ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИШИЙ НАВЧАЛЬНИЙ ЗАКЛАД

«МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»»

Інститут комп’ютерно-інформаційних технологій

Кафедра комп’ютерних інформаційних систем і технологій

**КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА**

**Тема «Розробка інформаційної системи для збору, обробки та аналізу метеоданих»**

Виконав студент групи T16-9-17-Б1ППі(4.0д) Носов М.Б.

Керівник: к.т.н., доцент Рябий М.О.

Рецензент: к.т.н., доцент Рудніченко М.Д.

Допущено до захисту

Завідувач кафедри \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ д.е.н., професор Кавун С.В.

(підпис)

Київ, 2021

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 71 с., 24 рис., 3 додатки, 30 джерел.

МЕТЕОРОЛОГІЯ, ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗАЦІЯ, POSTGRESQL, PYTHON, DJANGO, WEB–САЙТ.

Об’єктом дослідження є сучасні інформаційні системи, які використовуються в метеорології, та дозволяють автоматизувати робочі процеси, виконувати впровадження цифрових рішень до робочих процесів, адаптувати інформаційну систему для впровадження в різні напрямки метеорологічної діяльності.

Метою роботи є розробка інформаційної системи для збору, обробки та аналізу метеоданих. Реалізація такої системи дозволить спростити роботу синоптиків та метеорологів, а також зможе поширити інформацію про метеостан територій та прискорити аналіз доступної метеоінформації.

Для розробки використовуеться мова розробки Python включно з додатковим фреймворком для web-розробки Django, а також система керування базою даних PostgreSQL.

Результатом роботи є розробка інформаційної системи для збору, обробки та агалізу метеоданих.

METEOROLOGY, INFORMATION SYSTEM, AUTOMATION, DOCUMENTATION, SQLITE, PYTHON, DJANGO, WEB SITE.

The object of research is modern information systems used in meteorology, and allow to automate work processes, implement digital solutions to work processes, adapt the information system for implementation in various areas of meteorological activities.

The purpose of the thesis is to develop an information system for the collection, processing and analysis of meteorological data. The implementation of such a system will simplify the work of weather forecasters and meteorologists, as well as be able to disseminate information about the meteorological conditions and speed up the analysis of available meteorological information.

The Python development language is used, including an additional framework for Django web development, the PostgreSQL database management system.

The result of the thesis is the development of an information system for the collection, processing and analysis of meteorological data.

ЗМІСТ

[ВСТУП 4](#_Toc72613795)

[1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ЗБОРУ, ОБРОБКИ ТА АНАЛІЗУ МЕТЕОДАНИХ 7](#_Toc72613796)

[1.1 Опис предметної області 7](#_Toc72613797)

[1.2 Аналіз впливу інформаційних технологій на робочі процеси метеорологів 11](#_Toc72613798)

[1.3 Впровадження інформаційних систем в сферу метеорології 15](#_Toc72613799)

[1.4 Висновки до першого розділу 19](#_Toc72613800)

[2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНИХ СИСТЕМ ТА СХОЖИХ ПРОГРАМНИХ ПРОДУКТІВ 21](#_Toc72613801)

[2.1 Аналіз схожих програмних продуктів 21](#_Toc72613802)

[2.2 Аналіз засобів програмної реалізації інформаційних систем 28](#_Toc72613803)

[2.3 Висновки до другого розділу 43](#_Toc72613804)

[3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ЗБОРУ, ОБРОБКИ ТА АНАЛІЗУ МЕТЕОДАНИХ 44](#_Toc72613805)

[3.1 Проектування розробки інформаційної системи 44](#_Toc72613808)

[3.2 Опис розробленої бази даних інформаційної системи 53](#_Toc72613809)

[3.3 Опис розробленого інтерфейсу до інформаційної системи 59](#_Toc72613810)

[3.4 Практичне використання інформаційної системи 65](#_Toc72613811)

[3.5 Висновки до третього розділу 67](#_Toc72613812)

[ВИСНОВКИ 68](#_Toc72613813)

[ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 69](#_Toc72613814)

[ДОДАТОК А 72](#_Toc72613815)

[ДОДАТОК Б 93](#_Toc72613816)

[ДОДАТОК В 99](#_Toc72613817)

# ВСТУП

Інформаційні технології активно використовуються в метеорології і роботі синоптиків, так як дозволяють не тільки автоматизувати їх роботу, а також підвищити зручність її виконання і значно підвищити точність результату.

Метеорологія - це наука, яка вивчає процеси, явища, що відбуваються в атмосфері.

Сучасна метеорологія не змогла б розвинутися без впровадження в неї інформаційних технологи і особливо суперкомп'ютерів, пов'язано це з тим що робота синоптиків, зокрема побудова прогнозу погоди, вимагає великих як тимчасових так і людських ресурсів. Починаючи від простого побудови прогнозу погоди закінчуючи попередженням про насування погодних катастроф, які могли б привести до великих людських жертв і масштабних руйнувань.

Використання обчислювальної техніки і відповідних програм дозволяє вирішувати завдання метеорології з великою точністю, для значно більших територій, при цьому за більш короткий термін і з залученням менших людських ресурсів. У зв'язку з цим дана галузь вкрай сильно потребує розробки зручних і високопродуктивних програмних засобах.

Тема більш ніж актуальна так як розвинена метеорологія дозволить поліпшити важливі аспекти життя людей, наприклад: збільшити ефективність сільськогосподарської галузі, дозволить визначати безпечні для життя людей місця, прокладати більш оптимальні маршрути кораблям і літакам, мінімізувати наслідки природних катастроф.

Зв'язок роботи з програмами наукових досліджень кафедри ПІ. Для поліпшення загальних процесів автоматизації та створення єдиного інформаційного простору для сфери метеорології повинні використовуватися сучасні інформаційні технології, які дозволяють розробити сучасний програмний засіб, здатний виконати поставлене завдання за менший проміжок часу і забезпечити більш високу точність, ніж людина. А завдяки отриманим знанням з програмування, алгоритмізації та роботі з базами даних, розроблений програмний засіб може використовуватися з навчальною метою.

Мета і задачі дослідження. Метою роботи є розробка інформаційної системи для збору, обробки та аналізу метеоданих. Реалізація такої системи дозволить спростити роботу метеорологів та синоптиків, а також зможе надати вільній доступ до метеоданих. Розроблена інформаційна система повинна бути зі зручним інтерфейсом який надасть користувачеві комфорту в роботі в ним. Для виконання поставленої задачі, необхідно виконати:

* виконати аналіз літературних джерел;
* виконати аналіз предметної області;
* проаналізувати вплив інформаційних технологій на робочі процеси метеорології;
* обґрунтувати впровадження інформаційних систем в метеорологію;
* виконати порівняльний аналіз схожих програмних систем;
* визначити програмні засоби реалізації інформаційної системи;
* виконати проектування інформаційної системи;
* виконати розробку інформаційної системи для метеорологів.

Об’єкт дослідження. Об’єктом дослідження є сучасні інформаційні системи, які використовуються для роботи метеорологів, та дозволяють автоматизувати робочі процеси, виконати впровадження цифрових рішень до робочих процесів, адаптувати інформаційну систему для впровадження в різні напрямки метеорологічної діяльності.

Пріоритетними завданнями аналізу інформаційної складової в програмних системах, які інтегруються в системи обробки метеоданих є засоби аналітичного контролю за ефективністю роботи закладу та високого коефіцієнту якості, що включає в себе виконання наступних етапів:

* аналіз актуальних методів для прогнозування погоди;
* виявлення найточнішого метода аналізу метеоданих;
* визначення основних параметрів які необхідні для прогнозування погоди;
* створення універсальної системи збору метеоданих.

Методи дослідження. При виконанні роботи, були використано аналіз схожих інформаційних систем, методи об’єктно-орієнтованого програмування, машиного навчання, а також підходи до вирішення сучасних проблем метеорології.

Практичне значення одержаних результатів. Розроблена інформаційна система може використовуватися в якості системи автоматизації робочих процесів метеорологічних центрів, а також може виступати в якості початкової системи прогнозування погоди. Головною особистістю такої інформаційної системи є її простота використання та гнучкість до джерел данних.

Результатом розробки є інформаційна система для метеорологічних центрів, яка дозволяє робити автоматичний короткосрочний прогноз погоди та проводити аналіз метеоданих завдяки інструментам створення графіків.

# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ЗБОРУ, ОБРОБКИ ТА АНАЛІЗУ МЕТЕОДАНИХ

* 1. Опис предметної області

Метеорологія є комплексом наук все атмосферні явища, такі як: погода, клімат, хмари, склад атмосфери, її будова, тепловий режим і влагообмін, оптичні та акустичні, поведінка фронтів, вітру і електричного поля. Всі ці дослідження допомагають знайти і зрозуміти фізичні залежності які відбуваються в атмосфері, що в кінцевому рахунку наблизить людство до розуміння життєвих циклів планети.

Сучасних метеорологів можна розділити по виду діяльності на наступні категорії або наукові напрямки:

* Фізична метеорологія – займаються розробкою радіолокаційних і космічних методів дослідження атмосферних явищ.
* Динамічна метеорологія – вивчення фізичних механізмів атмосферних процесів.
* Синоптична метеорологія – наука про закономірності зміни погоди.
* Кліматологія – наука вивчаюча клімат як сукупність погодних характеристик за багаторічний період і їх зміна властивих певної території.
* Аерологія – наука вивчає верхні шари атмосфери для декількох десятків кілометрів від поверхні землі.

Прикладні напрямки сучасної метеорології:

* Авіаційна метеорологія – наука вивчає вплив погоди на діяльність авіації.
* Агрометеорологія – наука яка вивчає вплив погоди на сільське господарство.
* Біометеорологія – наука вивчає вплив атмосферних процесів на людину та інші живі організми.
* Ядерна метеорологія – наука вивчає природну і штучну радіоактивність, поширення в атмосфері радіоактивних домішок, вплив ядерних вибухів.
* Радіометеорологія – наука вивчає поширення радіохвиль в атмосфері.

Якщо узагальнити те кожен розділ займається пошуком деяких погодних залежностей і вивчення впливу погодних явище на деяку галузь. Для виконання цієї роботи необхідно взаємодіяти злагоджено і задіяти безліч фахівців по всьому світу, які кожні три години збирають інформацію про погоду зі всіх доступних джерел, далі проводять їх обробку і будують різні карти описують графічним методом поточний погодний стан атмосфери в різних регіонах. Далі за отриманими картками проводиться аналіз виявляє необхідну інформацію, наприклад це може бути аналіз погодних аномалій для розуміння того чи можуть вони перерости в природні катастрофи і якщо це так варто попередити населення цих регіонів.

Метеорологічні данні які найчастіше збирають для досліджень [3]:

* Місце та час коли були зняті метеодані, базова інформація яка потрібна для прив’язки метеоданих на карті.
* Висота хмар від рівня моря.
* Дальність видимості. Кількість метрів далі котрих не можна побачити наземні об’єкти.
* Вид хмар, якій преобладає в даному секторі.
* Напрям вітру.
* Швидкість вітру.
* Температура повітря в час збору даних.
* Точка роси, це температура повітря при якій починає випадати роса.
* Атмосферний тиск на рівні метеостанції.
* Атмосферний тиск на рівні моря.
* Максимальна та мінімальна температура за добу.
* Погодне явище.
* Кількість опадів.

На базі цих метеоданих будуються прогнози погоди та більшості з усіх метеорологічних розрахунків.

Основою для прогнозу погоди є облік періодичних і неперіодичних змін метеорологічних величин і явищ погоди [4]. Періодичні зміни тієї чи іншої метеорологічної величини обумовлюються добовим і річним ходом цієї величини, неперіодичні – еволюцією і переміщенням синоптичних об'єктів: циклонів і антициклонів, повітряних мас і атмосферних фронтів. Ось чому прогнозом погоди завжди передує прогноз синоптичного положення. Найбільшу трудність і практичний інтерес представляє облік імені не періодичних змін.

