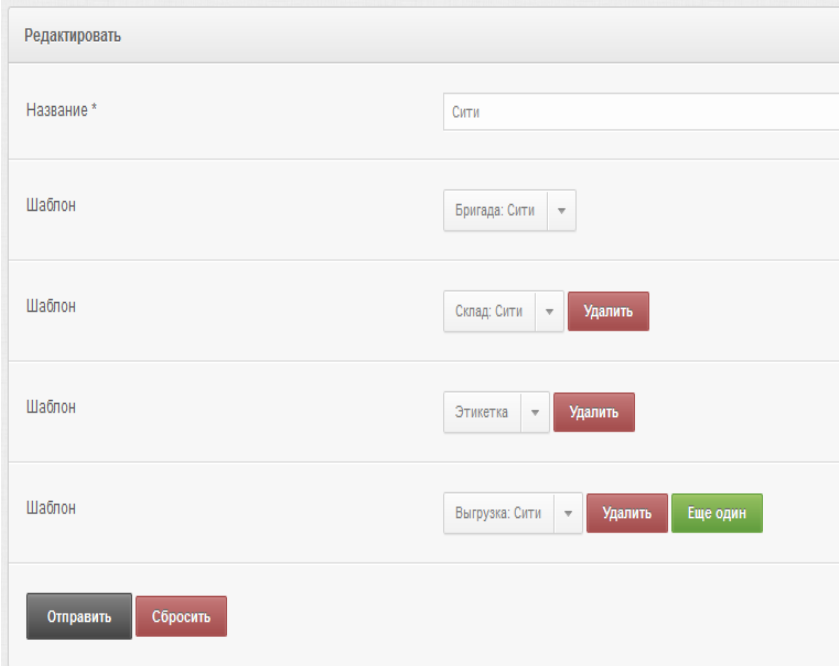


Рисунок 3.5 - Редагування групи шаблонів - додавання/видалення шаблонів з групи

Меню груп шаблонів дозволяє групувати шаблони в "комплекти" залежно від номенклатури. В проекті решітки зберігається ідентифікатор групи і на етапі розрахунків стають доступні відповідні шаблони, як це показано на рисунку 3.6.

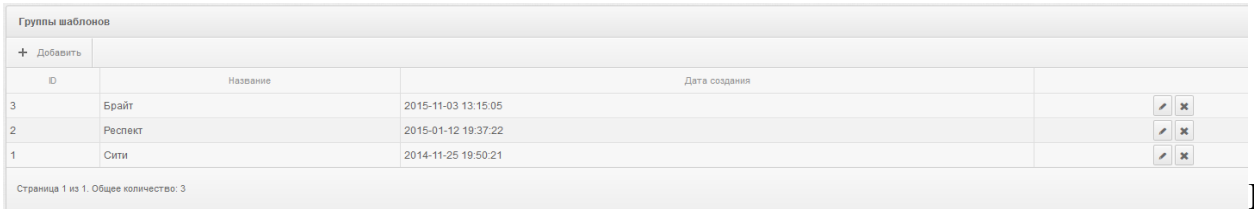


Рисунок 3.5 – Меню груп шаблонів

Саме ж меню шаблонів, що дозволяє виконувати функції створення, редагування та видалення, представлене на рисунку 3.6.

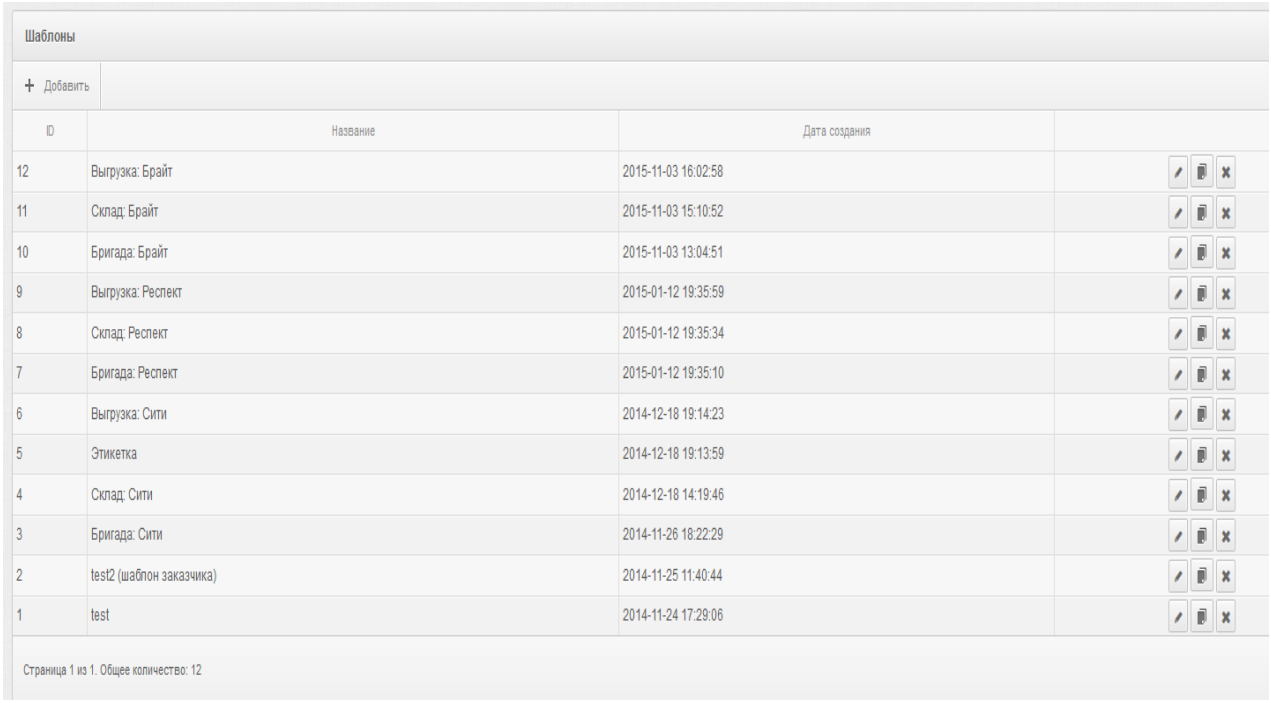


Рисунок 3.6 - Меню шаблонів

Для кожного замовника можливий вибір шаблону виробу та специфікації, які обираються перед створенням самого виробу та точного технічного опису конструкції придверної решітки.

Після того, як користувач натисне на відповідне посилання, система запропонує провести редагування шаблону, як це показано на рисунку 3.7.

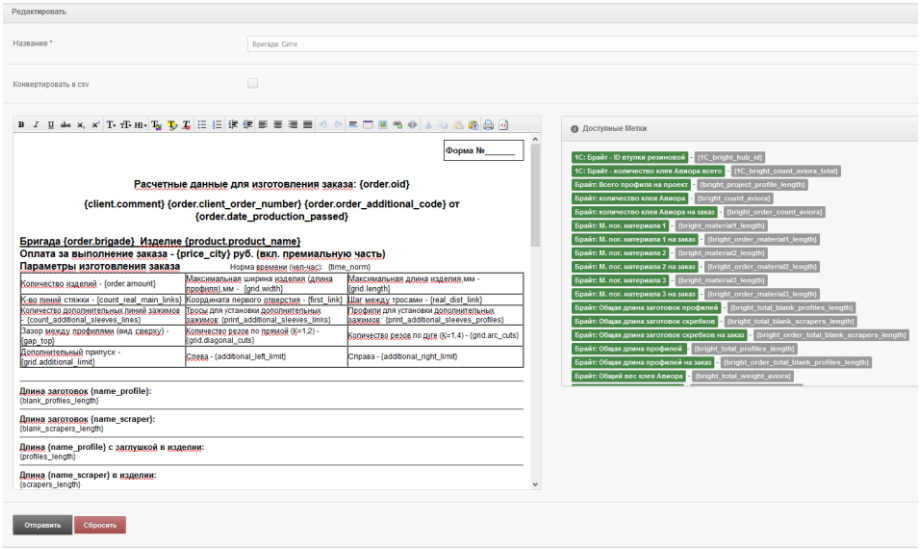


Рисунок 3.7 Редагування шаблону

На рисунку 3.7. представлено внесені тестові дані, реального шаблону, що використовується при проектуванні виробів (придверних решіток), що підтверджено довідкою про впровадження представленою в додатку Б.

В системі доступне редагування формул згідно, яких обчислюються специфікації виробу та розхідні матеріали. Здіснити ці операції можна з використанням меню перегляду та редагування формул (рисунок 3.8).

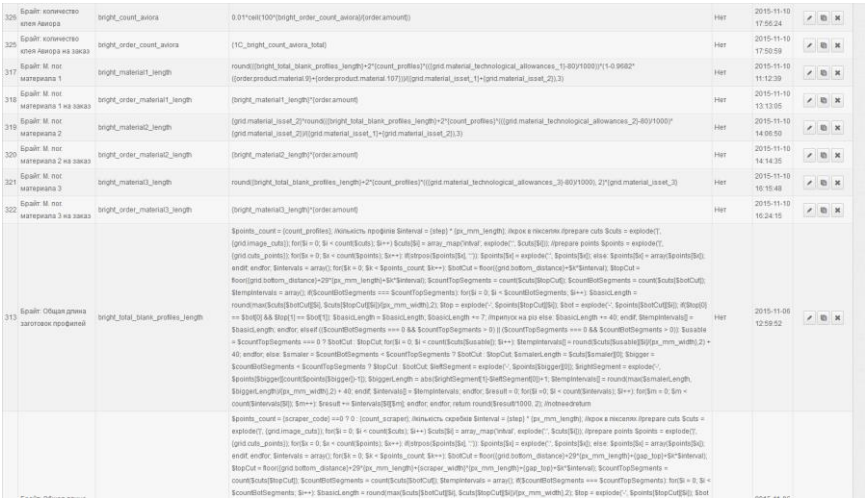


Рисунок 3.8 – Меню формул

Специфічною особливістю розробленої системи є надання адміністратору можливості внесення власноруч математичних формул та специфікацій, що дозволяє уникнути постійної модифікації коду під конкретні фірми та специфіки виробу (рисунок 3.9).

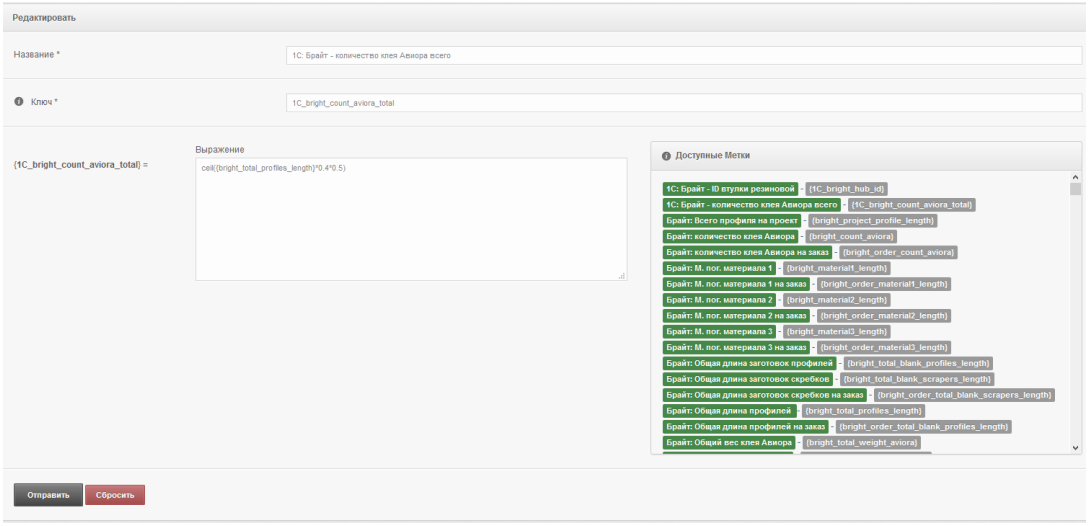


Рисунок 3.9 – Редагування формул для обчислення специфікацій виробу

При створенні проекту конструкції придверної решітки замовнику необхідно буде внести деталі замовлення в форму, що на рисунку 3.10.