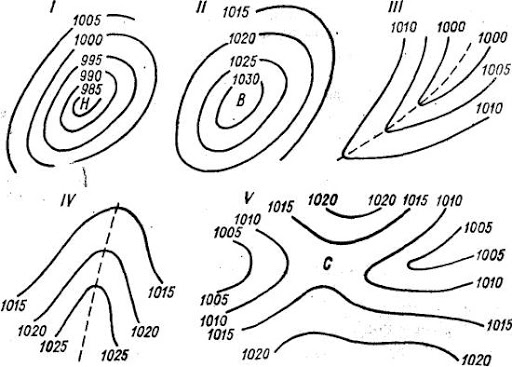


Рисунок 1.1 – Приклад відображення баричних систем на картах.

Синоптичний метод в даний час є основним при розробці довгострокового прогнозу погоди.

Суть методу в тому, що на підставі аналізу карт погоди за кілька послідовних термінів складають прогноз синоптичного положення, який заклю чає в прогнозі виникнення, переміщення і еволюції повітряних мас, атмосферних фронтів, баричних систем, приклад відображення баричних систем на картах рис 1.1.

Карта, на яку наносять передбачуване положення синоптичних об'єктів, називається прогностичною, рис 1.2.

Прогностичну карту складають на певний момент часу наступної доби, а іноді на двоє або більше доби вперед.

На підставі прогностичної карти складають прогноз погоди в тому чи іншому районі. Причому прогноз погоди є логічним продовженням прогнозу синоптичного положення і виходить з основного принципу, що полягає в припущенні, що з переміщенням і еволюцією синоптичних об'єктів переносяться з певними змінами і властиві їм умови погоди.

Тому за прогностичне значення метеорологічної величини в першому наближенні приймаються їх значення в районі, звідки очікується переміщення синоптичного об'єкту, в район, для якого складається прогноз погоди. При такому великому обсязі роботи та даних необхідних до зберігання дана галузь не може обійтися без використання інформаційних технологій. Особливо новітніх програмних засобів, які би спростили роботу з розрахунками та потужних комп’ютерів, які би ці алгоритми оброблювали.

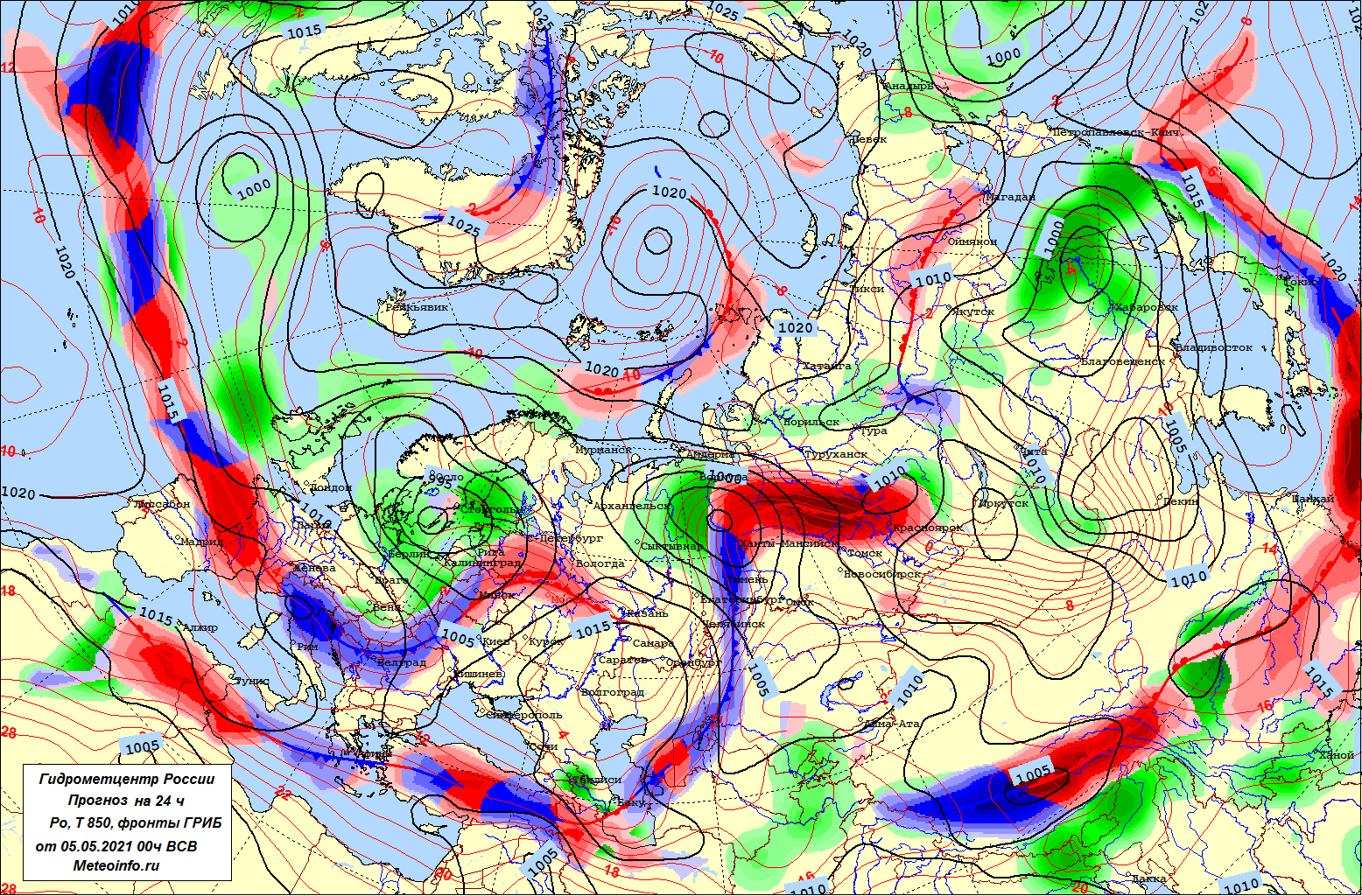


Рисунок 1.2 – Приклад прогностичної метеорологічної карти

* 1. Аналіз впливу інформаційних технологій на робочі процеси метеорологів

Інформаційні технології щільно увійшли і намертво закріпилися в житті сучасних метеорологів.

У своїй щоденній роботі їм необхідно використовувати масивні програмні комплекси які виконують завдання збору, зберігання і перегляду інформації з джерел метеоданих, програми для побудови погодних карт, для визначення і відстеження баричних систем і атмосферних фронтів, побудови прогностичних карт, виявлення погодних аномалій, аналізу погодних даних, побудови графіків залежностей, отримання статистики і графічного відображення метеоданих, а також прогнозування катастроф і інші.

Вплив інформаційних технологій на галузь досить велике, так як без них виконання більшості завдань сучасної метеорології було б неможливим. Якби метеорологи використовували старі або "класичні" методи, то всьому людству довелося б освоїти цю професію і спільно виконувати її завдання. Сам процес збору даних залучає більше кількість людей, а її аналіз зажадав би все доступне в добі час.

Для вирішення та прискорення цих процесів людство прийшло до використання суперкомп'ютерів [5].

Суперкомп'ютером називається комплекс електронних обчислювальних машин працюють як одне ціле, що дає всій системі велику обчислювальну потужність.

Однак навіть суперкомп'ютерів необхідно достатньо часу для розрахунків еволюції баричних систем і прогнозу погоди.

Наприклад, для розрахунку короткострокового прогнозу погоди для всієї території України, з точністю в п'ять квадратних кілометрів, середньостатистичному суперкомп'ютера знадобиться близько години.

На даний момент процес побудови прогнозу запускається не частіше ніж раз на три години, цього цілком достатньо для більшості територій.

Винятками ж є території з високою ймовірністю появи атмосферних катаклізмів. Як приклад таких областей можна привести

Японію, з їх штормами і цунамі, південні штати Америки, з місцевими повенями і торнадо і т.д.

З кожним днем ​​необхідних метеорологічних розрахунків стає все більше, через те що все більше галузей виявилися залежні від інформації про стан атмосфери або про її вплив на свою галузь і отже отримали необхідність в аналізі метеоданих.

Завдяки цій тенденції сильно збільшилося фінансування метеорології що в кінцевому рахунку дозволило метеорологам почати безліч досліджень, які в майбутньому допоможуть краще розуміти процеси атмосфери і їх вплив на наше життя.

Основна частина фінансової підтримки необхідна для придбання обладнання та розробки програмного забезпечення для всіх галузей даної сфери.

При активному розвитку інформаційних технологій в сфері метеорології для людства можуть відкритися можливості:

* + Ідеальна організація і оптимальний розподіл вирощуваних культур. При точному знанні того які метео-параметри були раніше і які будуть в кожній точці світу люди зможуть розподілити висаджуються культури ідеально по необхідним умовам. Дане нововведення дозволить збільшити обсяги вирощуваних культур, але також пристосуватися до змін клімату і вчасно реагувати на випадки коли земля стає менш придатною для вирощування конкретної культури.
  + Оптимізація і полегшення маршрутів суден і літаків. Дане нововведення дозволяє запобігти небезпеці в дорозі і потенційно врятували б життя людей, мінімізували б витрати на ремонтні роботи транспорту, а також не допустити забруднення природи від можливих аварій.
  + Можливість побудови різних архітектурних споруд. Для побудови різних споруд важливо знати які погодні умови переважають в районі будівлі щоб зрозуміти як сильно треба зміцнити конструкцію будівлі для її стійкості.
  + Ремонтні роботи. Частково відноситься в попередньому пункту, для деяких ремонтних робіт важливі певні погодні умови з чим і можуть допомогти нові системні комплекси метеорологів, точно вибравши проміжок часу з ідеальною погодою для ремонтних робіт.

Не менш значущий вплив інформаційні технології привнесли в системи для вимірювання метеоданих.

Раніше використовувані аналогові прилади, рис 1.3, хоч і відрізнялися дешевизною і примітивністю в роботі з ними, проте були недостатньо точними і досить громіздкими.



Рисунок 1.3 – Класична метеостанція обладнана аналоговою системою вимірювання

Сучасні прилади компактні, мають можливість передавати результати вимірювань по лініях зв'язку і при цьому мають велику точність. Це дозволяє прискорити процес збору даних, а їх автономність дає можливість вільно масштабувати мережу метеостанцій, рис 1.4.

Подібні системи мають основний стовп на якому закріплені різні датчики, наприклад датчики напряму вітру, кількості опадів, атмосферного тиску та різні датчики для аналізу повітря.

Система автономного живлення представляє собою джерело енергії, систему стабілізації енергії та систему накопичування енергії.

Як джерело енергії може бути сонячні панелі чи вітряки. Система накопичування енергії це акумуляторна батарея яка буди живити станцію в час коли енергії системи живлення не буде достатньо енергії [5].



Рисунок 1.4. – Приклад сучасної метеостанції та функцій які вона може виконувати

* 1. Впровадження інформаційних систем в сферу метеорології

Впровадження інформаційних систем в сферу метеорології може бути проведено в декількох напрямах:

* Інформування людей та навчання спеціалістів на нових методологіях.
* Популяризація метеорологічних досліджень серед населення.
* Розробка професійних програмних засобів та стандартів які допомогли би розробляти програмні системи для метеорологічних комплексів.

Інформування людей може зацікавити та посприяти кількості нових спеціалісті в сфері метеорології. Також навчання фахівців на сучасні технології, потенційно, може полегшити процес входу в професію, а також оновлені навчальні матеріали наддадуть найсвіжіші знання молодим фахівцям.



Рисунок 1.5 – Приклад метеостанції домашнього типу

Додаткова популяризація метеорології серед решти населення може посприяти розвитку з допоможи ентузіастів або навчальних центрів. Метеорологія є ідеальним напрямком для хобі, ентузіасти не раз допомагали світовій спільноті в розробці тестуванні і підтримки програмних засобів. Даний фактор міг би внести чималий внесок в розвиток сучасної метеорології. Також для збільшення мережі метеостанцій можна розробити її простий і домашній варіант, такі станції можуть використовуватися для збору даних в великих містах або місцях де неможливо встановити повноцінну систему [5]. Подібні станції сильно відрізняються від професійних аналогів але можуть збирати дані про температуру, вологість повітря і атмосферний тиск, рис 1.5. Деякі моделі дозволяють аналізувати якість повітря, конкретно його склад і кількість домішок. Питання відстеження та аналізу стану повітря не менш важливий для метеорології. У повітрі можуть міститися різні домішки які можуть той чи інший ефект на погоду в регіоні. Таким чином вивчаючи склад повітря можна в майбутньому передбачити згубні явища в регіоні по зміні складу його повітря і навпаки, припустити склад повітря через погодні явища в регіоні, рис 1.6.

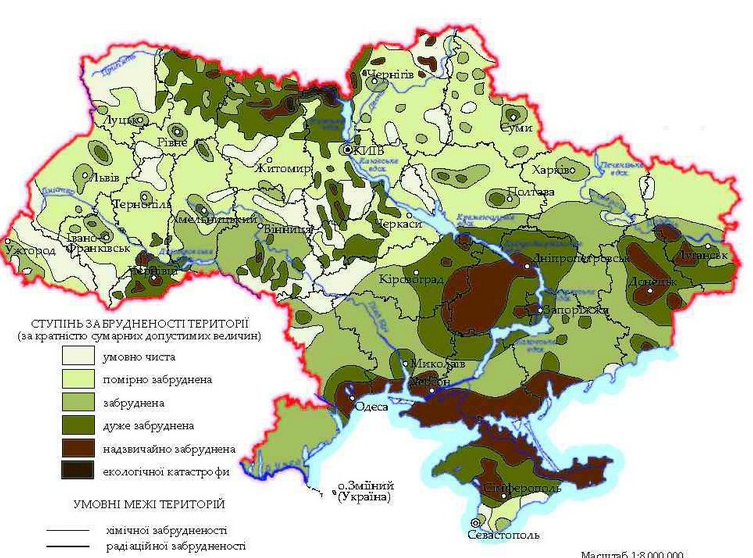


Рисунок 1.6 – Карта забруднення повітря і грунту України

Факторами зміни можуть бути, як досить очевидні, наприклад заводи або аварії, так і менш очевидні і внаслідок чого більш небезпечні, наприклад розкриття покладів метану внаслідок землетрусів. Аналіз складу повітря більш актуальний для другого випадку, конкретно для систем запобігання катастроф. Система вивчає аномалії в повітрі і може виявити виниклі проблеми, після чого попередити про це відповідні служби. Існують приклади місць які вимагають постійного моніторингу ситуації, наприклад озера в парку Йеллоустоун, які є потенційно небезпечною зоною і можуть завдати серйозної шкоди регіону. Також подібні системи фахівці можуть відслідковувати так звані «мертві зони» в морях і океанах, це зони в яких сталася якась аномалія в слідстві якої живі організми в ній вимерли або покинули її, рис. 1.7. Найчастіше подібний ефект викликаний забрудненнями вод і подібні місця легко відслідковуються за складом повітря.

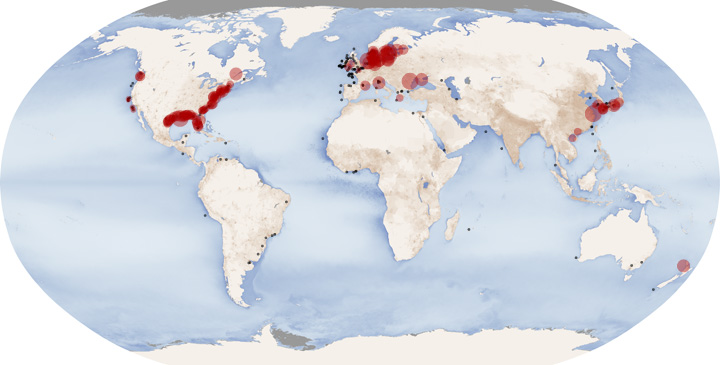


Рисунок 1.7 – Карта світового океану із зазначеними на ній мертвими зонами

Ефективність розвитку сфери сильно залежить від стандартизації її елементів, наприклад розробки визначених стандартів для метеостанцій, що дозволить максимально спростити процес розробки ПО для них, вибрати найоптимальніші алгоритми роботи [9]. Створення специфічних програмних бібліотек і фреймворків дозволить полегшити написання програм, так як базові алгоритми будуть вже готові, це знизить шанс помилки і збільшить швидкість розробки шляхом перевикористаня коду.

Для ефективного виконання завдань метеорології використане програмне забезпечення повинно мати наступні властивості:

* Швидкодія - так як даних які потрібно обробити дуже багато і сам розрахунок прогнозу досить складний алгоритми використовуються в програмі повинні бути дуже швидкими, як і інші функції програми, інакше необхідний розрахунок ризикує не бути проведені під час [11]. Також чим швидше він буде проведе тим менше енергії буде витрачено, в разі суперкомп'ютерів ця характеристика вкрай важлива, так як їм необхідно вкрай велика кількість енергії для роботи.
* Зручність використання - чим простіше і зрозуміліше для користувача інтерфейс програми тим легше йому буде з ним працювати і тим швидше будуть виконуватися його повсякденна робота. Також якщо програмне забезпечення буде доступно на всіх платформах дозволяє користувачеві виконувати свої завдання не залежати від оточуючих його пристроїв. З кожним днем ​​життя людей стає все динамічніше і багато компаній дозволяють своїм співробітникам виконувати свою роботу поза офісом. Професія метеоролога може не бути винятком і в майбутньому її представники зможуть виконувати свої обов'язки в процесі подорожі по світу [12].
* Точність – з урахуванням того що обчислення робить програма а не людина то було б дуже корисно максимально підвищити точність прогнозів [6]. З підвищенням точність рішення прийняті на основі обчислень матимуть потенційно велику ефективність.
* Простота налаштування – чим легше буде встановити програмний продукт тим більше метеоцентри можна відкрити і тим більше можна буде заощадити на послугах системного адміністратора який буде цю систему обслуговувати [12].

Однією з найважливіших елементів в сфері метеорології є самі метеостанції, на даний момент, в більшості випадків, вони представляють собою майданчик з різним набором аналогових детекторів. При переосмисленні метеостанції в вид повністю електронної системи розробляється програмний продукт повинен відповідати наступним вимогам:

* Швидкодія – чим швидше буде проходити передача даних тим швидше вони будуть оброблені в метеоцентрі, також ця характеристика говорить про те що система зможе передати більшу кількість даних [9].
* Малий розмір – чим менше місця в пам'яті пристрою буде займати програмний продукт тим більше можна заощадити на пам'яті пристрою що в цілому дозволить спростити сам пристрій і зробити його дешевше [10].
* Автономність – цей критерій застосовний як до програмного забезпечення так і до самого приладу. Реалізується фактор автономності оптимальними алгоритмами передачі даних і наявністю систем отримання і накопичення енергії в пристрої. Це дозволяє розміщувати станції в слабо підготовлених місцях, а самі станції не вимагають обслуговування.
* Дальність і стабільність зв'язку – чим краще буде реалізований протокол передачі даних тим на більшій відстані можна розміщувати самі станції. А чим стабільніше зв'язок тим більше автономне буде сама система [13].
  1. Висновки до першого розділу

Результатом проведеного аналізу предметної області, було визначено що сучасна метеорологія є добрим напрямком для впровадження сучасних інформаційних технологій [5]. Це пов’язано з актуальним використанням засобів комунікації та інформаційного обміну між метеостанціями та метеоцентрами [7].

Сучасні засоби автоматизації робочих процесів будуються таким чином, що можливо суттєво автоматизувати робочу діяльність метеорологів, а також покращити засоби обробки метеоданих, впровадити нові системи для роботи з ними, а також покращити вже використані алгоритми та методи які спрощують роботу.

Інтеграція нових програмних рішень повинна бути повсюдно, в усіх метеоцентрах та метеостанціях. Усі метеостанції повинні бути замінені на повністю автономні.

Важливим фактором є автономність системи. Також багато уваги потрібно виділити на універсальність системи, а саме на можливість системи проводити дослідження для всіх сфер діяльності людини які залежать від метеорології [8].

Виходячи з цього, необхідно проаналізувати сучасні програмні рішення серед метеорологічних інформаційних та програмних систем, а також проаналізувати сучасні засоби розробки програмного забезпечення. Після чого, можливо остаточно визначити необхідний набір технічних та програмних рішень для реалізації інформаційної системи.

# АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНИХ СИСТЕМ ТА СХОЖИХ ПРОГРАМНИХ ПРОДУКТІВ

2.1 Аналіз схожих програмних продуктів

З метою визначення усіх функціональних можливостей інформаційних систем для метеорологів необхідно проаналізувати існуючі рішення та сформувати головні вимоги до таких систем. Для об’єктивного порівняння, необхідно використовувати тільки web-рішення, які автоматизують робочі задачі метеорологів, або спрощують роботу метеорологічного центру.

Першим програмним продуктом для аналізу буде «WRF model». Даний програмний продукт це система прогнозування погоди, призначена для використання як у дослідженнях атмосфери, так і у високоточних прогнозах погоди. NWP спирається на моделювання та прогнозування погоди за допомогою комп'ютерної моделі, а WRF — це набір програмного забезпечення для цього. WRF оснащено двома динамічними (обчислювальними) ядрами (або *рішеннями*), системою засвоєння даних та архітектурою програмного забезпечення, що дозволяє паралельно обчислювати та розширювати систему. Ця модель обслуговує широкий спектр метеорологічних задач у масштабах від кількох метрів до тисяч кілометрів.

WRF дозволяє дослідникам обробляти реальні дані спостережень і на їхній основі розробляти передбачення погодних умов. WRF забезпечує швидке прогнозування. WRF використовується в NCEP та інших центрах прогнозування на міжнародному рівні. WRF перетворився на велику світову спільноту користувачів, з понад 30.000 зареєстрованих користувачів у 150 країнах, а семінари та навчальні програми проводяться щороку в NCAR. WRF широко використовується для досліджень та прогнозування в реальному часі в усьому світі. як у дослідженнях атмосфери, так прогнозах погоди WRF пропонує два динамічні вирішення для обчислення рівнянь, що управляють атмосферою, а варіанти моделі відомі як WRF-ARW і WRF-NMM негідростатична мезомасштабна модель. WRF слугує основою для моделей RAP та HRRR: оперативні прогнозні моделі високої роздільної здатності постійно працюють на NCEP. Версія WRF-NMM, розроблена для прогнозування урагану, HWRF, дослідження урагану та прогнозування погоди, почала діяти 2007 року. Система реалізована у вигляді набору з підпрограм, кожна з яких виконує свою невелику задачу. Сам процес роботи з цим програмним комплексом є запуск його підпрограм в правильному порядку з різними параметрами запуску. Інтерфейс роботи з програмою є консольним, а висновок програми це спливаючі вікна з графіками [14]. Даний програмний продукт є продуктом з відкритим вихідним кодом, проте написаний спеціалізоване для ОС Windows. У його розробці використовувалися мови програмування Fortran, Roff, C++, C і Assembly [6]. У зв'язку з складністю і рідкістю цих технологій розробку можуть потримати малу кількість фахівців і така розробка займає багато часу. Приклад результуючих графіків представлений на рис. 2.1.