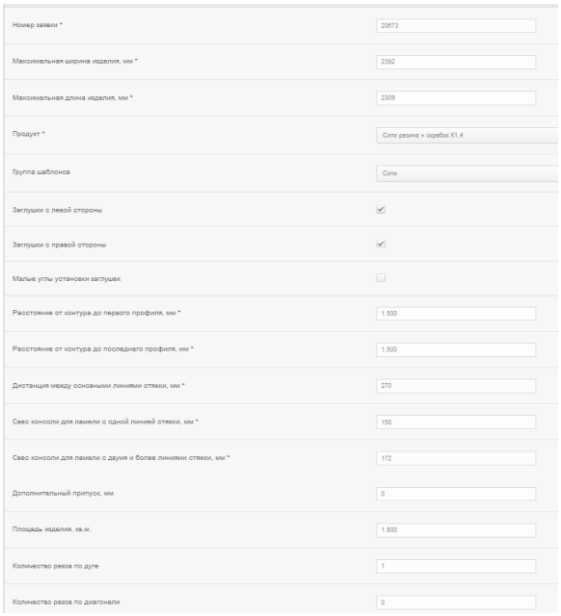


Рисунок 3.10 – Форма створення, редагування проекту решітки

Після внесення даних про проектований виріб користувачеві буде відображено інформацію про його замовлення з поточним статусом процесу проектування та виготовлення (рисунок 3.11).

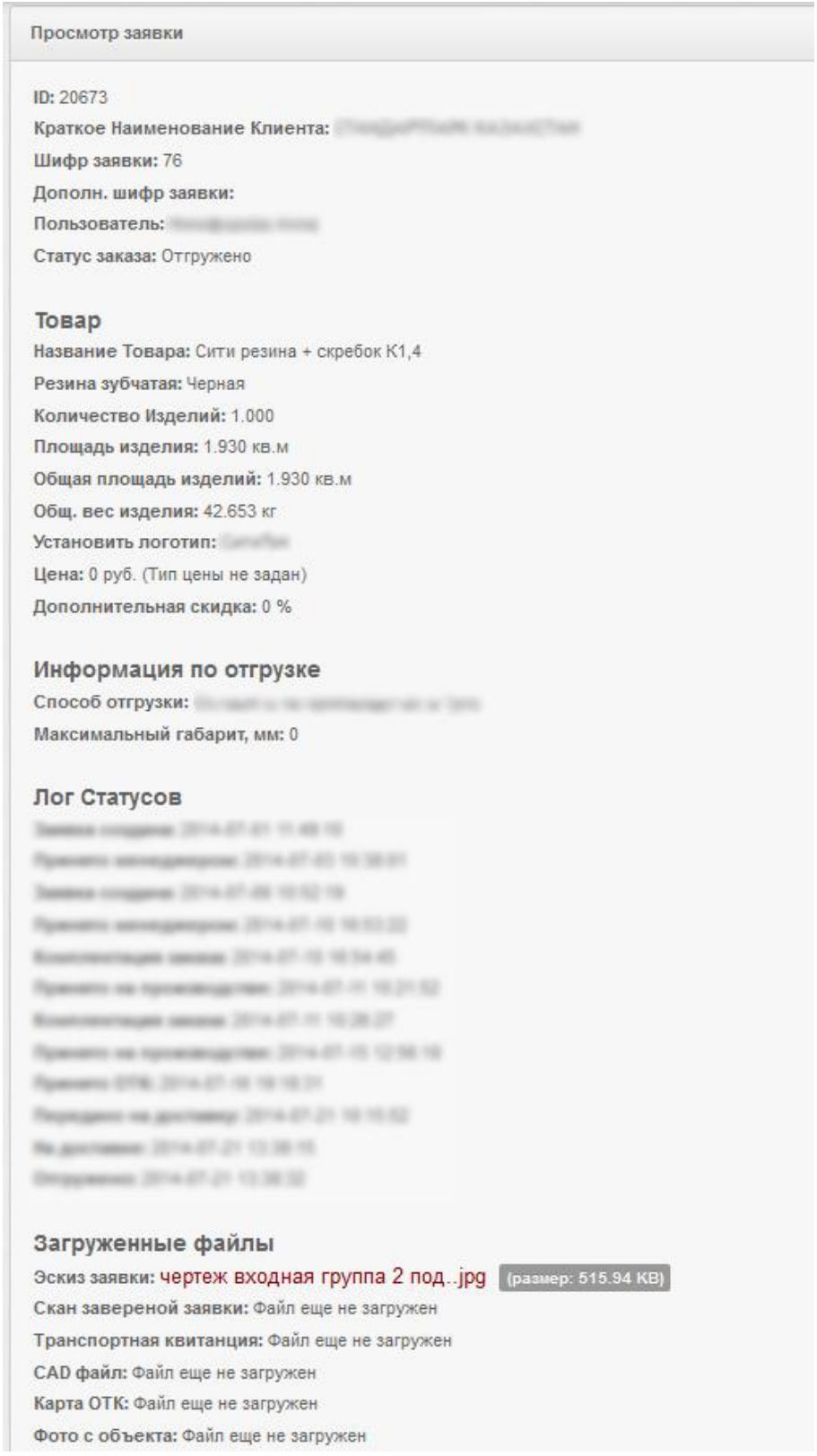


Рисунок 3.11 - Інформація про замовлення

Після завершення опрацювання заявки, система автоматично сформує відомість для списання на складі розхідних матеріалів, яка буде містити специфікацію елементів та самого виробу (рисунок 3.12).

Ведомость материалов для выдачи и списания на заказ:

[redacted] **Бригада 3**

Общие данные по заявке

Общая площадь изделий по заявке, м кв.	1.930	Общий вес изделия по заявке (кг.)	42.653
Наименование изделия	Сити резина + скребок К1,4	Кол-во изделий (шт.)	1.000
Количество резов по прямой (К=1,2)	0	Количество резов по дуге (К=1,4)	1
Размеры изделия (ширина х длина)	2392 x 2309	-	-

Альбомная ориенті

Портретная ориенті

Материалы к выдаче:
Длины профилей с припуском:
1 - 2386 ‖ 2 - 2386 ‖ 3 - 2386 ‖ 4 - 2432 ‖ 5 - 2127.12 ‖ 6 - 1892.22 ‖ 7 - 1731.29 ‖ 8 - 1605.34 ‖ 9 - 1497.39 ‖ 10 - 1404.43 ‖ 11 - 1320.46 ‖ 12 - 1245.49 ‖ 13 - 1175.52 ‖ 14 - 1110.55 ‖ 15 - 1051.58 ‖ 16 - 995.6 ‖ 17 - 944.62 ‖ 18 - 895.64 ‖ 19 - 849.66 ‖ 20 - 807.68 ‖ 21 - 767.69 ‖ 22 - 728.71 ‖ 23 - 692.73 ‖ 24 - 659.74 ‖ 25 - 626.75 ‖ 26 - 597.77 ‖ 27 - 568.78 ‖ 28 - 542.79 ‖ 29 - 517.8 ‖ 30 - 493.81 ‖ 31 - 471.82 ‖ 32 - 451.83 ‖ 33 - 432.84 ‖ 34 - 413.84 ‖ 35 - 397.85 ‖ 36 - 381.86 ‖ 37 - 367.86 ‖ 38 - 355.87 ‖ 39 - 343.87 ‖ 40 - 333.88 ‖ 41 - 323.88 ‖ 42 - 316.88 ‖ 43 - 309.89 ‖ 44 - 303.89 ‖ 45 - 298.89 ‖ 46 - 295.89 ‖ 47 - 292.89 ‖ 48 - 291.89

Наименование материала:	Х-ка материала	Кол-во шт на изделие	Всего шт	Всего пог. (кв.) м.	Всего кг.	Удельный вес (гр.)	Выдано
Профиль Сити		42.85	48	43.65	17.199	394	
Профиль Скребок		40.39	47	40.39	9.169	227	
Трос 1.8 мм.		12.66		12.66	0.17	13.400	
Резина зубчатая	Черная	39.01		39.01	10.143	260.000	
Полоса опорная, м.		81.86		81.86	1.097	13.400	
Втулка резиновая	9 мм.	338	338	---------	0.338	1.000	
Заглушка Сити, шт.		64	64	---------	0.178	2.780	
Заглушка Страйп, шт.		0	0	---------	0	0.610	
Заглушка мебельная, шт.		11	11	---------	0.002	0.161	
Кубик маленький		24	48	---------	0.031	0.655	
Винт для зажима		19	19	---------	0.035	1.829	
Втулка для зажима		19	19	---------	0.038	2.000	
Втулка пресовая		16	16	---------	0.023	1.437	
Клей АВИОРА, шт.:		17	17	---------	0	0.002	
Шайба М3*7		16	16	---------	0.02	0.125	
Саморез 2,9х9,5		192	192	---------	0.23	1.200	
Логотип-пластина	нет	0	0	---------	0	4.500	
Корпус логотипа		0	0	---------	0	7.500	
Саморез 2,9х9,5		24	24				
Винт М 3*16		8	8				
Гайка М3		8	8				
Заклепка вытяжная 3,2 *8		20	20				
Заклепка L24 (L25)		16	16				

Рисунок 3.12 – Відомість про списання на складі

Після вибору всіх необхідних специфікацій та шаблонів, що визначають спосіб виготовлення та елементи виробу буде проведено обчислення для автоматизації виготовлення та підбору елементної бази виробу, та представлено на відповідних формах(3.13-3.15).

Форма №_____

Расчетные данные для изготовления заказа: w020673

[redacted]

Бригада 3 Изделие Сити резина + скребок К1,4

Оплата за выполнение заказа - [redacted]

Параметры изготовления заказа Норма времени (чел-час): 13

Количество изделий - 1.000	Максимальная ширина изделия (длина профиля),мм - 2392	Максимальная длина изделия,мм - 2309
К-во линий стяжки - 9	Координата первого отверстия - 100	Шаг между тросами - 274
Количество дополнительных линий зажимов - 2	Тросы для установки дополнительных зажимов: 1 - 3	Профили для установки дополнительных зажимов: 16 - 32
Зазор между профилями (вид сверху) - 4.21	Количество резов по прямой (К=1,2) - 0	Количество резов по дуге (К=1,4) - 1
Дополнительный припуск - 0	Слева - Нет	Справа - Нет

Длина заготовок Профиль Сити:

1 - 2386 || 2 - 2386 || 3 - 2386 || 4 - 2432 || 5 - 2127.12 || 6 - 1892.22 || 7 - 1731.29 || 8 - 1605.34 || 9 - 1497.39 || 10 - 1404.43 || 11 - 1320.46 || 12 - 1245.49 || 13 - 1175.52 || 14 - 1110.55 || 15 - 1051.58 || 16 - 995.6 || 17 - 944.62 || 18 - 895.64 || 19 - 849.66 || 20 - 807.68 || 21 - 767.69 || 22 - 728.71 || 23 - 692.73 || 24 - 659.74 || 25 - 626.75 || 26 - 597.77 || 27 - 568.78 || 28 - 542.79 || 29 - 517.8 || 30 - 493.81 || 31 - 471.82 || 32 - 451.83 || 33 - 432.84 || 34 - 413.84 || 35 - 397.85 || 36 - 381.86 || 37 - 367.86 || 38 - 355.87 || 39 - 343.87 || 40 - 333.88 || 41 - 323.88 || 42 - 316.88 || 43 - 309.89 || 44 - 303.89 || 45 - 298.89 || 46 - 295.89 || 47 - 292.89 || 48 - 291.89

Длина заготовок Профиль Скребок:

1 - 2386 || 2 - 2386 || 3 - 2386 || 4 - 2254.06 || 5 - 1936.2 || 6 - 1764.27 || 7 - 1627.33 || 8 - 1516.37 || 9 - 1416.42 || 10 - 1330.45 || 11 - 1252.48 || 12 - 1181.51 || 13 - 1115.54 || 14 - 1054.57 || 15 - 997.59 || 16 - 944.61 || 17 - 895.63 || 18 - 848.65 || 19 - 805.67 || 20 - 764.69 || 21 - 725.71 || 22 - 688.72 || 23 - 654.74 || 24 - 621.75 || 25 - 591.76 || 26 - 562.77 || 27 - 535.79 || 28 - 510.8 || 29 - 485.81 || 30 - 463.82 || 31 - 442.82 || 32 - 423.83 || 33 - 403.84 || 34 - 387.85 || 35 - 371.85 || 36 - 357.86 || 37 - 344.87 || 38 - 332.87 || 39 - 321.88 || 40 - 311.88 || 41 - 303.88 || 42 - 296.89 || 43 - 290.89 || 44 - 285.89 || 45 - 281.89 || 46 - 246.89 || 47 - 245.89

Длина Профиль Сити с заглушкой в изделии:

1 - 2392 || 2 - 2392 || 3 - 2392 || 4 - 2392 || 5 - 2087.12 || 6 - 1852.22 || 7 - 1691.29 || 8 - 1565.34 || 9 - 1457.39 || 10 - 1364.43 || 11 - 1280.46 || 12 - 1205.49 || 13 - 1135.52 || 14 - 1070.55 || 15 - 1011.58 || 16 - 955.6 || 17 - 904.62 || 18 - 855.64 || 19 - 809.66 || 20 - 767.68 || 21 - 727.69 || 22 - 688.71 || 23 - 652.73 || 24 - 619.74 || 25 - 586.75 || 26 - 557.77 || 27 - 528.78 || 28 - 502.79 || 29 - 477.8 || 30 - 453.81 || 31 - 431.82 || 32 - 411.83 || 33 - 392.84 || 34 - 373.84 || 35 - 357.85 || 36 - 341.86 || 37 - 327.86 || 38 - 315.87 || 39 - 303.87 || 40 - 293.88 || 41 - 283.88 || 42 - 276.88 || 43 - 269.89 || 44 - 263.89 || 45 - 258.89 || 46 - 255.89 || 47 - 252.89 || 48 - 251.89

Длина Профиль Скребок в изделии:

1 - 2392 || 2 - 2392 || 3 - 2392 || 4 - 2229.06 || 5 - 1911.2 || 6 - 1739.27 || 7 - 1602.33 || 8 - 1491.37 || 9 - 1391.42 || 10 - 1305.45 || 11 - 1227.48 || 12 - 1156.51 || 13 - 1090.54 || 14 - 1029.57 || 15 - 972.59 || 16 - 919.61 || 17 - 870.63 || 18 - 823.65 || 19 - 780.67 || 20 - 739.69 || 21 - 700.71 || 22 - 663.72 || 23 - 629.74 || 24 - 596.75 || 25 - 566.76 || 26 - 537.77 || 27 - 510.79 || 28 - 485.8 || 29 - 460.81 || 30 - 438.82 || 31 - 417.82 || 32 - 398.83 || 33 - 378.84 || 34 - 362.85 || 35 - 346.85 || 36 - 332.86 || 37 - 319.87 || 38 - 307.87 || 39 - 296.88 || 40 - 286.88 || 41 - 278.88 || 42 - 271.89 || 43 - 265.89 || 44 - 260.89 || 45 - 256.89 || 46 - 252.89 || 47 - 251.89

Длина стяжки по каждому профилю:

1 - 30 || 2 - 78.43 || 3 - 126.86 || 4 - 175.29 || 5 - 223.72 || 6 - 272.15 || 7 - 320.58 || 8 - 369.01 || 9 - 417.44 || 10 - 465.87 || 11 - 514.3 || 12 - 562.73 || 13 - 611.16 || 14 - 659.59 || 15 - 708.02 || 16 - 756.45 || 17 - 804.88 || 18 - 853.31 || 19 - 901.74 || 20 - 950.17 || 21 - 998.6 || 22 - 1047.03 || 23 - 1095.46 || 24 - 1143.89 || 25 - 1192.32 || 26 - 1240.75 || 27 - 1289.18 || 28 - 1337.61 || 29 - 1386.04 || 30 - 1434.47 || 31 - 1482.9 || 32 - 1531.33 || 33 - 1579.76 || 34 - 1628.19 || 35 - 1676.62 || 36 - 1725.05 || 37 - 1773.48 || 38 - 1821.91 || 39 - 1870.34 || 40 - 1918.77 || 41 - 1967.2 || 42 - 2015.63 || 43 - 2064.06 || 44 - 2112.49 || 45 - 2160.92 || 46 - 2209.35 || 47 - 2257.78 || 48 - 2306.21

Рисунок 3.13 – Розрахункові дані для виготовлення виробу

Специфікаці стяжок виробу, як і сам орнамент придверних решіток буде відображена в автоматично згенерованому звіті, що на рисунку 3.14.

Основные линии стяжки

Координата отверстия слева	Координата отверстия справа	Начальный профиль	Конечный профиль	Длина троса, мм
100	2292	48	1	2557
374	2018	32	1	1782
648	1744	22	1	1297
922	1470	15	1	958
1196	1196	11	1	765
1470	922	8	1	619
1744	648	6	1	522
2018	374	4	1	425
2292	100	4	1	425

Дополнительные линии стяжки

Координата отверстия слева	Координата отверстия справа	Начальный профиль	Конечный профиль	Длина троса, мм
226	2166	48	32	1055
515	1877	26	22	474
803	1589	18	15	425
1072	1320	13	11	377
1374	1018	9	8	328
1587	805	7	6	328
1905	487	5	4	328

Винтовые крепления
0

Рисунок 3.14 – Специфікація форми виробу

Материалы

Наименование материала:	Длина профиля, мм.:	Количество на 1 изделие:	Количество на заказ:
Профиль Сити, м.		42.85	42.85
Профиль Скребок, м.		40.39	40.39
Трос 1.8 м.		12.66	12.66
Полоса опорная, м.		81.86	81.86
Втулка резиновая 9 мм.		338	338
Резина зубчатая Черная		39.01	39.01
Заглушка Сити, шт.		64	64
Заглушка мебельная, шт.		11	11
Кубик маленький		24	48
Винт для зажима		19	19
Втулка для зажима		19	19
Втулка пресовая		16	16
Шайба М3*7		16	16
Клей АВИОРА, шт.:		17	17
Саморез 2,9х9,5		192	192
Саморез 2,9х9,5		24	24
Винт М 3*16		8	8
Гайка М3		8	8
Заклепка вытяжная 3,2 *8		20	20
Заклепка L24 (L25)		16	16
Профиль Сити, м. (для отделки)		0.8	0.8

Срок сдачи выполненной заявки на склад не позднее:

Примечание: Заказ выполнить в соответствии с Заявкой покупателя, технологическими инструкциями и утвержденными контрольными образцами.

Получил бригадир _____

Рисунок 3.15 – Перелік матеріалів з яких буде складатися виріб

Система автоматизованого проектування придверних конструкцій дозволяє також графічно відобразити креслення виробу з врахуванням попередньо обчислених параметрів та специфікацій, як це показано на рисунку 3.16.

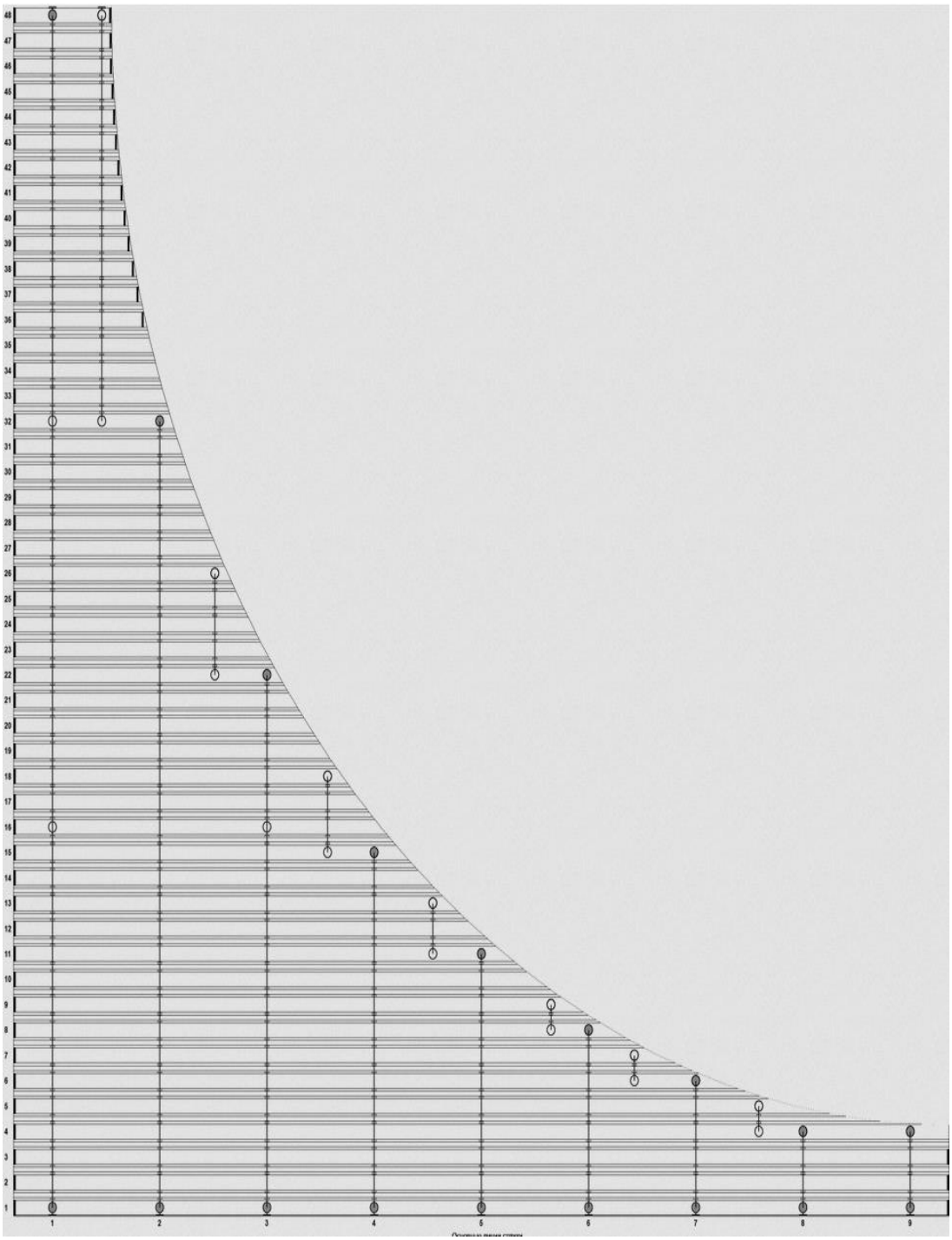


Рисунок 3.16 – Схематичне представлення зпроектованого виробу

Кінцевим етапом автоматизованого проектування виробу буде звіт про стан замовлення, короткі технічні специфікації виробу та схематичне відображення контуру виробу.