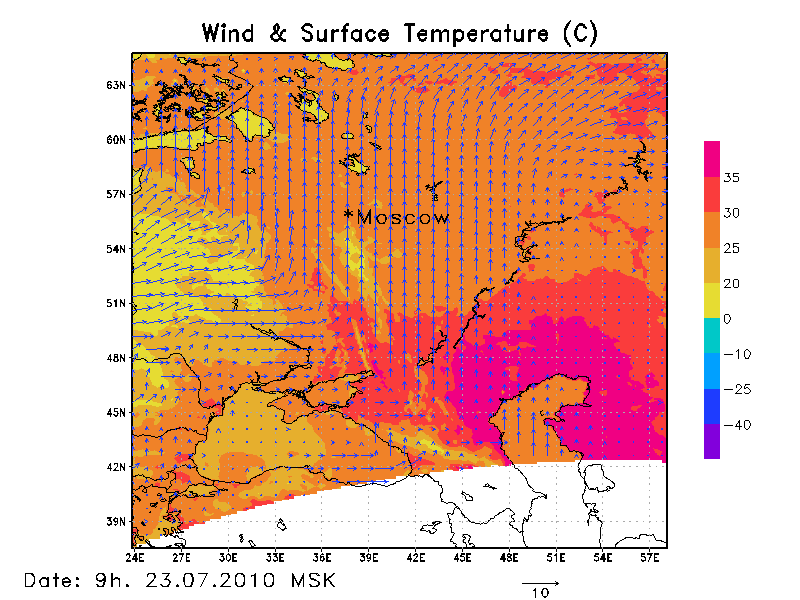


Рисунок 2.1 – Вікно головної сторінки інформаційної системи «Поехали с нами»

Основні переваги даної системи:

* Багатий функціонал.
* Висока точність.
* Простота навчання для роботи з системою.
* Вартість.

Недоліки даної системи:

* Швидкість розробки.
* Складність розширення функціоналу.
* Незручність інтерфейсу.
* Швидкість роботи.
* Платформні обмеження.
* Відсутність системи отримання даних.
* Відсутність системи роботи з даними.

Висновок з аналізу наступний, даний програмний комплекс хоч і є мультинаціональним проте сильно застаріло і є абсолютно непридатним в сучасній професійній метеорології.

Відсутність системи збору метеоданих це критичний недолік для майбутнього метеорології, який не дасть їй можливості розвиватися.

Іншим програмним комплексом для метеорологів є «Система краткосрочного прогноза погоды и предупреждения о чрезвычайных погодных условиях - Hazardous Weather Warning System (HWWS)»

Розробником даного продукту є компанія AUX Link Group (бренд MeteOSS) яка завоювала репутацію в області поставок і інтеграції метеорологічного обладнання, експлуатації багаторівневих метеорологічних систем, створення та обслуговування систем метеорологічного обліку та інвентаризації, розробки навчальних і методичних матеріалів з навчання, технічного обслуговування і ремонту метеорологічного обладнання, розробки і впровадження систем підтримки прийняття рішень і систем довгострокового і короткострокового прогнозування погоди, створення індивідуальних систем отримання і відображення метеорологічних даних і забезпечення високоякісних аудиторських і консалтингових послуг.

Клієнтам компанії надається комплексне оснащення і постачання різного метеорологічного обладнання, систем раннього попередження про надзвичайні погодні умови і систем довгострокового прогнозу погоди для національних метеорологічних агенств, авіації і дорожніх служб.

Розробляється нами система інвентаризації метеоролгіческого обладнання включає в себе методичні матеріали (аудіо та відео матеріали в тривимірній графіці) з технічного обслуговування і ремонту метеорологічного обладнання та його частин. Команда висококваліфікованих фахівців нашої компанії виробляє інтеграцію різних метеорологічних систем в єдиний інформаційний простір для управління всією інфраструктурою організацій, що відповідають за метеорологічне забезпечення в національних метеослужб, авіації, управліннях метеозабезпечення автомобільного та залізничного транспорту [17].

Компанія здійснює інтеграцію будь-яких типів метеорологічних інструментів в єдину систему відображення і прогнозування короткострокової і довгострокової метеорологічної ситуації, візуалізації метеоінформації в необхідному для наших Замовників вигляді і забезпечуємо наших Клієнтів системами управління, обслуговування та підтримки.

Що відносно самого програмного продукту, на відміну від систем побудови довгострокового прогнозу погоди, які забезпечують моделювання погоди на дні або систем довгострокового прогнозу погоди, де йде моделювання ситуації на тижні, система короткострокового прогнозу погоди і попередження про надзвичайні погодні умови оперує хвилинами і годинами. Аналіз погодних умов і моделювання прогнозу погоди здійснюється в інтервалі від декількох хвилин до чотирьох годин. Найбільш точний прогноз про час і кількості опадів, локалізація на місцевості і своєчасне попередження всіх відповідних міських і метеорологічних служб - основне призначення системи. Вчасно оцінити рівень небезпеки того чи іншого погодного катаклізму дозволяє вчасно попередити населення і відповідні державні служби, дозволяє мінімізувати можливі збитки, усунути загрозу людським життям, закріпити або перемістити в ангари літаки в аеропорту, обмежити рух в схильні до затоплення райони міста і вирішити багато інших завдань, які допоможуть звести до мінімуму шкоду і найбільш швидко відновити нормальну життєдіяльність суспільства після закінчення надзвичайної ситуації.

Основним завданням системи є аналіз метеорологічних даних з різних джерел і побудова короткострокового прогнозу погоди з прогнозуванням можливих надзвичайних погодних явищ і аналізі тривалості і можливий рівень небезпеки і шкоди. Гроза і сильна злива, град і шквальний вітер, сніговий буран або екстремальна температура часто формуються протягом короткого часу і часто мають відносно невеликі розміри, що не дозволяє звичайним інструментам створення прогнозу погоди точно передбачити загрозу підтоплення або руйнувань в рамках окремо взятої місцевості, такого як місто або аеропорт.

Система короткострокового прогнозу погоди і попередження про надзвичайні погодні умови аналізує метеорологічні дані в режимі реального часу в рамках заданої місцевості обмеженою відносно невеликою територією і будує прогноз виникнення надзвичайних погодних умов в рамках декількох годин. Це дозволяє надати більш точний прогноз виникнення небезпечної ситуації і також дає можливість своєчасного попередження всіх міських і державних служб, які можуть відповідно до отриманої інформації інформувати населення і підприємства, організувати дорожній рух розвантаживши дорогу надзвичайних службам і направивши транспортні потоки повз місця, які найбільш схильні до затоплення.

При наявності статистичних даних і (або) даних рельєфу місцевості, стану зливових споруд та інших даних можливе створення семантичних шарів, що відображають найбільш вразливі міські точки при тій чи іншій ситуації Так, наприклад, різні території і міські служби будуть відображатися в системі при загрозі зливи з підтопленням або в умовах різкого зниження температури та ожеледицю.

На рис. 2.2 наведено приклад відображення рівня накопичених дощових опадів в середньому по басейну над хребтом в зоні Колорадо.

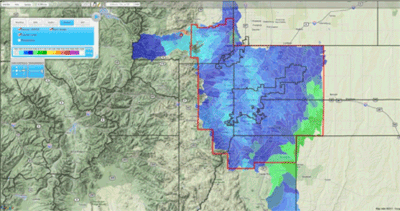


Рисунок 2.2 – Приклад відображення рівня накопичених дощових опадів

Головні переваги системи:

* Унікальність функціоналу.
* Гарний і зрозумілий інтерфейс.
* Висока швидкість роботи.

Недоліки системи:

* Вартість.
* Складність установки.
* Працює тільки з апаратного системами самої компанії.

Висновком з аналізу даного програмного продукту є те що він хоч і виконує важливу й унікальну задачу, але при це працює тільки з обладнанням самої компанії, це є критичним недоліком так як він не дозволяє вільно масштабувати мережу метеостанцій і метеоцентри [7].

Останньою системою яку варто проаналізувати є система для візуалізації метеоданих «Metview»

Metview є метеорологічну робочу станцію - програмний додаток, розроблене для того, щоб забезпечити метеоролога і дослідника необхідним і повним робочим інструментарієм. Головне призначення програмного забезпечення Metview - надати користувачеві доступ до інформації, можливості для її обробки і візуалізації. Для вирішення цих завдань розроблено як інтерактивний графічний інтерфейс, так і спеціалізований скриптова мова для пакетної обробки завдань. Розуміється можливість автоматичної генерації коду.

Слід зазначити, що графічний інтерактивний інтерфейс програм Metview досить оригінальний (раніше мені такий тип GUI не зустрічається). Всі об'єкти в ньому (дані, програми-скрипти, запити до БД, зовнішні програми, елементи або їх групи для візуалізації - осі, написи, точки, лінії, ...) представляють собою іконки, тобто візуальним поданням будь-якого об'єкта є іконка.

Завдання це послідовності операцій над іконками. Наприклад, у Вас є іконка, що представляє властивості географічної підкладки (включаючи налаштовані властивості проекції, берегової лінії, кордонів держав і т.п. із заданими товщиною ліній і квітами і т.п.), а також група іконок, що представляє дані (наприклад, поля тиску і температури на різних висотах). Для відтворення підкладки користувач вибирає з меню першої іконки метод «отрисовать». Для зображення того чи іншого поля просто мишкою перетаскуєте потрібну іконку з даними на малюнок підкладки і на ньому буде отрисовать відповідне поле. Для відтворення іншого поля «кидаєте» на малюнок іконку з іншими даними і отримаєте нове зображення і т.д. є ряд підготовлених розробниками іконок, для полегшення початку роботи з програмою.

Metview може читати дані з різних джерел і різних форматів, включаючи:

* GRIB файли (як версії 1, так і версії 2).
* BUFR файли.
* MARS (ECMWF метеорологічний архів - БД Європейського центру середньострокових прогнозів погоди).
* ODB (база даних спостережень) локальні бази даних.
* Файли з даними в ASCII коді (CSV, grids and scattered data).
* NetCDF.

Є можливості графічного представлення даних, включаючи:

* різні картографічні проекції;
* поперечним перерізом тривимірних полів;
* вертикальні профілі;
* x-y графіки;
* накладення даних з різних джерел на одній карті;
* створення груп графіків на одній сторінці.

Основні недоліки системи:

* Складний і «морально старий» інтерфейс.
* Відсутність системи для збору даних.
* Відсутність системи для прогнозування погодних умов.
* Відсутність підтримки програмного продукту.

Перевагами системи є:

* Можливість підтримки великої кількості форматів файлів.
* Багатий набір графіків для побудови за даними.
* Вартість.

Висновок з аналізу даного програмного продукту наступний. Система недостатньо доопрацьована для використання в сучасних метеоцентри і метеостанціях.

2.2 Аналіз засобів програмної реалізації інформаційних систем

Для розробки даного програмного продукту використовувалися наступні технології:

* Python – мова програмування.
* Anaconda – програмне середовище.
* Django – веб-фреймворк.
* HTML \ CSS \ JavaScript – допомужні програмні засоби.
* PotsgreSQL – система керування базою даних.
* SQL – мова запитів для бази даних.

Далі по кожному пункту детальніше. Python – один з тих рідкісних мов програмування, які одночасно претендують на звання простих і потужних. Вас приємно здивує те, як легко можна зосередитися на вирішенні поставленого завдання, а не на синтаксисі і структурі мови, на якому користувач програмує.

Python – це простий в освоєнні і потужний мову програмування. Він надає ефективні високорівневі структури даних, а також простий, але ефективний підхід до об'єктно-орієнтованого програмування. Його елегантний синтаксис і динамічна типізація поряд з тим, що він є інтерпретується, роблять його ідеальним мовою для написания сценаріїв і швидкої розробки додатків в різних областях і на більшості платформ [12].

Історія назви починається з того як Гвідо ван Россум, творець мови Python, назвав його так на честь телешоу на BBC під назвою «Літаючий цирк Монті Пайтона» 1, а зовсім не тому, що він любить змій, що вбивають тварин обвиття свого довгого тіла навколо них і задавливания. Сама мова дуже простий і мінімалістичний. Читання хорошою програми на Python дуже нагадує читання англійського тексту, хоча і досить суворого. Така псевдокодовая природа Python є однією з його найсильніших сторін. Вона дозволяє вам зосередитися на вирішенні завдання, а не на самій мові.

Python – це приклад вільного і відкритого програмного забезпечення – FLOSS (Free / Libré and Open Source Soware).