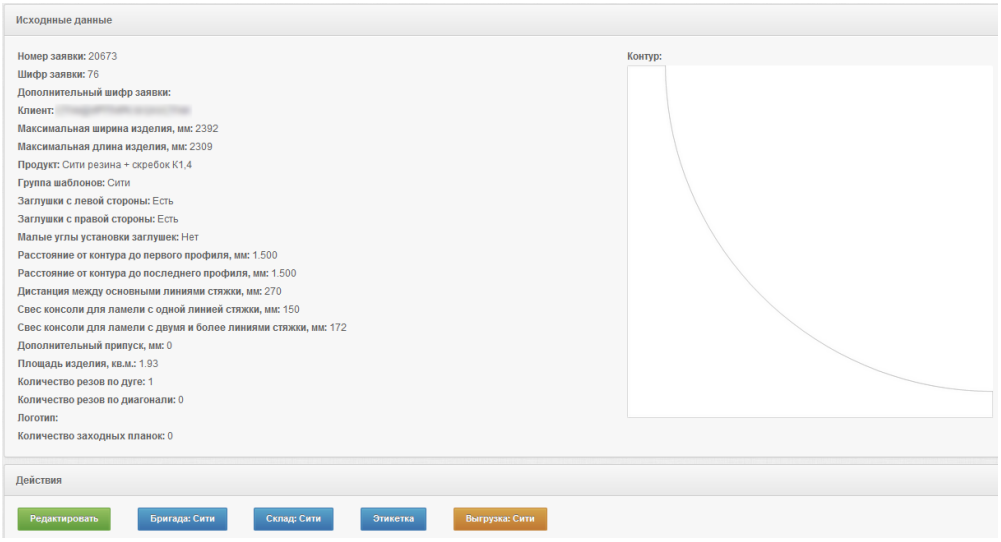


Рисунок 3.17 – Створений проект для автоматизованого проектування конструкцій придверних решіток.

Для тестування та впровадження інтернет ресурс розміщений на сервері Сервер HPE ProLiant ML10 Gen9 (837826-421), з мінімально встановленим програмним забезпеченням PHP/MySQL.

Висновки до розділу 3

1. Проведено дослідження математичного забезпечення системи автоматизованого проектування конструкцій придверних решіток, для побудови графічних моделей.

2. Розроблено та досліджено архітектуру бази даних для системи автоматизованого проектування конструкцій придверних решіток.

3. Проведено верифікацію та тестування розробленої автоматизованої системи конструювання придверних решіток та внесено тестовий приклад згідно акту впровадження та плану тестування.

ВИСНОВКИ

В процесі виконання магістерського проєкту отримані наступні наукові та практичні результати:

1. Проведенно аналізу існуючих рішень та архітектур веб-орієнтованих систем.

2. Обґрунтовано необхідність та розроблено алгоритми автоматизованого проектування конструкцій придверних решіток, що дозволило застосувати відповідне програмне та апаратне забезпечення для реалізації запропонованого рішення;

3. Проведено дослідження математичного забезпечення системи автоматизованого проектування конструкцій придверних решіток, для побудови графічних моделей.

4. Розроблено та досліджено архітектуру бази даних для системи автоматизованого проектування конструкцій придверних решіток.

5. Виконано проектування архітектури системи автоматизованого проектування консурукцій придверних решіток. Розроблено системи автоматизованого проектування конструкцій придверних решіток. Реалізовано програмне забезпечення для автоматизованої побудови конструкцій придверних решіток засобами PHP, MySQL.

6. Проведено верифікацію та тестування розробленої автоматизованої системи конструювання придверних решіток та внесено тестовий приклад згідно акту впровадження та плану тестування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alur D. Core J2EE Patterns: Best Practices and Design Strategies / D.Alur, D.Malks, J. Crupi. - Sun Microsystems Press: Prentice Hall, - 2003. – 271 p.
2. Almes G. The Impact of Language and System on Remote Procedure Call Design / G.Almes // Proc. of the 6th Int. Conference on Distributed Computing Systems: IEEE CS Press, 1986. – p. 414–421.
3. Su Jun. Data Transmission Optimal Routing in WSN Using Ant Colony Algorithm / Su Jun, V.Yatskiv, A. Sachenko, N. Yatskiv // Proc. of the International Conf. TCSET’ 2012. – Lviv-Slavsko, Ukraine, 2012 – Pp. 342–343.
4. Гладцин В.А. Сетевые технологии / В.А. Гладцин В.В. Яновский. - СПб.: СПбГЕТУ (ЛЕТИ), 1998 р. – 322 с.
5. Оліфер В.Г. Комп'ютерні мережі: Принципи, технології, протоколи / В.Г. Оліфер, Н.А. Оліфер. - К.: Освіта, 2005. - 472 с.
6. Вишняков В.М. Сучасні технології побудови комп'ютерних мереж / В.М.Вишняков. – К.: КНУБА, 2004. – 128 с.
7. Вуди Л. Microsoft Windows XP SP2 для "чайников" / Л.Вуди - М.: «Диалектика», 2007. – 284 с.
8. Гриценко В.И. Распределенные информационные системы. Состояния. Перспективы развития / В.И.Гриценко, А.А. Урсатьев // Управляющие системы и машины.– 2003.–№4.– с. 11-21.
9. Гук М. Аппаратные средства локальных сетей: Энциклопедия / М.Гук.- СПб.: Питер, 2000. – 576 с.
10. Хьюкаби Д. Руководство Cisco по конфигурированию коммутаторов Catalyst = Cisco Field Manual / Хьюкаби Д., Мак – Квери С. - М.: «Вильямс», 2004. – 293 с.
11. Кульгин М. Технологии корпоративных сетей: Энциклопедия / М.Кульгин. – СПб.: Питер, 2000. - 704 с.
12. Листвин А. В. Оптические волокна для линий связи / Листвин А. В., Листвин В. Н., Швирков Д. В. - М.: ЛЕСАРарт, 2003 р. – 294 с.

13. Белорусов И. И. Радиочастотные кабели / И. И.Белорусов. - М.: Госэнергоиздат, 1959. – 154 с.
14. Мартин Дж. Организация баз данных в вычислительных системах / Дж. Мартин. – М.: Мир, 1980. – 662с.
15. Новиков Ю. В. Основи локальних мереж. Курс лекцій / Ю. В.Новиков, С. В. Кондратенко. - К.: Інтернет-університет інформаційних технологій, 2005. – 216 с.
16. Новиков Ю.В. Локальные сети: архитектура, алгоритмы, проектирование / Ю.В.Новиков, С.В.Кондратенко.- М.: ЭКОМ, 2000.- 312 с.
17. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2000. – 672 с
18. Семенов А.Б. Структурированные кабельные системы / А.Б.Семенов, С.К.Стрижаков, И.Р.Сунчелей. – М.: ДМК Пресс, 2002. – 640 с.
19. Стеклов В.К.. Проектування телекомунікаційних мереж. / В.К. Стеклов, Л.Н. Беркман. – К.: Техніка, 2002.–792с.
20. Столлингс В. Беспроводные линии связи и сети / В.Столлингс. – М.: Вильямс, 2003. – 640 с.
21. Таненбаум Э. Компьютерные сети. 4-е изд./ Э. Таненбаум. – СПб.: Питер, 2003. – 992с.
22. Уолрэнд Дж. Телекоммуникационные и компьютерные сети: Вводный курс/ Дж.Уолрэнд.- М.: Постмаркет, 2003.- 480с.
23. Методичні рекомендації до виконання дипломного проекту з освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050102 «Комп’ютерна інженерія» фахового спрямування «Комп’ютерні системи та мережі» / О.М. Березький, Л.О.Дубчак, Р.Б. Трембач, Г.М. Мельник, Ю.М. Батько, С.В. Івасьєв / Під ред. О.М. Березького. - Тернопіль: ТНЕУ, 2016.– 65 с.
24. Бойчик В.Є. Автоматизація проектування конструкцій придверних решіток/ В.Є. Бойчик // Матеріали VI Всеукраїнської школи-семінару молодих вчених

і студентів «Сучасні комп’ютерні інформаційні технології» АСІТ-2017; 19-20 травня 2017 р. – Тернопіль, 2017. – С. 134.

25. Ськляров А.Я. Інструментальні засоби проектування, імітаційного моделювання і аналізу комп'ютерних мереж/ Ськляров А.Я., Пономаренко л.А., Щелкунов в.І., - Навчальний посібник. – До: Нук. Думання, 2002. – 508 с.
26. Яцків В. В. Метод розподілу трафіку в безпровідних сенсорних мережах на основі системи залишкових класів / В. В. Яцків // Тези міжнародної науково-технічної конференції “Контроль і управління в складних системах”, КУСС – 2012, «УНІВЕРСУМ – Вінниця», 2012. С. 39.
27. Яцкив Н. Г. Метод кодирования данных в беспроводных компьютерных сетях на основе преобразования системы остаточных классов / Н. Г. Яцкив, В. В. Яцкив, Р. В. Крепич // Физика, математика, информатика. Вестник Брестского государственного технического университета. – 2007. – № 5 (47). – С. 8–10.
28. Яцків В. В. Проблеми створення комп’ютерних мереж на основі відкритих лазерних каналів зв’язку / В. В. Яцків // Тези міжнародної науково-технічної конференції. “Контроль і управління в складних системах”, КУСС – 2003. – Вінниця: «УНІВЕРСУМ – Вінниця», 2003. – С. 70.
29. Самойлов, В. Д. Модельное конструирование компьютерных приложений / В. Д. Самойлов. - К.: Наукова думка. - 2007. - 198 с.
30. Лаврищева, Е. М. Методы программирования: теория, инженерия, практика / Е. М. Лаврищева. - К.: Наукова думка. - 2006. - 451 с.
31. Buasilovsky, P. Adaptive and intelligent Web-based educational systems / P. Buasilovsky, C Peylo // International Journal of Artificial Intelligence in Education. Special Issue on Adaptive and Intelligent Web-based Educational Systems -2003. -№13 (2-4). -P. 159-172.
32. Murray, T. Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art / T. Murray. // International Journal of Artificial Intelligence in Education. -1999.-№10-P. 98-129

33. Андон, Ф. И. Логические модели интеллектуальных информационных систем / Ф. И. Андон, А. Е. Яшунин, В. А. Резниченко. - К: Наукова думка.- 1999.-397 с.
34. Олецький О. В. Застосування формальних моделей онтологій для формалізації інформаційних потоків у системах управління контентом / О. В. Олецький //Теоретичні та прикладні аспекти побудови програмних систем. Матеріали міжнародної конференції TAAPSD'2005. Київ, 7-9 грудня 2005 р. - С. 26-29.
35. Lewisand D. Lewisand D. Acomparison of two learning algorithms for text categorization [Електронний ресурс] / D. Lewisand, M. Ringuette // In Third Annual Symposium on Document Analysis and Information Retrieval. – 1994. – Режим доступу до ресурсу: http://www.research.att.com/~lewis/papers/lewis94b.ps.
36. FreshKnowledge CMS - онтологічно-орієнтована система керування контентом, розроблена здобувачем [Електронний ресурс]. - Режим доступу : http://www.freshknowledge.net.
37. Елизаренко, Г. Н. Проектирование компьютерных курсов обучення: концепция, язык, структура / Г. Н. Елизаренко. - К.: НТУУ «КПИ», 2001.
38. Люгер, Джордж, Ф. Искусственный интеллект: стратегии и методы решення сложных проблем / Люгер, Джордж, Ф. - 4-е издание.: Пер. с англ.- М.: Издательский дом «Вильяме», 2005. - 864 с.
39. TENCompetence - European research project for lifelong competence development [Електронний ресурс]. - Режим доступу : **Ошибка! Недопустимый объект гиперссылки.**www.tencompetence.org/.
40. Валгина, Н. С. Теория текста: Учебное пособие / Н. С. Валгина. - Москва. Изд-во МГУП «Миркниги» - 1998. - 210 с.
41. Дивак М. П., Шпінталь М.Я., Шевчук Р.П., Козак О.Л., Пукас А.В., Спільчук В.М., Гончар Л.І. Методичні рекомендації до виконання та захисту дипломної роботи на здобуття освітньо-кваліфікаційного рівня магістр за спеціальностями: 8.05010301 "Програмне забезпечення систем" та

8.05010302 “Інженерія програмного забезпечення” // Тернопіль : Економічна думка, 2011. — 31 с.

ДОДАТОК А.

Розробка основних функцій

При розробці основних функціональних можливостей було розроблено ряд функцій для роботи з базою даних та маніпуляції данними.

```
<?php
/**
 * Отримання даних зображення
 */
function getImageData($id)
{
    $temp = [];

    // завантаження зображення
    $url = Yii::app()->params['file_folders']['drawings'] . 'cropped_' . $id . '.jpg';
    $image = @imagecreatefromjpeg($url);

    if($image){
        $imgWidth = imagesx($image);
        $imgHeight = imagesy($image);

        // дозволені відтінки контуру: RGB(0,0,0)-RGB(30,30,30);
        $lineColors = [];
        for($i = 0; $i <= 30; $i++){
            $lineColors[] = imagecolorallocate($image, $i, $i, $i);
        }

        // дані настройки дозволяють не вилітати по тайм-ауту
        ini_set('max_execution_time', 300);
        ini_set('memory_limit', '-1');

        // масиви для зберігання проміжних даних
        $arrayLeadingX = [];
        $arrayLeadingY = [];
        $points = []; // крайні точки відрізків в рядках
        $cuts = []; // відрізки в рядках


        // обхід зображення по всіх стовпчиках пікселів
        for($x = 0; $x < $imgWidth; $x++){
            $yMin = 0;
            $yMax = 0;
            // знаходження нижнього краю контура в стовбці
            for($y = 0; $y < $imgHeight; $y++){
                $colorOnPoint = imagecolorat($image, $x, $y);
                if(in_array($colorOnPoint, $lineColors)){
                    $yMin = $y;
                    break;
                }
            }
            // знаходження верхнього краю контура в стовбці
            for($y = $imgHeight-1; $y >= 0; $y--){
                $colorOnPoint = imagecolorat($image, $x, $y);
                if(in_array($colorOnPoint, $lineColors)){
                    $yMax = $y;
                    break;
                }
```

```
        }
        $arrayLeadingY[$yMin][] = $x;
        $arrayLeadingY[$yMax][] = $x;
    }

    // обхід зображення по всіх рядках пікселів
    for($y = 0; $y < $imgHeight; $y++){
        $xMin = 0;
        $xMax = 0;
        // знаходження лівого краю контура в рядку
        for($x = 0; $x < $imgWidth; $x++){
            $colorOnPoint = imagecolorat($image, $x, $y);
            if(in_array($colorOnPoint, $lineColors)){
                $xMin = $x;
                break;
            }
        }
        // знаходження правого краю контура в рядку
        for($x = $imgWidth-1; $x >= 0; $x--){
            $colorOnPoint = imagecolorat($image, $x, $y);
            if(in_array($colorOnPoint, $lineColors)){
                $xMax = $x;
                break;
            }
        }
        $arrayLeadingY[$y][] = $xMin;
        $arrayLeadingY[$y][] = $xMax;
    }

    // дані збиралися для рядків, тобто в масиві кожен елемент верхнього рівня відповідав за координату Y, а внутрішні елементи - за X
    // для кожного рядка залишаємо тільки унікальні X і сортуємо
    for($i = 0; $i < count($arrayLeadingY); $i++){
        $arrayLeadingY[$i] = array_unique($arrayLeadingY[$i], SORT_NUMERIC);
        sort($arrayLeadingY[$i], SORT_NUMERIC);
    }

    // масив рядків сортується по ключах, так отримаємо всі рядки в правильному порядку
    ksort($arrayLeadingY, SORT_NUMERIC);
    // перевертаємо масив вверх ногами, так як зчитування данних з зображення йшло від верхнього лівого кута, а всі подальші розрахунки будуть йти від нижнього лівого
    $arrayLeadingY = array_reverse($arrayLeadingY);


    // формуємо масив для стовбчиків з попередніх даних, залишаємо тільки унікальні Y і сортуємо
    foreach($arrayLeadingY as $y => $value){
        foreach($value as $x){
            $arrayLeadingX[$x][] = $y;
        }
    }
    for($i = 0; $i < count($arrayLeadingX); $i++){
        $arrayLeadingX[$i] = array_unique($arrayLeadingX[$i], SORT_NUMERIC);
        sort($arrayLeadingX[$i], SORT_NUMERIC);
        $arrayLeadingX[$i] = array($arrayLeadingX[$i][0], $arrayLeadingX[$i][count($arrayLeadingX[$i])-1]);
    }

    // зберігаємо в масив точки контуру по рядках
    for($y = 0; $y < count($arrayLeadingY); $y++){
        $counter = count($arrayLeadingY[$y]);
        for($x = 0; $x < $counter - 1; $x++){
            if($x+1 <= $counter-1 && abs($arrayLeadingY[$y][$x+1] - $arrayLeadingY[$y][$x]) > 1){
                $center = (int)floor(($arrayLeadingY[$y][$x+1] + $arrayLeadingY[$y][$x])/2);
                if($y > $arrayLeadingX[$center][0] && $y < $arrayLeadingX[$center][1]){
                    $cuts[$y][] = abs($arrayLeadingY[$y][$x+1] - $arrayLeadingY[$y][$x]) + 1;
```

```
                    $points[$y][] = ($arrayLeadingY[$y][$x]+1) . '-' . ($arrayLeadingY[$y][$x+1]+1);
                }
            }
        }
    }
    // значення в рядках об'єднуються через ":"
    $tempCuts = array();
    $tempPoints = array();
    foreach($cuts as $cut){
        $tempCuts[] = implode(':', $cut);
    }
    foreach($points as $point){
        $tempPoints[] = implode(':', $point);
    }
    $points = $tempPoints;

    // всі рядки об'єднуються через "|" для подальшого зберігання в БД
    $temp['cuts'] = "".implode('|',$cuts)."";
    $temp['points'] = "" . implode('|', $points) . "";
    $temp['width'] = $imgWidth;
    $temp['length'] = $imgHeight;
  }
  // видаляєм з пам'яті зображення
  imagedestroy($image);
  // повертаєм результуючий масив для збереження в БД
  return $temp;
}

/**
 * Визначення розмірів профілів у виробі
 */
function profilesLength()
{
  $points_count = {count_profiles}; //кількість профілів згідно формули
  $interval = {step}; //крок конструкції в пікселях згідно формули

  // розбиваємо дані про відрізки на рядки
  $cuts = explode('|', {grid.cuts});

  // розбиваємо рядки на відрізки
  for($i = 0; $i < count($cuts); $i++)
    $cuts[$i] = array_map('intval', explode(':', $cuts[$i]));
  // розбиваємо дані про крайні точки на частини
  $points = explode('|', {grid.points});
  // розділяємо всі точки в рядках
  for($x = 0; $x < count($points); $x++):
    if(strpos($points[$x], ':')):
      $points[$x] = explode(':', $points[$x]);
    else:
      $points[$x] = array($points[$x]);
    endif;
  endfor;

  $profiles = [];
  for($k = 0; $k < $points_count; $k++):
    $botCut = $k*$interval; // нижня грань профіля
    $topCut = 30+$k*$interval; // верхня грань профіля
    $countTopSegments = count($cuts[$topCut]);
    $countBotSegments = count($cuts[$botCut]);
    $tempProfiles = array();

    // визначаємо кількість і довжину сегментів профіля
    if($countBotSegments === $countTopSegments):
      for($i = 0; $i < $countBotSegments; $i++):
```

```
            $tempProfiles[] = round(max($cuts[$botCut][$i], $cuts[$topCut][$i]),2);
         endfor;
      elseif (($countBotSegments === 0 && $countTopSegments > 0) || ($countTopSegments === 0 && $countBotSegments > 0)):
         $usable = $countTopSegments === 0 ? $botCut : $topCut;
         for($i = 0; $i < count($cuts[$usable]); $i++):
            $tempProfiles[] = round($cuts[$usable][$i],2);
         endfor;
      else:
         $smaller = $countBotSegments < $countTopSegments ? $botCut : $topCut;
         $smallerLength = $cuts[$smaller][0];

         $bigger = $countBotSegments < $countTopSegments ? $topCut : $botCut;
         $leftSegment = explode('-', $points[$bigger][0]);
         $rightSegment = explode('-', $points[$bigger][count($points[$bigger])-1]);
         $biggerLength = abs($rightSegment[1]-$leftSegment[0])+1;
         $tempProfiles[] = round(max($smallerLength, $biggerLength),2);
      endif;

      $profiles[] = implode(' + ', $tempProfiles);
   endfor;

   return $profiles;
}

/**
 * Визначення, де будуть проходити основні лінії стяжки
 */
function calculateMainLinks()
{
   $links = explode(' || ', {links_coordinates}); // координати основних ліній стяжки згідно формули
   $profiles = {count_profiles}; // кількість профілів згідно формули
   $step = {step}; //крок конструкції в пікселях згідно формули

   // розбиваємо дані про відрізки на рядки
   $points = explode('|', {grid.points});

   // розділяємо всі точки в рядках
   for($x = 0; $x < count($points); $x++):
      if(strpos($points[$x], ':')):
         $points[$x] = explode(':', $points[$x]);
      else:
         $points[$x] = array($points[$x]);
      endif;
   endfor;

   $temp = [];

   // визначення, які профілі перетинаються лініями стяжки

   // ітерація по всіх розрахованих можливих основних стяжках
   foreach($links as $link):
      $linkExtended = $link;
      // ітерація по всіх профілях
      for($k = 0; $k < $profiles; $k++):
         $coord5mm = $k*$step + 5; // нижня грань профіля + 5мм
         $coord25mm = $k*$step + 25; // нижня грань профіля + 25мм
         $segments5mm = count($points[$coord5mm]); // сегменти профіля у конкретному рядку
         $segments25mm = count($points[$coord25mm]);

         if($segments5mm == $segments25mm):
            for($m = 0; $m < $segments5mm; $m++):
               $delimiters5mm = explode('-', $points[$coord5mm][$m]);
               $delimiters25mm = explode('-', $points[$coord25mm][$m]);
```

```
            if($linkExtended - $delimiters5mm[0] > 5 && $delimiters5mm[1] - $linkExtended > 5 && $linkExtended
- $delimiters25mm[0] > 5 && $delimiters25mm[1] - $linkExtended > 5)
                $temp[$link][] = $k+1;

          endfor;
        endif;

      endfor;
   endforeach;

   // серіалізуємо дані в конкретний формат
   $main_links = [];
   foreach($temp as $key=>$link):
      if(count($link) >= 2) $main_links[] = $key . '-' . $link[0] . '-' . $link[count($link)-1] . '-' . ((count($link)-1) *
{step});
   endforeach;

   return "" . implode('||', $main_links) . "";
}

/**
 * Розахунок додаткових ліній стяжки
 */
function calculateAdditionalLinks()
{
   $links = explode('||', trim({links_main}, "")); // основні лінії стяжки
   $profiles_count = {count_profiles}; // кількість профілів
   $step = {step}; //крок конструкції в пікселях згідно формули
   $links_coordinates = explode(' || ', trim({links_coordinates}, "")); // координати основних ліній стяжки
   $distance_links = {real_dist_link}; // фактична розрахована відстань між основними стяжками

   // розділяємо всі точки в рядках
   $points = explode('|', {grid.cuts});
   for($x = 0; $x < count($points); $x++):
      if(strpos($points[$x], ':')):
         $points[$x] = explode(':', $points[$x]);
      else:
         $points[$x] = array($points[$x]);
      endif;
   endfor;

   $profiles_links = [];
   $pureProfileLinks = [];

   // визначаємо точки перетину основними стяжками для кожного профіля і шукаємо верхній/нижній профіль
для кожної основної стяжки
   for($k = 1; $k <= $profiles_count; $k++):
      foreach($links as $link):
         $temp = explode('-', $link);
         if($k >= $temp[1] && $k<= $temp[2]):
            $profiles_links[$k][] = (int)$temp[0];
         endif;
         if($k == $temp[1] || $k == $temp[2]):
            $pureProfileLinks[$k][] = (int)$temp[0];
         endif;
      endforeach;
   endfor;

   // для кожного профіля ділимо стяжки на групи
   // в групі можуть бути тільки такі стяжки, відстань між якими попарно не більша за певну розраховану
   // Приклад:
   // Виріб увігнутий по осі У, через конкретний профіль проходить такі стяжки: 1-2-3-6-7-8
   // буде поділено на групи - [[1,2,3],[6,7,8]]
```

```
foreach ($profiles_links as $profile => $links):
   $linkGroups = [];
   $groupCount = 0;
   $linkGroups[$groupCount][] = $links[0];

   for($t = 0; $t < count($links)-1; $t++):
      if($links[$t+1] - $links[$t] == $distance_links):
         $linkGroups[$groupCount][] = $links[$t+1];
      else:
         $groupCount++;
         $linkGroups[$groupCount][] = $links[$t+1];
      endif;
   endfor;
   $profiles_links[$profile] = $linkGroups;
endforeach;

$profiles = array(
   'left'=>array(
      'single'=>[],
      'double'=>[]
   ),
   'right'=>array(
      'single'=>[],
      'double'=>[]
   )
);

// для кожноъ групи визначаємо граничний розмір звисаючого куска профіля
foreach($profiles_links as $profile => $links):
   foreach ($links as $linksGroup):
      $right = $linksGroup[count($linksGroup)-1];
      $left = $linksGroup[0];
      $overhang = 0; // допустиме значення висячого куска
      $type = '';

      if(count($linksGroup) >= 2):
         $overhang = {grid.overhang_two_links}; // сегмент профіля закріплено 2+ стяжками
         $type = 'double';
      else:
         $overhang = {grid.overhang_one_link}; // сегмент профіля закріплено 1 стяжкою
         $type = 'single';
      endif;

      $coords = $profile-1*$step;
      for($m = 0; $m < count($points[$coords]); $m++):
         $delimiters = array_map('intval', explode('-', $points[$coords][$m]));

         // визначаємо зліва чи справа від групи стяжок потрібні додаткові кріплення
         if($delimiters[0] < $left && $right < $delimiters[1]):
            if($delimiters[1] - $right > $overhang):
               $profiles['right'][$type][$profile][] = array(
                  'left' => $left,
                  'right' => $right
               );
            endif;
            if($left - $delimiters[0] > $overhang):
               $profiles['left'][$type][$profile][] = array(
                  'left' => $left,
                  'right' => $right
               );
            endif;
         endif;
      endfor;
   endforeach;
```