Простіше кажучи, користувач має право вільно поширювати копії цього програмного забезпечення, читати його вихідні тексти, вносити зміни, а також використовувати його частини в своїх програмах. В основі вільного ПЗ лежить ідея спільноти, яке ділиться своїми знаннями. Це одна з причин, за якими Python такий хороший: він був створений і постійно поліпшується співтовариством, яке просто хоче зробити його краще.Завдяки своїй відкритій природі, Python був портований на багато платформ (тобто змінений таким чином, щоб працювати на них). Всі ваші програми зможуть запускатися на будь-який з цих платформ без будь-яких змін, якщо тільки користувач уникал використання системно-залежних функцій. Python можна використовувати в GNU / Linux, Windows, FreeBSD, Macintosh, Solaris, OS / 2, Amiga, AROS, AS / 400, BeOS, OS / 390, z / OS, Palm OS, QNX, VMS, Psion, Acorn RISC OS , VxWorks, PlayStation, Sharp Zaurus, Windows CE і навіть на PocketPC. Користувач може навіть використовувати таку платформу, як Kivy для створення ігор для iOS (iPhone, iPad) і Android. Важливо відзначити що мова програмування Python інтерпретується, а відрізняє його від компільованих то що програма, написана на компільовані мови програмування, як наприклад, C або C ++, перетворюється з вихідної мови, тобто C або C ++, в бінарний код, тобто нулі і одиниці за допомогою компілятора із застосуванням різноманітних прапорів і параметрів [18]. Коли користувач запускає таку програму, компонувальник / завантажувач копіює програму з диска в оперативну пам'ять і запускає її. Python ж, навпаки, не вимагає компіляції в бінарний код. Програма просто виконується з вихідного тексту.

Python сам перетворює цей вихідний текст в деяку проміжну форму, звану байткод, а потім переводить його на машинний мову і запускає. Все це помітно полегшує використання Python, оскільки немає необхідності піклуватися про компіляції програми, підключенні і завантаженні потрібних бібліотек і т.д. Разом з тим, це робить програми на Python набагато більш стерпним, так як досить їх просто скопіювати на інший комп'ютер, і вони працюють.

На завершення можна зазначити що Python є об'єктно-орієнтованої мовою програмування. Це означає що мінімальною одиницею в ньому явлется об'єкт, а також мову підтримує головні парадигми ООП, такі як інкапсуляція, поліморфізм і успадкування. Можливість розширення мови не менш важлива його характеристика, вона говорить про те що при потребі користувач може розширити функціонал мови програмування написавши до нього деякі модулі на більш низьких мовах програмування наприклад на С або С++, а вже потім використовувати ці модулі в Python коді.

Anaconda – дистрибутив мов програмування Python і R, що включає набір популярних вільних бібліотек, об'єднаних проблематики науки про дані і машинного навчання. Основна мета – поставка єдиним узгодженим комплектом найбільш затребуваних відповідним колом користувачів тематичних модулів (таких як NumPy, SciPy, Astropy і інших) з дозволом виникають залежностей і конфліктів, які неминучі при одиночній установці. Станом на 2019 рік містить більше 1,5 тис. Модулів. Для зберігання усіх модулів в одному місці є программа Anaconda Navigator, рис. 2.3., в якій користувач може легко завантажити будь-який модуль.

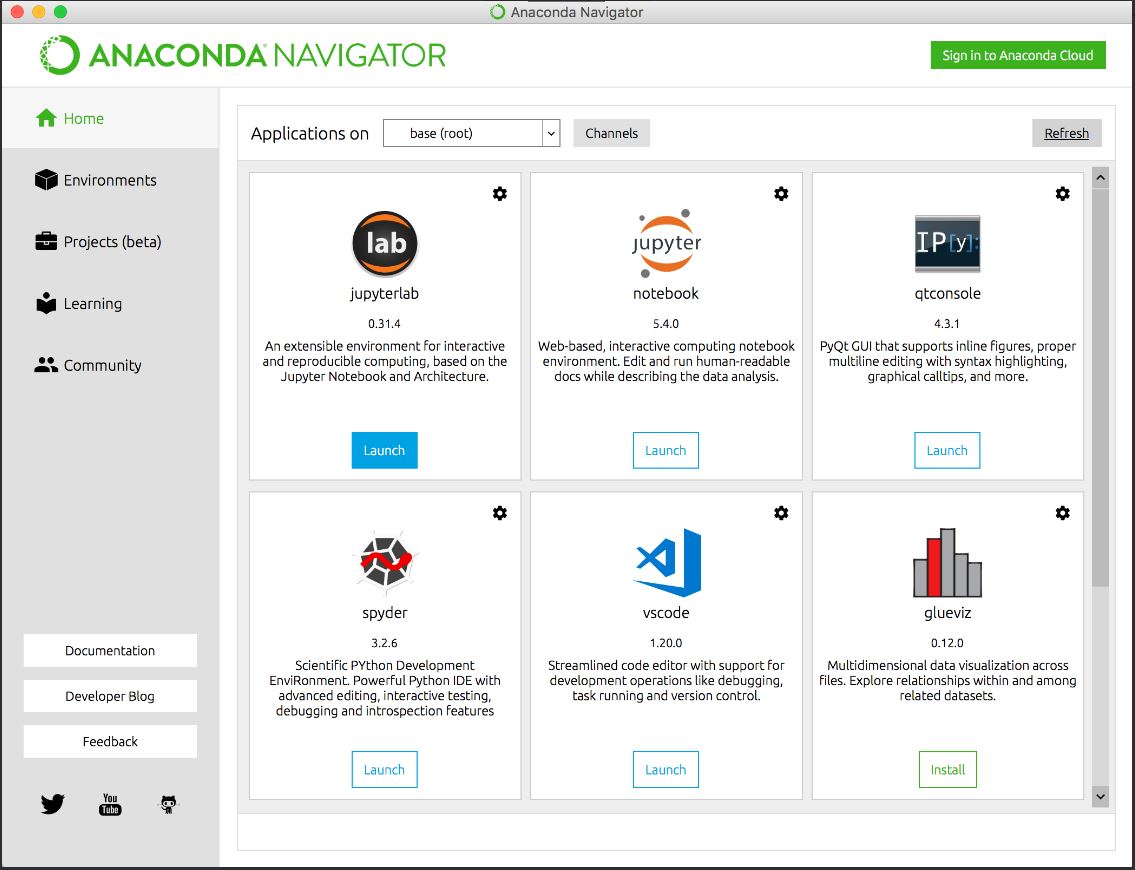


Рисунок 2.3 – Приклад інтерфейсу программи Anaconda Navigator

Основна особливість дистрибутива - оригінальний менеджер залежності не будуть задоволені conda з графічним інтерфейсом Anaconda Navigator, що дозволяє відмовитися від стандартних менеджерів пакетів (таких, як pip для Python). Дистрибутив скачується раз, і вся подальша конфігурація, в тому числі установка додаткових модулів, може проводитися в офлайні. Крім того, забезпечується можливість ведення декількох ізольованих середовищ з роздільним дозволом версійність залежностей в кожній.

Підтримуються платформи Linux (x86-64), Windows (i686, x86-64), macOS. Розповсюджується за ліцензією BSD, існує також комерційна версія (Anaconda Enterprise).

Основними бібліотеками, які надає Anaconda і які використовуються в проекті це Sklearn, numpy, pandas і matplotlib, далі про кожну трохи продробнее.

NumPy – фундаментальний модуль для наукових обчислень. Він включає в себе високоефективну реалізацію багатовимірних масивів, методів лінійної алгебри, генераторів випадкових чисел і т.д. Багато наукові бібліотеки, включаючи бібліотек для машинного навчання, використовують NumPy.

Всі дані, які подаються на вхід, обробляються і виходять на виході методів аналізу даних і машинного навчання, представлені у вигляді багатовимірних масивів. Методи обробки цих даних реалізовані як операції над цими масивами. Тому ефективне зберігання і обробка таких масивів даних є абсолютно фундаменательной завданням для високоефективних обчислень.

Як було видно до цього NumPy дозволяє ефективно зберігати багатовимірні масиви і виконувати векторизованних операції над ними. Це необхідно для роботи різних наукових бібліотек і методів машинного навчання. Однак дані в реальному світі не зберігаються у вигляді багатовимірних масивів чисел, готових для подачі в функції бібліотек машинного навчання. Зазвичай дані можуть зберігається в розрізнених файлах (наприклад, `csv`,` json`, `excel`) або часто в СУБД. Перед тим, як почати створювати навчальні моделі необхідно вивантажити, об'єднати в єдину структуру, провести очистку, аналіз, предобработку даних. Для цього існує pandas.

Pandas – бібліотека для аналізу і маніпуляції даних. Він працює поверх NumPy включаючи в себе всі його переваги і доповнюючи великою функціональністю. Pandas дозволяє завантажувати дані з різних джерел, наприклад `csv`,` json`, `excel`,` SQL`, об'єднувати різні дані в єдину структуру, включає в себе готові функції для обчислення всіляких статистичних показників, дозволяє працювати над ієрархічною структурою даних, включає в себе дуже потужний механізм угруповання і трансформації даних.

Scikit learn (коротко sklearn) надає уніфікований програмний інтерфейс для різних алгоритмів машинного навчання. Для навчання моделі дані розбиваються на дві частини. Перша частина називається навчальною вибіркою (training set), друга тестової (test set). Навчальна вибірка використовується для навчання моделі. Після отримання навченої моделі її якість тестується на тестовій вибірці.

Matplotlib – бібліотека на мові програмування Python для візуалізації даних двовимірної (2D) графікою (3D графіка також підтримується). Одержувані зображення можуть бути використані в якості ілюстрацій в публікаціях.

Matplotlib написаний і підтримувався в основному Джоном Хантером і поширюється на умовах BSD-подібної ліцензії. Генеруються в різних форматах зображення можуть бути використані в інтерактивній графіці, в наукових публікаціях, графічному інтерфейсі користувача, веб-додатках, де потрібна побудова діаграм. У документації автор зізнається, що Matplotlib починався з наслідування графічним командам MATLAB, але є незалежним від нього проектом

Django – це програмний каркас з багатими можливостями, що підходить для розробки складних сайтів і веб-додатків, написаний на мові програмування Python.

Django – фреймворк для веб-додатків на мові Python [13]. Один з основних принципів фреймворка – DRY (do not repeat yourself). Веб-системи на Django будуються з одного або декількох додатків, які рекомендується робити відчужуваними і підключаються. Це одне з помітних архітектурних відмінностей цього фреймворка від деяких інших (наприклад, Ruby on Rails). Також, на відміну від багатьох інших фреймворків, обробники URL в Django конфигурируются явно (за допомогою регулярних виразів), а не автоматично задаються зі структури контролерів.

Django проектувався дляроботи під управлінням Apache (з модулем mod\_python) і з використанням PostgreSQL в якості бази даних. В даний час, крім PostgreSQL, Django може працювати з іншими СУБД: MySQL (MariaDB), SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere і Oracle. Для роботи з базою даних Django використовує власний ORM, в якому модель даних описується класами Python, і по ній генерується схема бази даних.

Архітектура Django схожа на «Модель-Представлення-Контролер» (MVC). Контролер класичної моделі MVC приблизно відповідає рівню, який в Django називається Представлення (View), а презентаційна логіка Уявлення реалізується в Django рівнем Шаблонів (Templates). Через це рівневу архітектуру Django часто називають «Модель-Шаблон-Подання» (MTV).

Спочатку розробка Django велася для забезпечення більш зручної роботи з новинними ресурсами, що досить сильно відбилося на архітектурі: фреймворк надає ряд засобів, які допомагають у швидкій розробці веб-сайтів інформаційного характеру. Наприклад, розробнику не потрібно створювати контролери та сторінки для адміністративної частини сайту, в Django є вбудований додаток для керування вмістом, яке можна включити в будь-який сайт, зроблений на Django, і яке може управляти відразу декількома сайтами на одному сервері. Адміністративне додаток дозволяє створювати, змінювати і видаляти будь-які об'єкти наповнення сайту, протоколюючи всі скоєні дії, і надає інтерфейс для управління користувачами і групами (з пооб'єктного призначенням прав).

Веб-фреймворк Django використовується в таких великих і відомих сайтах, як Instagram, Disqus, Mozilla, The Washington Times, Pinterest, lamoda і ін.