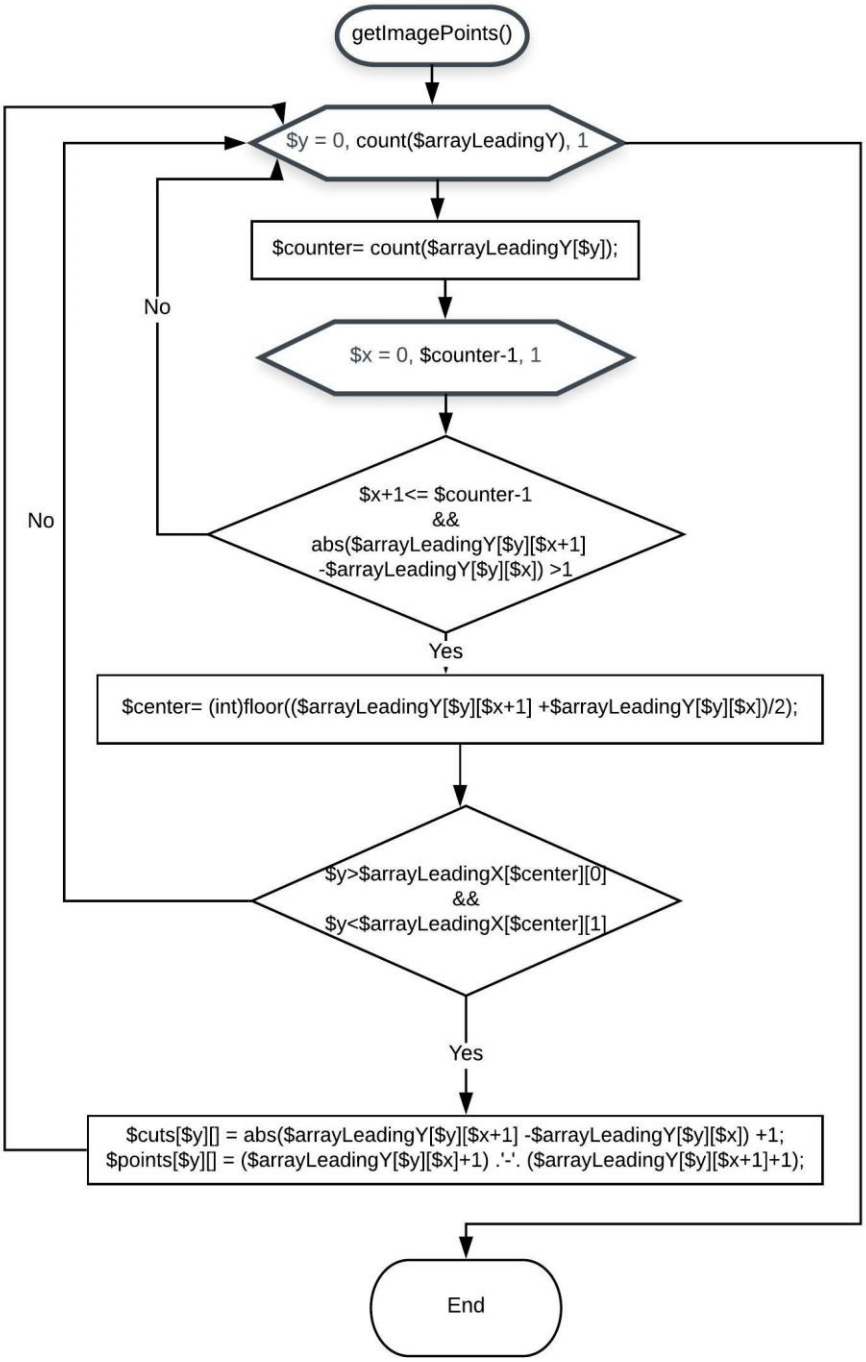
```
    endforeach;

    $additional_links = [];
    // цикл по всіх профілях
    foreach ($profiles as $side => $type):
        // цикл по обох сторонах
        foreach ($type as $typeName => $amount):
            // потрібні додаткові стяжки
            if(!empty($amount)):
                $profiles_numbers = array_keys($amount);
                sort($profiles_numbers, SORT_NUMERIC);
                $groups = array();
                $groupCount = 0;

                // профілі діляться на групи,
                // наприклад, потрібно з правої сторони виробу додаткові тяжки на профіляй №№ 9-10-11-12-15-16-21-
22-23
                // результат буде [[9,10,11,12],[15,16],[21,22,23]]
                $groups[$groupCount][] = $profiles_numbers[0];
                for($t = 0; $t < count($profiles_numbers) - 1; $t++):
                    if($profiles_numbers[$t+1] - $profiles_numbers[$t] == 1):
                        $groups[$groupCount][] = $profiles_numbers[$t+1];
                    else:
                        $groupCount++;
                        $groups[$groupCount][] = $profiles_numbers[$t+1];
                    endif;
                endfor;

                // проходимо всі знайдені групи
                while(count($groups) > 0):
                    usort($groups, function($a, $b){
                        return $a[0] - $b[0];
                    });
                    $count = count($groups[0]);
                    $group = $groups[0];

                    $limit = 25; // відступ додаткової стяжки від зрізу профіля

                    // для кожного профіля в групі визначаємо приблизне розташування додаткової стяжки на основі
відступу
                    $links = array();
                    foreach ($group as $prof):
                        foreach ($amount[$prof] as $entry):
                            $profilePlus25mm = ($prof - 1)*$step + 25;
                            for($m = 0; $m < count($points[$profilePlus25mm]); $m++):
                                $delimiters = array_map('intval', explode('-', $points[$profilePlus25mm][$m]));
                                if($delimiters[0] < $entry['left'] && $entry['right'] < $delimiters[1]):
                                    if($side == 'right'):
                                        $links[] = round($delimiters[1] - 12, 0);
                                    elseif($side == 'left'):
                                        $links[] = round($delimiters[0] + 12, 0);
                                    endif;
                                endif;
                            endfor;
                        endforeach;
                    endforeach;

                    // шукаємо загальне розташування стяжки для групи, залежить від сторони виробу
                    $generalLink = $side == 'right' ? min($links) : max($links);

                    // ще один прохід по кожному профілю
                    foreach ($group as $prof):
                        foreach ($amount[$prof] as $entry):
                            $coords0mm = ($prof-1)*$step; // нижній край профіля
```