Деякі можливості Django:

* ORM, API доступу до БД з підтримкою транзакцій.
* вбудований інтерфейс адміністратора, з уже наявними перекладами багатьма мовами.
* диспетчер URL на основі регулярних виразів.
* розширювана система шаблонів з тегами і спадкуванням.
* система кешування.
* інтернаціоналізація.
* підключається архітектура додатків, які можна встановлювати на будь-які Django-сайти.
* «generic views» - шаблони функцій контролерів.
* авторизація та утентіфікація, підключення зовнішніх модулів аутентифікації: LDAP, OpenID та інші.
* система фільтрів ( «middleware») для побудови додаткових обробників запитів, як наприклад включені в дистрибутив фільтри для кешування, стиснення, нормалізації URL і підтримки анонімних сесій.
* бібліотека для роботи з формами (успадкування, побудова форм по існуючої моделі БД).
* вбудована автоматична документація по тегам шаблонів і моделей даних, доступна через адміністративне додаток.

Деякі компоненти фреймворка між собою пов'язані слабо, тому їх можна досить просто замінювати на аналогічні. Але з деякими (наприклад, з ORM) це зробити не дуже просто.

Крім можливостей, вбудованих в ядро фреймворка, існують пакети, що розширюють його можливості.

На базі Django розроблено досить багато готових рішень, які розповсюджуються під вільною ліцензією, серед яких системи для управління інтернет-магазинами, універсальні системи управління вмістом, а також більш цілеспрямовані проекти.

HTML – стандартизований мову розмітки веб-сторінок у Всесвітній павутині. Код HTML інтерпретується браузерами; отриманий в результаті інтерпретації сторінка відображається на екрані монітора комп'ютера або мобільного пристрою [19].

Мова HTML до 5-ї версії визначався як додаток SGML, стандартної узагальненої мови розмітки за стандартом ISO 8879. Специфікації HTML5 формулюються в термінах DOM, об'єктної моделі документа.

Суворим варіантом HTML є XHTML, він успадковує синтаксис XML і є додатком мови XML в області розмітки гіпертексту.

HTML-сторінки, як правило, відкриваються браузерами обмінюючись з сервером інформацією по протоколу HTTP або HTTPS, у вигляді простого тексту або з використанням шифрування [20].

Мова гіпертекстової розмітки HTML був розроблений британським вченим Тімом Бернерс-Лі приблизно в 1986-1991 роках в стінах ЦЕРНу в Женеві в Швейцарії. HTML створювався як мова для обміну науковою і технічною документацією, придатний для використання людьми, які не є фахівцями в області верстки.

HTML успішно справлявся з проблемою складності SGML шляхом визначення невеликого набору структурних і семантичних елементів - дескрипторів. Дескриптори також часто називають «тегами». За допомогою HTML можна легко створити відносно простий, але красиво оформлений документ. Крім спрощення структури документа, в HTML внесена підтримка гіпертексту. Мультимедійні можливості були додані пізніше [21].

Першим загальнодоступним описом HTML був документ «Теги HTML», вперше згаданий в Інтернеті Тімом Бернерс-Лі в кінці 1991 року. У ньому описуються 18 елементів, що становлять початковий, відносно простий дизайн HTML. За винятком тега гіперпосилання, на них сильно вплинув SGMLguid, внутрішній формат документації, заснований на стандартному узагальненому мовою розмітки (SGML), в CERN. Одинадцять з цих елементів все ще існують в HTML 4 [17].

Спочатку мова HTML був задуманий і створений як засіб структурування та форматування документів без їх прив'язки до засобів відтворення (відображення) [28]. В ідеалі, текст з розміткою HTML повинен був без стилістичних та структурних спотворень відтворюватися на обладнанні з різною технічною оснащеністю (кольоровий екран сучасного комп'ютера, монохромний екран органайзера, обмежений за розмірами екран мобільного телефону або пристрою і програми голосового відтворення текстів). Однак сучасне застосування HTML дуже далеко від його початкової задачі. Наприклад, тег <table> призначений для створення в документах таблиць, але іноді використовується і для оформлення розміщення елементів на сторінці. З плином часу основна ідея платформонезавісимость мови HTML була принесена в жертву сучасним потребам в мультимедійному і графічному оформленні.

CSS – формальна мова опису зовнішнього вигляду документа, написаного з використанням мови розмітки, найчастіше HTML або XHTML. Також може застосовуватися до будь-яких XML-документах, наприклад, до SVG або XUL.

CSS використовується творцями веб-сторінок для завдання кольорів, шрифтів, стилів, розташування окремих блоків і інших аспектів представлення зовнішнього вигляду цих веб-сторінок. Основною метою розробки CSS було відділення опису логічної структури веб-сторінки, яке проводиться за допомогою HTML або інших мов розмітки, від опису зовнішнього вигляду цієї веб-сторінки, яке тепер проводиться за допомогою формального мови CSS. Такий поділ може збільшити доступність документа, надати велику гнучкість і можливість управління його поданням, а також зменшити складність і повторюваність в структурному вмісті.

Крім того, CSS дозволяє представляти один і той же документ в різних стилях або методах виведення, таких як екранне уявлення, друковане подання, читання голосом, спеціальним голосовим браузером або програмою читання з екрану, або при виведенні пристроями, що використовують шрифт Брайля.

JavaScript – мультіпарадігменний мову програмування. Підтримує об'єктно-орієнтована, імперативний і функціональний стилі. Є реалізацією стандарту ECMAScript.

JavaScript зазвичай використовується як вбудований мова для програмного доступу до об'єктів додатків. Найбільш широке застосування знаходить в браузерах як мова сценаріїв для додання інтерактивності веб-сторінок.

Основні архітектурні риси: динамічна типізація, слабка типізація, автоматичне керування пам'яттю, прототипне програмування, функції як об'єкти першого класу.

На JavaScript вплинули багато мов, при розробці була мета зробити мову схожим на Java. Мовою JavaScript не володіє будь-яка компанія або організація [21], що відрізняє його від ряду мов програмування, використовуваних в веб-розробці.

Назва «JavaScript» є зареєстрованим товарним знаком корпорації Oracle в США.

JavaScript є об'єктно-орієнтованою мовою, але що використовується в мові прототипирование обумовлює відмінності в роботі з об'єктами в порівнянні з традиційними клас-орієнтованими мовами. Крім того, JavaScript має ряд властивостей, властивих функціональним мовам, – функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання - що додає мові додаткову гнучкість.

Незважаючи на схожий з Сі синтаксис, JavaScript в порівнянні з мовою Сі має корінні відмінності:

* об'єкти з можливістю інтроспекції;
* функції як об'єкти першого класу;
* автоматичне приведення типів;
* автоматичне прибирання сміття;
* анонімні функції.

У мові відсутні такі корисні речі, як:

* стандартна бібліотека: зокрема, відсутній інтерфейс програмування додатків по роботі з файловою системою, управління потоками введення-виведення, базових типів для бінарних даних;
* стандартні інтерфейси до веб-серверів і баз даних;
* система управління пакетами, яка б відстежувала залежності і автоматично встановлювала їх.

PostgreSQL – вільна об'єктно-реляційна система управління базами даних, СУБД.

Існує в реалізаціях для безлічі UNIX-подібних платформ, включаючи AIX, різні BSD-системи, HP-UX, IRIX, Linux, macOS, Solaris / OpenSolaris, Tru64, QNX, а також для Microsoft Windows. Також для неї є веб-інтерфейс бібліотена під назвою pgadmin4, рис. 2.4.

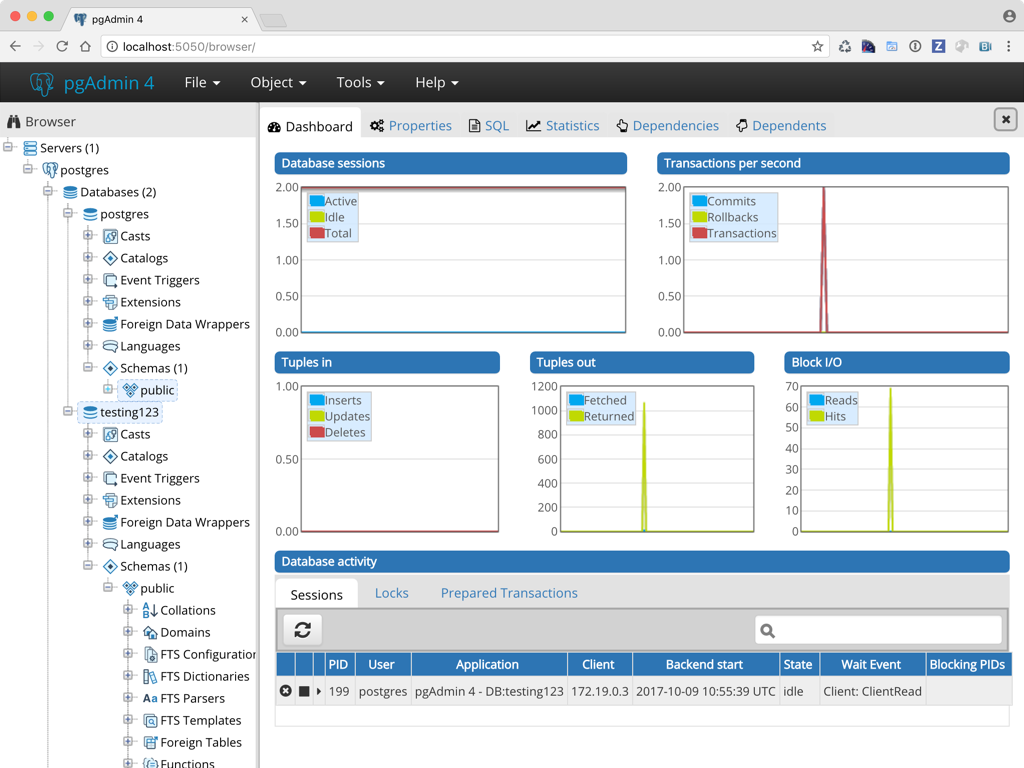


Рисунок 2.4 – Веб-інтерфейс для PostgreSQL

PostgreSQL базується на мові SQL і підтримує багато з можливостей стандарту SQL: 2011 [24].

У PostgreSQL версії 12 є наступні обмеження – максимальний розмір бази даних обмежень немає, максимальний розмір таблиці 32 Тбайт, максимальний розмір поля 1 Гбайт, максимум записів в таблиці обмежено розмірами таблиці, максимум полів у записі 250-1600, залежно від типів полів, максимум індексів в таблиці обмежень немає.

Сильними сторонами PostgreSQL вважаються:

* високопродуктивні і надійні механізми транзакцій і реплікації;
* розширювана система вбудованих мов програмування: в стандартному постачанні підтримуються PL / pgSQL, PL / Perl, PL / Python і PL / Tcl; додатково можна використовувати PL / Java, PL / PHP, PL / Py, PL / R, PL / Ruby, PL / Scheme, PL / sh і PL / V8, а також є підтримка завантаження модулів розширення на мові C [22];
* успадкування;
* можливість індексування геометричних об'єктів і наявність базується на ній розширення PostGIS;
* вбудована підтримка слабоструктурованих даних в форматі JSON з можливістю їх індексації;
* розширюваність (можливість створювати нові типи даних, типи індексів, мови програмування, модулі розширення, підключати будь-які зовнішні джерела даних).

PostgreSQL поддерживает большой набор встроенных типов данных:

PostgreSQL підтримує великий набір вбудованих типів даних:

* + - Чисельні типи.
      * Цілі.
      * С фіксованою точкою.
      * С плаваючою точкою.
      * Грошовий тип (відрізняється спеціальним форматом виведення, а в іншому аналогічний числах з фіксованою крапкою з двома знаками після коми).
    - Символьні типи довільної довжини.
    - Двійкові типи (включаючи BLOB).
    - Типи «дата / час» (повністю підтримують різні формати, точність, формати виведення, включаючи останні зміни в часових поясах).
    - Логічний тип.
    - Перерахування.
    - Геометричні примітиви.
    - Мережеві типи.
      * IP і IPv6-адреси.
      * CIDR-формат.
      * MAC-адресу.
    - UUID-ідентифікатор.
    - XML-дані.
    - Масиви.
    - JSON.
    - Ідентифікатори об'єктів БД.
    - Псевдотіпов.

Більш того, користувач може самостійно створювати нові необхідні йому типи і програмувати для них механізми індексування за допомогою GiST.

SQL – простими словами, це мова програмування структурованих запитів (SQL, Structured Query Language), який використовується в якості ефективного способу збереження даних, пошуку їх частин, поновлення, вилучення з бази і видалення.

З 1974 року, коли мова структурованих запитів тільки з'явився, він забезпечує взаімодейтсвіе з системами управління базами даних (СКБД) в усьому світі.

SQL, як простий і легкий у вивченні мову з області вільного програмного забезпечення, сьогодні активно застосовується:

* розробниками баз даних (забезпечують функціональність додатків),
* тестувальниками (в ручному і автоматичному режимі),
* адміністраторами (виконують підтримку працездатності середовища).

Мова універсальний і має чітко певною структурою за рахунок усталених стандартів [25]. Взаємодія з базами даних відбувається швидко навіть в ситуаціях, коли обсяги даних великі (Big Data). Крім того, ефективне управління можливе навіть без особливих знань коду.

Незважаючи на наявність діалектів і відмінностей в синтаксисі, в більшості своїй тексти SQL-запитів, що містять DDL і DML, можуть бути досить легко перенесені з однієї СУБД в іншу.

Існують системи, розробники яких спочатку орієнтувалися на застосування щонайменше кількох СУБД (наприклад: система електронного документообігу Documentum може працювати як з Oracle Database, так і з Microsoft SQL Server і DB2).

Природно, що при застосуванні деяких специфічних для реалізації можливостей такої переносимості добитися вже дуже важко.

Наявність стандартів і набору тестів для виявлення сумісності і відповідності конкретній реалізації SQL загальноприйнятому стандарту тільки сприяє «стабілізації» мови.

Правда, варто звернути увагу, що сам по собі стандарт місцями занадто формалізований і роздутий в розмірах (наприклад, базова частина стандарту SQL: 2003 складається з більш ніж 1300 сторінок тексту).

За допомогою SQL програміст описує тільки те, які дані потрібно витягнути або модифікувати. Те, яким чином це зробити, вирішує СУБД безпосередньо при обробці SQL–запиту [29].

Однак не варто думати, що це повністю універсальний принцип - програміст описує набір даних для вибірки або модифікації, проте йому при цьому корисно уявляти, як СУБД розбиратиме текст його запиту.

Чим складніше сконструйований запит, тим більше він допускає варіантів написання, різних за швидкістю виконання, але однакових за підсумковим набору даних. Це дозволяє максимально опитимізувати запити, що може бути дуже корисним для систем високої навантеженості.

2.3 Висновки до другого розділу

Виконавши аналіз сучасних інформаційних систем, які використовуються в якості головних електронних інструментів в метеорології, було сформовано концепцію розробки програмного засобу.

Щоб максимально відповідати запитам сучасних клієнтів, інформаційна система має бути легкодоступною для користувачів надаючи зрозумілий і простий інтерфейс для роботи незалежно від пристрою користувача або встановленого на ньому ПЗ.

Програмний комплекс повинен надавати багатий функціонал для отримання і перегляду даних. Побудова короткострокових і довгострокових прогнозів погоди для кожного регіону.

Розробку інформац­ійної системи слід виконати засобами програмування на мові Python з використанням спеціалізованого фреймворку для web-розробки – Django [13].

Таке рішення дозволить розробити сучасну та функціональну системиу, а також покращити показники ефективності розподілу системних ресурсів.

Розроблені інформаційні системи з використанням таких засобів є дуже потужними та не потребують великих витрат від апаратної складової.

Так як даних буде багато і вони будуть приходити від великого кількість джерел то програмний комплекс повинен бути оптимізований для роботи з великими даними, в даному випадку кращим рішенням буде використовувати систему управління базами даних PostgreSQL.

# РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ ЗБОРУ, ОБРОБКИ ТА АНАЛІЗУ МЕТЕОДАНИХ



## Проектування розробки інформаційної системи

Початковим етапом при розробці програмного забезпечення є проектування. Проектування дозволяє визначити головні функціональні можливості програмного забезпечення, його архітектуру, складові, а також компоненти з якими взаємодіє програмне забезпечення.

В більшості випадків проект складається з розробки головних діаграм, які в графічному вигляді дозволяють визначити усі складові.

В якості засобу проектування використовується мова UML, яка є спеціалізованою нотаціонною мовою, яку використовують розробники для графічного представлення програмного забезпечення. В якості середовищ проектування може бути використано різні редактори, які дозволяють взаємодіяти зі спеціалізованими об’єктами проектування.

Одним із засобів проектування, є створення діаграми варіантів використання, завдяки якій, визначаються усі вимоги до кінцевої системи.

Діаграма варіантів використання складається з об’єктів, які мають вигляд овалу, та запису функціональної ролі. Зовнішні об’єкти, якими можуть бути як користувачі так і сервера, зображаються у вигляді чоловічка.

Одним з вайжливіших елементів на діаграмі варіантів використання є використання стрілок, кожна з яких має своє спеціалізоване призначення. В більшості випадків, саме стрілки, дозволяють розробникам орієнтуватися в програмній системі, а також створювати систему зв’язків елементів між собою.

Діаграма варіантів використання для інформаціійнох системи для збору, обробки та аналізу метеоданих, зображено на рис. 3.1.

Програмне рішення має web-архітектуру, отже дозволяє користувачам отримувати усі функціональні можливості без інсталяції.

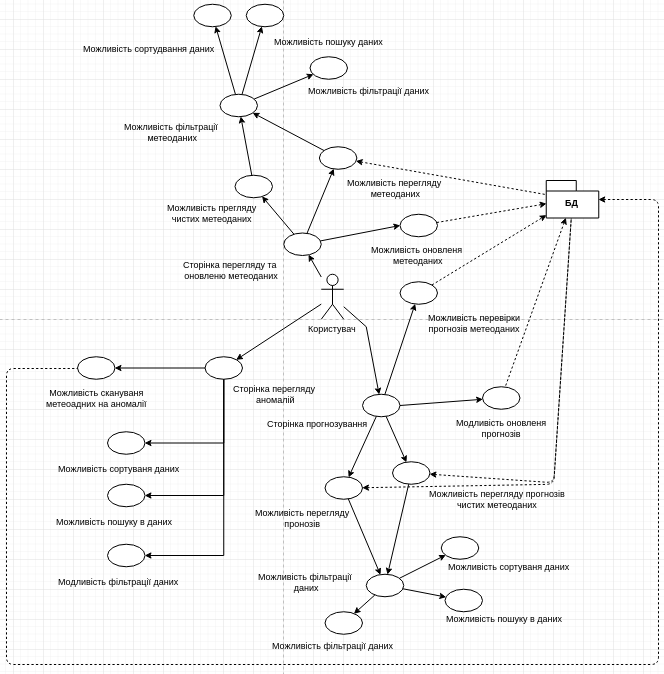


Рисунок 3.1 – Діаграма варіантів використання до інформаційної системи

Після виконання всіх етапів проектування, здійснюється перехід до стадії розробки програмного забезпечення.

Добре обґрунтований проект, дозволяє виявити усі недоліки системи, та виправити їх, ще до початка кодінгу.

Також, проектування є допоміжним засобом, для подальшого тестування, за рахунок того, що воно поєднує у собі усі особистості роботи системи, та допомагає проаналізувати схожість розробленої системи з проектом.

Усі елементи програмного забезпечення є взаємопов’язаними з роботою з метеорологічними даннми. Це дозволяє отримати повний огляд метеорологічної ситуації.

Кожен із варіантів використання має спеціалізоване призначення, і має бути описан індивідуально.

Перегляд знайдених аномалій. Для перегляду знайдених аномалій користувачу потрібно в шапці сайту виьрати пункт «перегляд аномалій». Після переходу на данну сторінку користувач бачить таблицью з усіми знайденими аномаліями в метеоданих, доступна функція фільтрації, пошуку та сортування.

Перегляд зібраних метеоданих. Для перегляду зібраних метеоданих користувачу потрібно в шапці сайту виьрати пункт «метеоданні». Після переходу на данну сторінку користувач бачить таблицью з усіми зібраними метеоданими, доступна функція фільтрації, пошуку та сортування.

Перегляд розрахованих прогнозів. Для перегляду розрахованих прогнозів користувачу потрібно в шапці сайту виьрати пункт «перегляд прогнозів». Після переходу на данну сторінку користувач бачить таблицью з усіма прогнозами метеоданих та погодних явищь, доступна функція фільтрації, пошуку та сортування.

Перегляд данних в таблицях. Користувач може переглядати дані виведені в таблиці на сторінках сервісу. Перший рядок таблиці це рядок з іменами стовпчикиів, при натисканні на ім’я виконуеться сортування таблиці в алфавітному порядку за цим ім’ям. При повторному натисканні буде відортивовано в обратному алфавітному порядку.

Пошук. Є розширенням інформаційної системи та дозволяє відшукувати данні в таблицях за будь-яким ключем. Ключем може бути будь-який текст, будь-то цифри, символи чи букви.

Оновлення метеоданих. На сторінці прегляду метеоданниз користувач може натиснути кнопку оновлення метеоданих, це передасть сигнал на сервер та він почне викачувати нові метеоданні з ресурсів. Після оновлення система запускае алгоритм пошуку аномалій в цих данних, далі на вилучених та «чистих» данних система робить прогнозування метеоданих на наступні 3 доби і вже по цим метеоданни система погнозуе погодні явища. Усі данні які оброблюються на данному етапі записуються в базу данних.

Оновлення прогнозу. На сторінці перегляду прогнозів користувач має можливість оновити зроблений системою прогноз, ця функція потрібна для тестування алгоритму прогнозування, та використовуется коли нових данних не має або сервер оновив нейромережі які працювали з даними.

Запуск алгоритма пошуку аномалій. На сторінці перегляду аномалій користувач має можливість оновити список знайдений аномалій, ця функція потрібна для тестування алгоритму пошуку аномалій, та використовуется коли нових данних не має або сервер оновив нейромережі які працювали з даними.

Можливість проведення тестування ефективності навчання нейромереж на сирих та «чистих» метеоданих. Процесс тестування починается з того що нейромережа для пошуку аномалій пробігає данні, формує два списки данних, на цих списках вчаться нейромережі прогнозування метеоданих та погодних явищь, пілся цього ці нейромережі проходять тестування на сирих та «читстих» данних де і порівнюется їх точність.

Після визначення усіх функціональних можливостей програмного забезпечення, є необхідність виконати побудову головних класів. Діаграма класів дозволяє визначити розробникам усі складові елементи системи, та їх особливості роботи. В більшості випадків діаграма класів будується до початка розробки, а потім модифікується, в залежності від зміни умов, або функціоналу програмного забезпечення.

Інформаційну систиему можно розділити на два головних компоненти для кожного з них була розроблена діаграма класів. Компонент інтерфейсу та контрелеру та компонент роботи з алгоритмами обробки та нейромережами. Діаграма класів компонента інтерфейсу та контролеру, зображено на рис. 3.2.

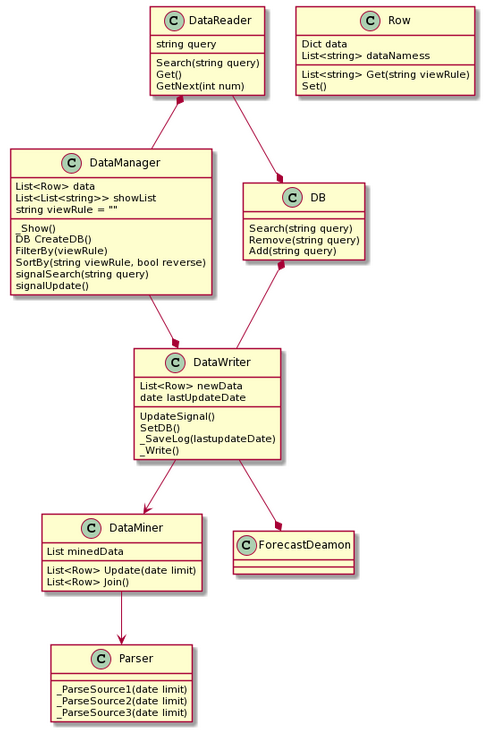


Рисунок 3.2 – Діаграма класів компонента інтерфейсу та контролеру інформаційної системи

На діаграмі зображено функціональну залежність компонентів. У зв’язку з тим, що це web-система то простір класів розроблюється з точки зору окремих компонентів.