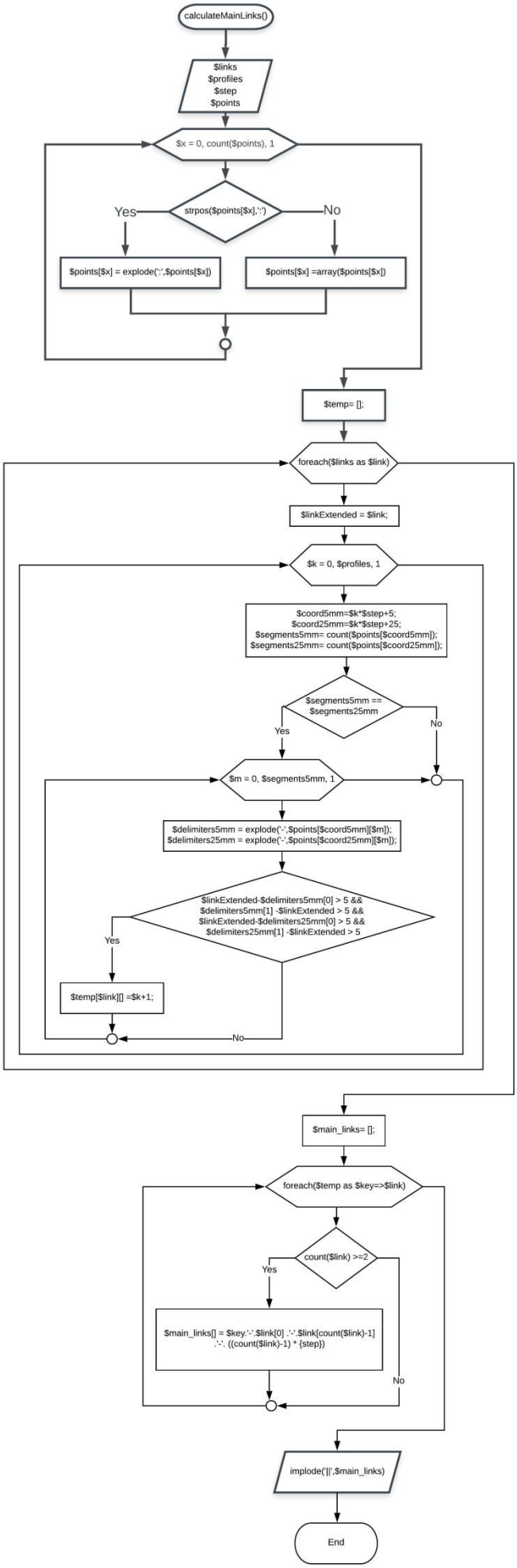
```
                $coords29mm = ($prof-1)*$step + 29; // верхній край профіля

                $segments0mm = count($points[$coords0mm]);
                $segments29mm = count($points[$coords29mm]);

                // перепризначаємо загальну координату додаткової стяжки для групи, якщо для верхній/нижній краю профіля не виконується правило відступу
                // після цього циклу отримаємо остаточне значення додаткової координати
                for($m = 0; $m < $segments0mm; $m++):
                   $delimiters0mm = array_map('intval', explode('-', $points[$coords0mm][$m]));
                   for($p = 0; $p < $segments29mm; $p++):
                      $delimiters29mm = array_map('intval', explode('-', $points[$coords29mm][$p]));
                      if($delimiters0mm[0] < $entry['left'] && $entry['right'] < $delimiters0mm[1] && $delimiters29mm[0] < $entry['left'] && $entry['right'] < $delimiters29mm[1]):
                         if($side == 'right'):
                            if($delimiters0mm[1] - $generalLink < $limit || $delimiters29mm[1] - $generalLink < $limit):
                               $generalLink = min($delimiters0mm[1] - $limit, $delimiters29mm[1] - $limit);
                            endif;
                         elseif ($side == 'left'):
                            if($generalLink - $delimiters0mm[0] < $limit || $generalLink - $delimiters29mm[0] < $limit):
                               $generalLink = max($delimiters0mm[0] + $limit, $delimiters29mm[0] + $limit);
                            endif;
                         endif;
                      endif;
                   endfor;
                endfor;
             endforeach;
          endforeach;

          $overhang = $typeName == 'single' ? {grid.overhang_one_link} : {grid.overhang_two_links}; // значення допустимого звисання для групи
          $notFit = array();

          // повторний прохід
          // якшо для якогось профіля звисання більше, ніж дозволено - виносимо в окремий масив
          foreach ($group as $prof):
             foreach ($amount[$prof] as $entry):
                $coords29mm = ($prof-1)*$step+29;
                for($m = 0; $m < count($points[$coords29mm]); $m++):
                   $delimiters = array_map('intval', explode('-', $points[$coords29mm][$m]));
                   if($delimiters[0] < $entry['left'] && $entry['right'] < $delimiters[1]):
                      if(($side == 'right' && $delimiters[1] - max($entry['right'], $generalLink) > $overhang) || ($side == 'left' && min($generalLink, $entry['left']) - $delimiters[0] > $overhang)):
                         $notFit[] = $prof; // <-- тут
                      endif;
                   endif;
                endfor;
             endforeach;
          endforeach;

          // якшо є профілі для яких не підходить загальна координата стяжки
          if(!empty($notFit)):
             $temp = array();
             $temp[] = $notFit; // додаєм ці профілі в тимчасовий масив
             $temp[] = array_diff($group, $notFit); // додаєм правильні профілі в тимчасовий масив
             $newGroups = array();
             $grCount = 0;

             // цикл по тимчасовому масиву
             // повторно ділимо профілі на групи і додаємо в кінець головного масиву груп
             for($m = 0; $m < count($temp); $m++):
                if(!empty($temp[$m])):
```

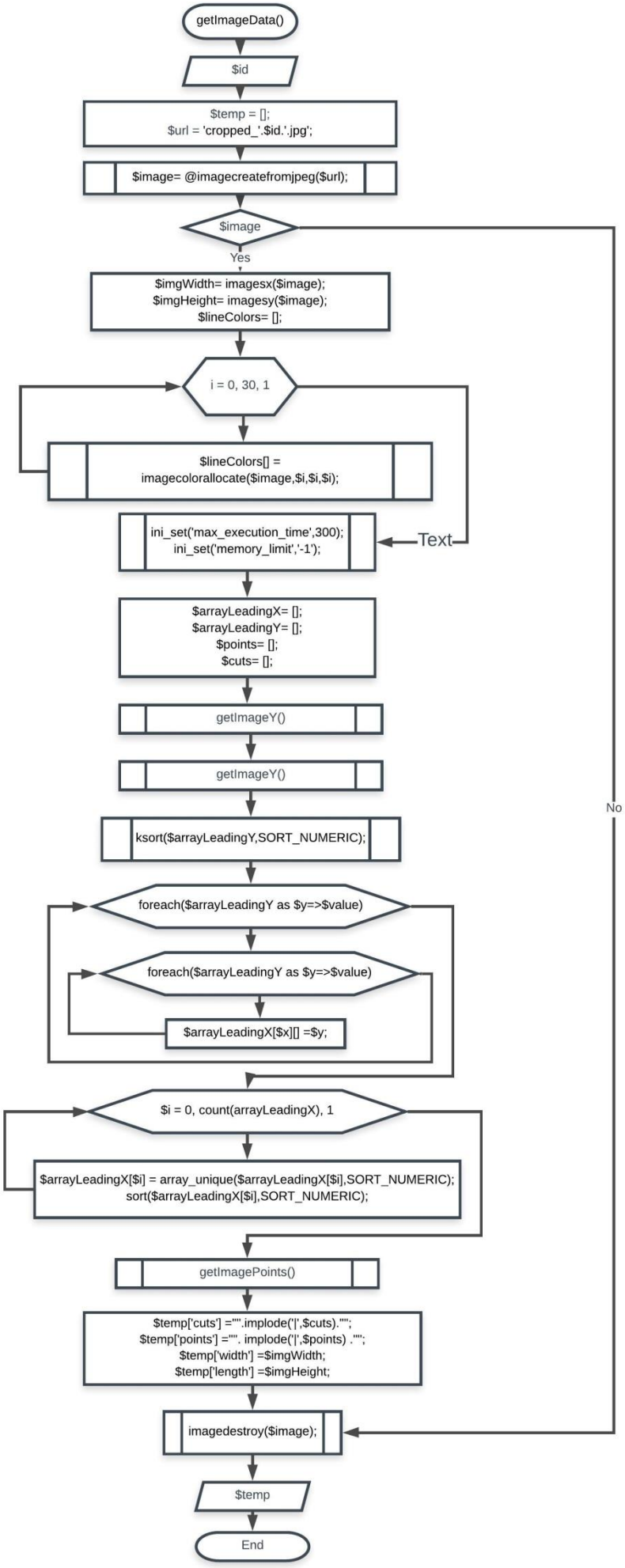
```
                        $subGroup = $temp[$m];
                        sort($subGroup);
                        $keys = array_keys($subGroup);
                        $newGroups[$grCount][] = $subGroup[$keys[0]];
                        for($r = 0; $r < count($subGroup)-1; $r++):
                           if($subGroup[$r+1] - $subGroup[$r] == 1):
                              $newGroups[$grCount][] = $subGroup[$r+1];
                           else:
                              $grCount++;
                              $newGroups[$grCount][] = $subGroup[$r+1];
                           endif;
                        endfor;
                        $grCount++;
                     endif;
                  endfor;
                  foreach ($newGroups as $newGroup):
                     $groups[] = $newGroup;
                  endforeach;
                  array_shift($groups);
                  continue; // коли всі профілі погруповані і додані в верхній масив (можна рахувати верхній масив чергою) - закінчуємо поточний крок циклу і переходимо до наступної групи
               endif;
               //для всіх профілів в групі підходить загальна координата

               // додаткова стяжка не може бути ближче ніж за 50мм від основної, якшо ближче - ігноруємо таку стяжку
               $threshold = false;
               foreach($links_coordinates as $link):
                  if(abs($link - $generalLink) <= 50) $threshold = true;
               endforeach;

               // стяжка може бути додана до виробу
               if($threshold === false && count($group) > 1):
                  $finalCoord = 0;

                  // якшо вже є додаткова стяжка з такою координатою (можливо, коли виріб увігнутий по осі X) - додаємо 1мм, щоб не перезаписати
                  if(isset($additional_links[$generalLink])):
                     $finalCoord = $generalLink + 1;
                  else:
                     $finalCoord = $generalLink;
                  endif;

                  //додаємо стяжку в масив, ключ - координата, значення - масив профілів через які проходить
                  $additional_links[$finalCoord] = $group;
               endif;
               array_shift($groups); // видаляємо поточну групу з черги
            endwhile;
         endif;
      endforeach;
   endforeach;

   // сералізація
   $final = array();
   if(!empty($additional_links)):
      ksort($additional_links);
      foreach($additional_links as $key=>$link):
         sort($link, SORT_NUMERIC);
         $final[] = $key . '-' . $link[0] . '-' . $link[count($link)-1] . '-' . (($link[count($link)-1] - $link[0]) * $step + 30);
      endforeach;
   endif;

   return empty($additional_links) ? 0 : "" . implode('||', $final) . "";
}
```

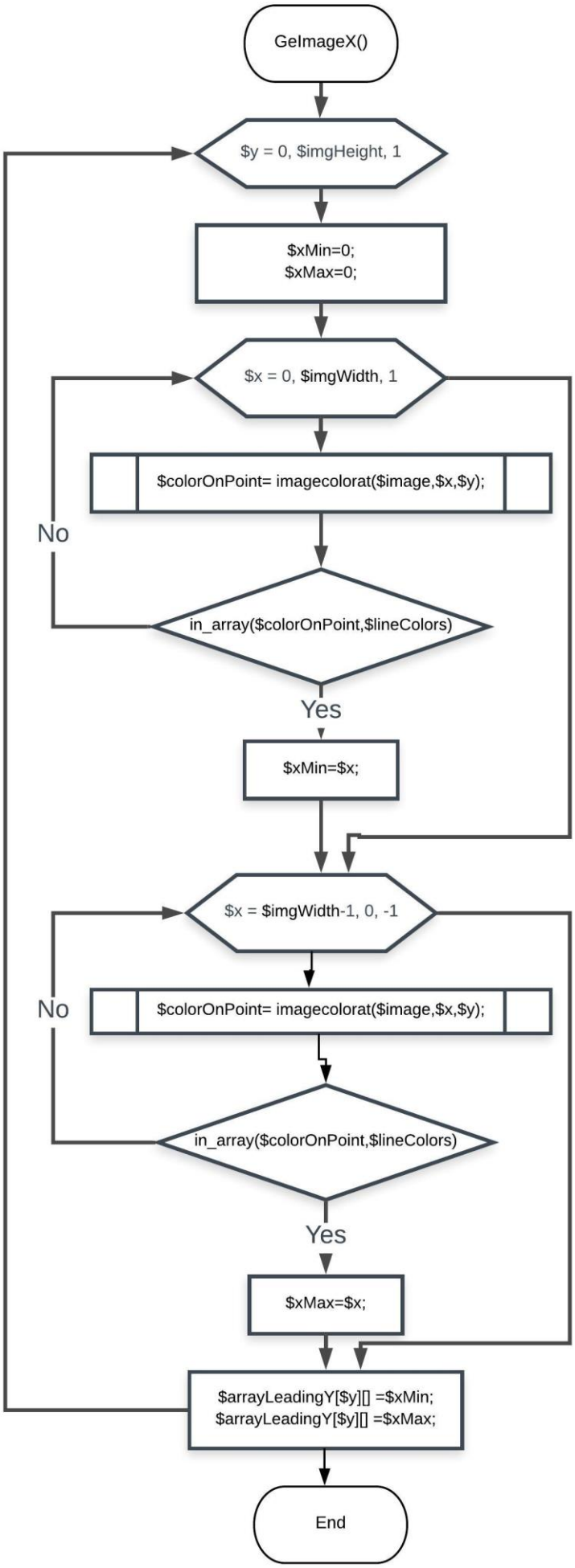
ДОДАТОК В

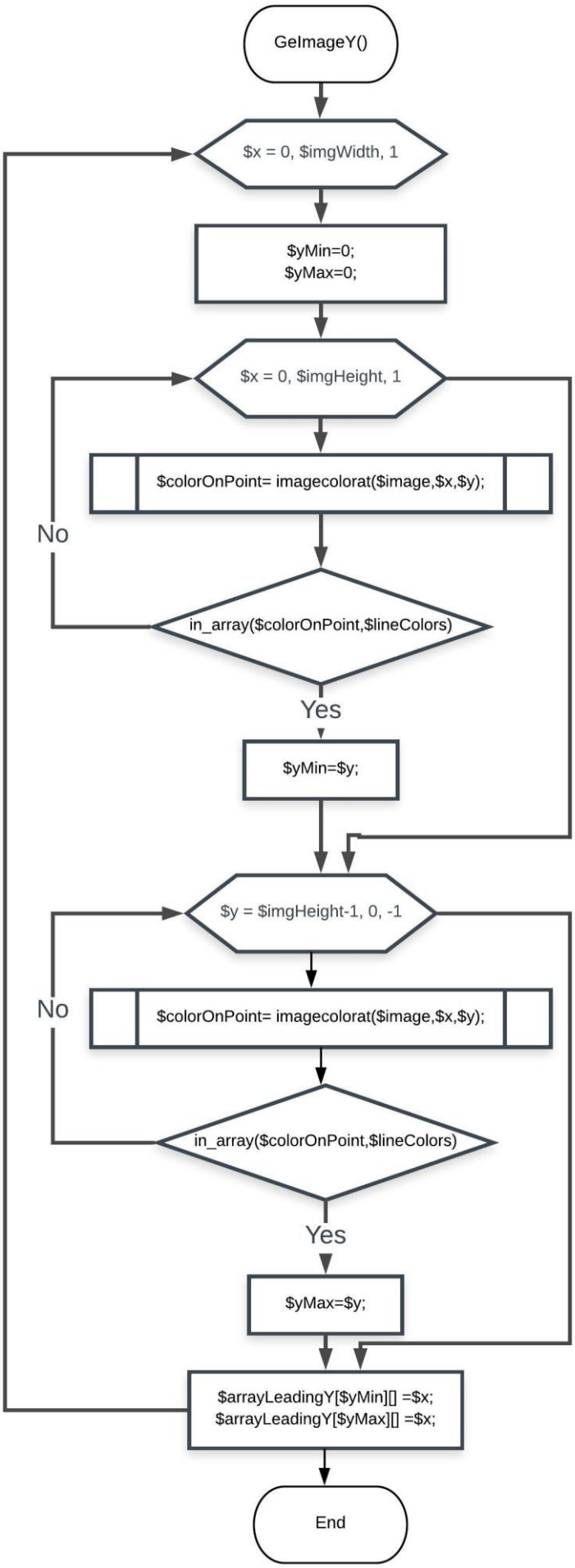
Розроблені алгоритми для проведення автоматизованого проектування конструкцій придверних решіток

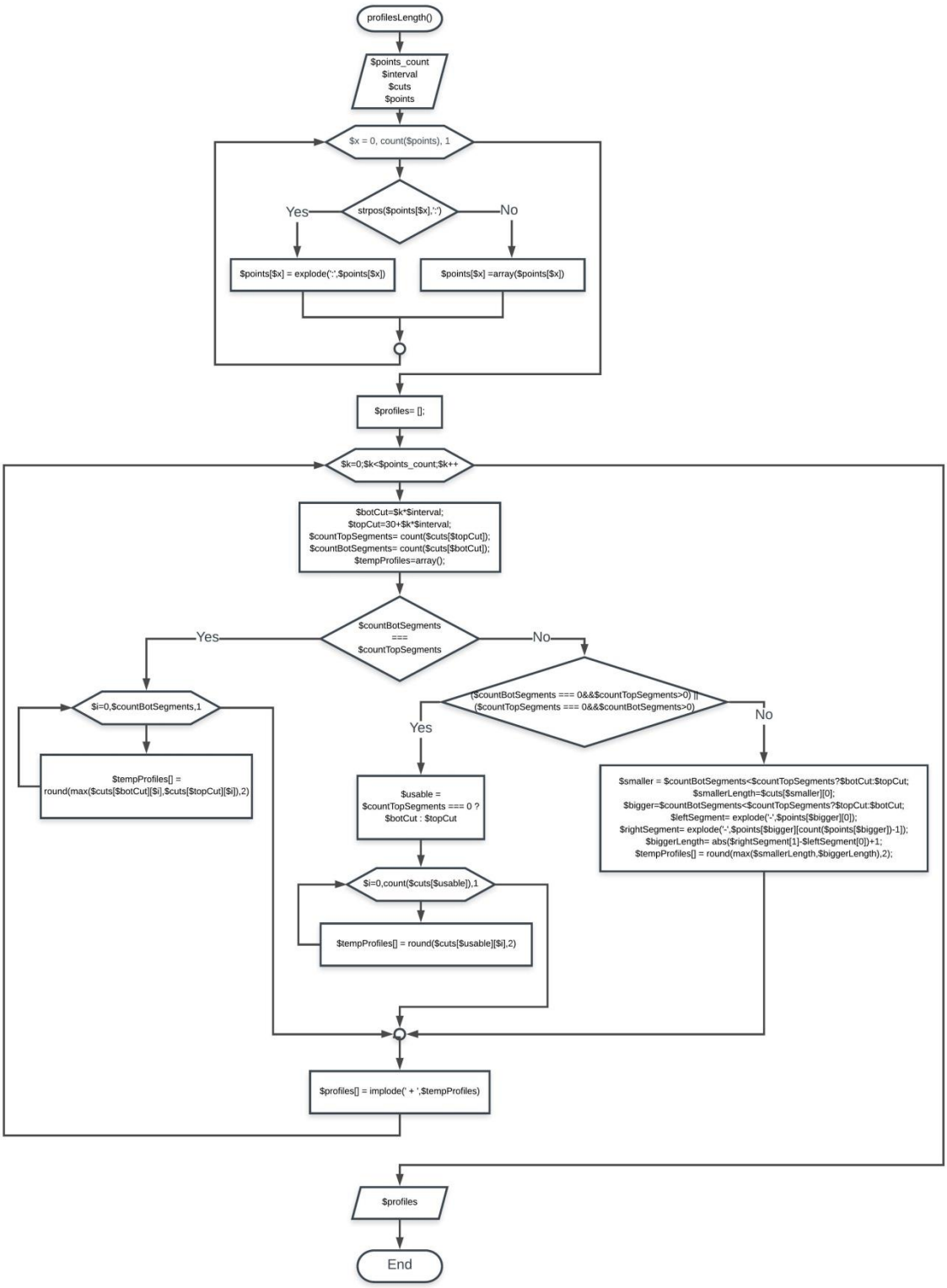



calculateMainLinks()
$links
$profiles
$step
$points
$x = 0, count($points), 1
strpos($points[$x],':')
Yes
No
$points[$x] = explode(':',$points[$x])
$points[$x] =array($points[$x])
$temp= [];
foreach($links as $link)
$linkExtended = $link;
$k = 0, $profiles, 1
$coord5mm=$k*$step+5;
$coord25mm=$k*$step+25;
$segments5mm= count($points[$coord5mm]);
$segments25mm= count($points[$coord25mm]);
$segments5mm == $segments25mm
Yes
No
$m = 0, $segments5mm, 1
$delimiters5mm = explode('-',$points[$coord5mm][$m]);
$delimiters25mm = explode('-',$points[$coord25mm][$m]);
$linkExtended-$delimiters5mm[0] > 5 &&
$delimiters5mm[1] -$linkExtended > 5 &&
$linkExtended-$delimiters25mm[0] > 5 &&
$delimiters25mm[1] -$linkExtended > 5
Yes
$temp[$link][] =$k+1;
No
$main_links= [];
foreach($temp as $key=>$link)
count($link) >=2
Yes
$main_links[] = $key.'-'.$link[0] .'-':$link[count($link)-1]
.'-'. ((count($link)-1) * {step})
No
implode('||',$main_links)
End


getImageData()
$id
$temp = [];
$url = 'cropped_'.$id.'.jpg';
$image= @imagecreatefromjpeg($url);
$image
Yes
No
$imgWidth= imagesx($image);
$imgHeight= imagesy($image);
$lineColors= [];
i = 0, 30, 1
$lineColors[] =
imagecolorallocate($image,$i,$i,$i);
ini_set('max_execution_time',300);
ini_set('memory_limit','-1');
Text
$arrayLeadingX= [];
$arrayLeadingY= [];
$points= [];
$cuts= [];
getImageY()
getImageY()
ksort($arrayLeadingY,SORT_NUMERIC);
foreach($arrayLeadingY as $y=>$value)
foreach($arrayLeadingY as $y=>$value)
$arrayLeadingX[$x][] =$y;
$i = 0, count(arrayLeadingX), 1
$arrayLeadingX[$i] = array_unique($arrayLeadingX[$i],SORT_NUMERIC);
sort($arrayLeadingX[$i],SORT_NUMERIC);
getImagePoints()
$temp['cuts'] =''.implode('|',$cuts).'';
$temp['points'] =''. implode('|',$points) .'';
$temp['width'] =$imgWidth;
$temp['length'] =$imgHeight;
imagedestroy($image);
$temp
End


GeImageX()
$y = 0, $imgHeight, 1
$xMin=0;
$xMax=0;
$x = 0, $imgWidth, 1
$colorOnPoint= imagecolorat($image,$x,$y);
No
in_array($colorOnPoint,$lineColors)
Yes
$xMin=$x;
$x = $imgWidth-1, 0, -1
No
$colorOnPoint= imagecolorat($image,$x,$y);
in_array($colorOnPoint,$lineColors)
Yes
$xMax=$x;
$arrayLeadingY[$y][] =$xMin;
$arrayLeadingY[$y][] =$xMax;
End


GeImageY()
$x = 0, $imgWidth, 1
$yMin=0;
$yMax=0;
$x = 0, $imgHeight, 1
$colorOnPoint= imagecolorat($image,$x,$y);
No
in_array($colorOnPoint,$lineColors)
Yes
$yMin=$y;
$y = $imgHeight-1, 0, -1
No
$colorOnPoint= imagecolorat($image,$x,$y);
in_array($colorOnPoint,$lineColors)
Yes
$yMax=$y;
$arrayLeadingY[$yMin][] =$x;
$arrayLeadingY[$yMax][] =$x;
End


profilesLength()
$points_count
$interval
$cuts
$points
$x = 0, count($points), 1
strpos($points[$x],'-')
Yes
No
$points[$x] = explode('-',$points[$x])
$points[$x] =array($points[$x])
$profiles= [];
$k=0;$k<$points_count;$k++
$botCut=$k*$interval;
$topCut=30+$k*$interval;
$countTopSegments= count($cuts[$topCut]);
$countBotSegments= count($cuts[$botCut]);
$tempProfiles=array();
$countBotSegments
===
$countTopSegments
Yes
No
$i=0,$countBotSegments,1
$tempProfiles[] =
round(max($cuts[$botCut][$i],$cuts[$topCut][$i]),2)
($countBotSegments === 0&&$countTopSegments>0) ||
($countTopSegments === 0&&$countBotSegments>0)
Yes
No
$usable =
$countTopSegments === 0 ?
$botCut : $topCut
$i=0,count($cuts[$usable]),1
$tempProfiles[] = round($cuts[$usable][$i],2)
$smaller = $countBotSegments<$countTopSegments?$botCut:$topCut;
$smallerLength=$cuts[$smaller][0];
$bigger=$countBotSegments<$countTopSegments?$topCut:$botCut;
$leftSegment= explode('-',$points[$bigger][0]);
$rightSegment= explode('-',$points[$bigger][count($points[$bigger])-1]);
$biggerLength= abs($rightSegment[1]-$leftSegment[0])+1;
$tempProfiles[] = round(max($smallerLength,$biggerLength),2);
$profiles[] = implode(' + ',$tempProfiles)
$profiles
End