В залежності від типу проекту діагарма класів може відрізнятися одна від одної, але в більшості випадків вона має єдину логічну струкруту.

Також на діаграмі визначається залежність компонентів один від одного та вказується система зв’зку між ними.

Реляційні зв’язки є досить суттєвими та необхідні для функціонального аналізу проекту системи.

На цій диаграмі можна розглянути усі задіяні класи. Клас DataReader, DB та DataWriter це підкомпонент для роботи з базою данних, Вони використовуються для запису та зчитування данних.

Кожен з цих класів є обгорткою над інтегрованих в Django класів та пракює за данимим через спеціальний контейнер, за ради більш зручної роботи з ними.

Клас DataReader тісно пов’язаний з інтерфейсом сторінок, він витягує потрібні данні з бази данних та передає їх до головного класу контролера DataManager.

Клас DataReader реалізуе функціонал фільтрації, пошуку, сортування та зміни сторінок.

Клас DataWriter реалізує функціонал запису данних в базу. Цей клас бере данні з двох джерел, з компонента обробки та аналізу метеоданих та компонента парсінгу даних з зовнішніх джерел. Клас DB реалізує базову логіку роботи з базою даних.

Підкомпонент для парсінгу данних з зовнішніх джерел складається з двох класів DataMiner і Parser, клас Parser реаоізує функціонал збору данних з зовнішніх джерел, він використувую https запроси та бібліотеку BS4 для парсінгу. Клас DataMiner використовується для обєднання усіх вилучених данних в єдиний список для подальшого завантащення в базу данних.

Діаграма класів компонента алоритмів розрахунку та нейромереж на рис 3.3.

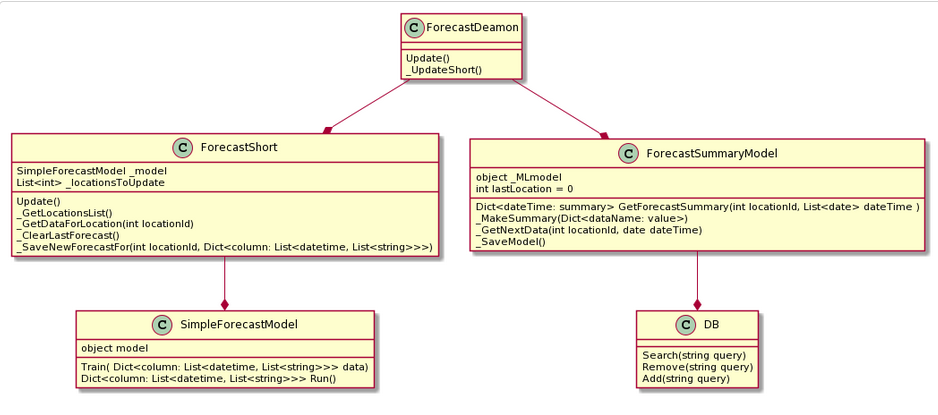


Рисунок 3.3 – Діаграма класів компонента алоритмів розрахунку та нейромереж

Компонент алоритмів розрахунку та нейромереж складається з класів ForecastDeamon, ForecastShort та ForecastSummaryModel. ForecastDeamon це клас контролер усіх алгоритмів обробки та аналізу, він отримує сигнал на запуск алгоритмів обробки потім передає його далі в класи ForecastShort та ForecastSummaryModel. Клас ForecastShort потрібен для прогнозування метеорологічних данних та роботи з нейромережею SimpleForecastModel, яка прогнозує метеодані. Клас ForecastSummaryModel реалізує функціонал прогнозування погодних явищ використовуючи нейромережу для аналізу метеоданих.

Для більш покращеного розуміння алгоритмічного складу розробленого програмного забезпечення, використовується побудова алгоритмічних блок-схем які дозволяють зрозуміти користувачам та розробникам, як саме працює програмне забезпечення.

В першу чергу слід описати схему роботи алгоритму збору та аналізу даних на аномалії, рис. 3.4. Даний алгоритм наводиться в дію при нажаніі на кнопку «оновити дані». Дана блок-схема чітко відображає наступний порядок дій, спершу активується система збору даних, вона опитує наявні у неї джерела даних і виробляє їх завантаження, далле за допомогою нейромережі для посіка аномалій виявляються проблемні дані які потім записують у різні списки, сирих і очищених дані, ці дані заносяться в базу для подальшого аналізу і прогнозування.

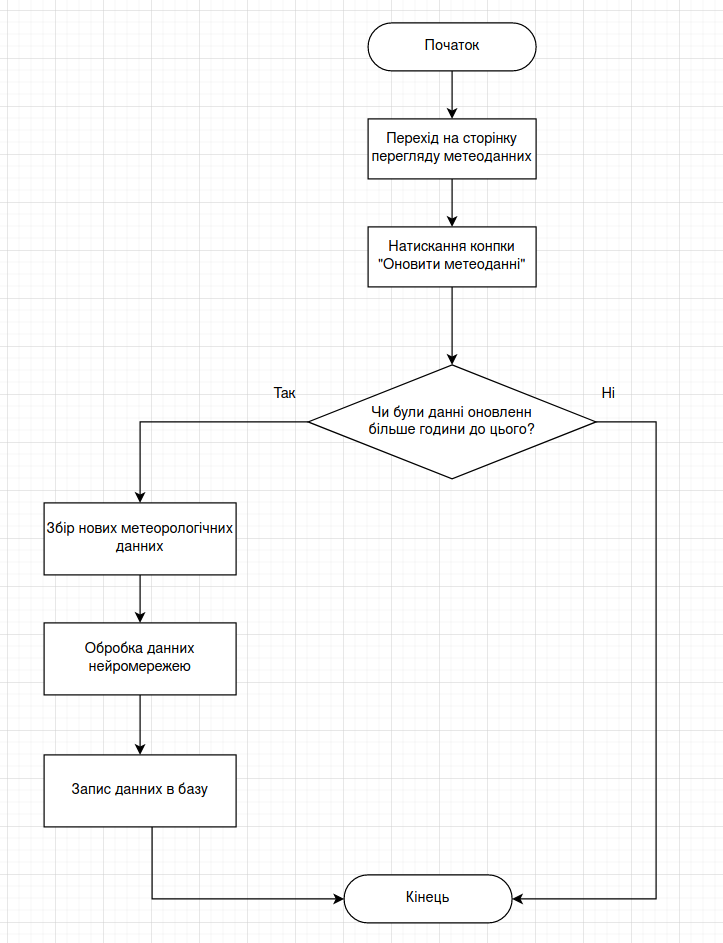


Рисунок 3.4 – Схема работи алгоритму збору и аналізу данних на аномалії

Друга блок-схема представлена ​​в даному розділі це схема обробки отриманих даних і побудова прогнозу на слушні три доби, рис. 3.5, кожна запість прогнозу містить спрогнозовані інформацію про погоду і прогнозоване за цими даними погодне явище, різниця між записами о третій годині часу.

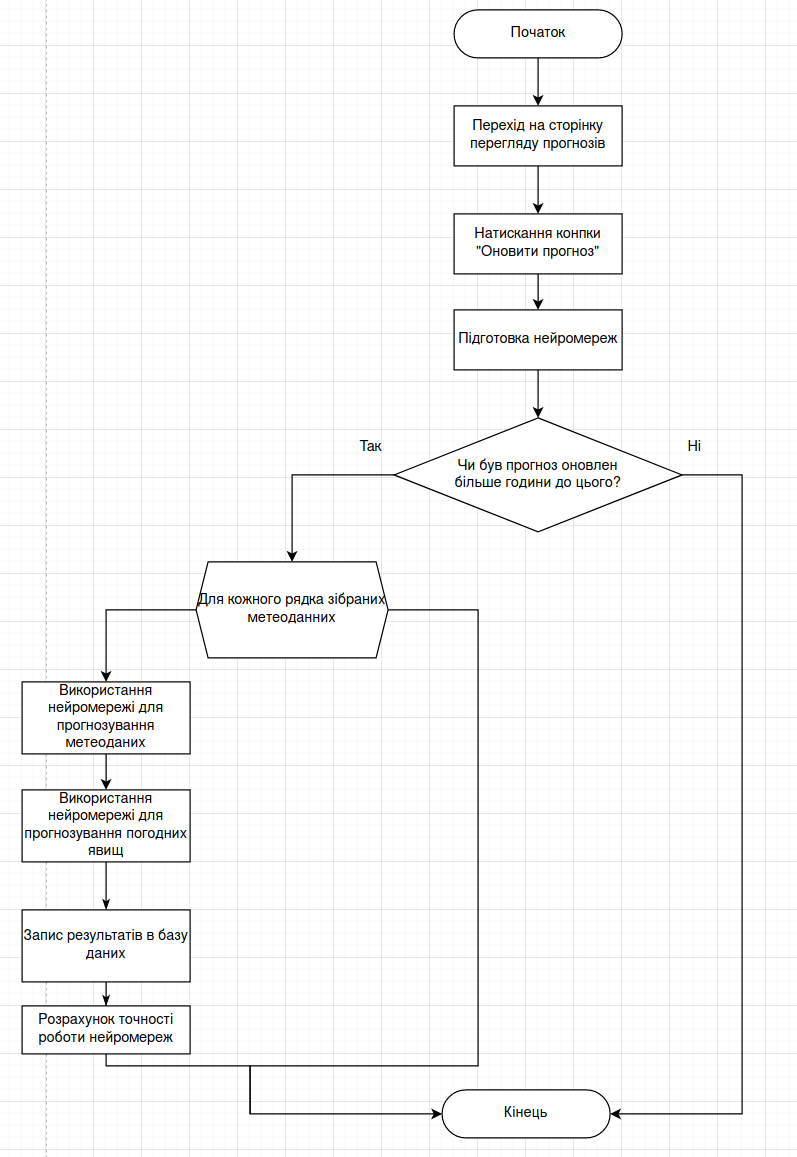


Рисунок 3.5 – Схема обробки отриманих даних і побудова прогнозу

Кроки по блок-схемі наступні, вичітваніе даних з обох списків, далі підготовка нейромереж під ці дані і складання прогнозу метеоданих, слідом інша нейросеть за прогнозом робить прогноз погодного явища, в процесі система аналізує точність роботи всіх нейромереж і порівнює дані їх точності.

Ці результати потрібні для основної мети даної роботи, аналіз впливу систем пошуку аномалій в метеорологію шляхом порівняння точності алгоритмів прогнозування навчених на даних з аномаліями і без.

Алгоритм представлений з додатковими оптимізаціями, внаслідок чого алгоритм простий для розуміня і при тому швидкий та зручний в використані.

Інші алгоритми використовані в системі більш прості та не потребують додаткового зобарження у вигляді блочної схеми.

Це алгоритми пошуку, фільтрації, сортування, вичитки даних, вичитки данних з зовнішніх джерел, роботи з базою даних та розміщення даних та збору сторінок виводу інформації.

## Опис розробленої бази даних інформаційної системи

Під час розробки інформаційної ситсеми було визначено перелік таблиць та їх структуру, які є необхідними для доброго функціонування усієї системи.

Ці талиці складають структуру організації даних в базі з якою і працює система.

Кожна таблиця складаеться з стовпців, кожен з яких має своє ім’я та тип даних, це частини даних які записуються в базу. Кожен рядок таблиці це окремих запис з даних.

Однією з перших таблиць, яка була розроблена є таблиця метеорологічних даних, рис. 3.6.

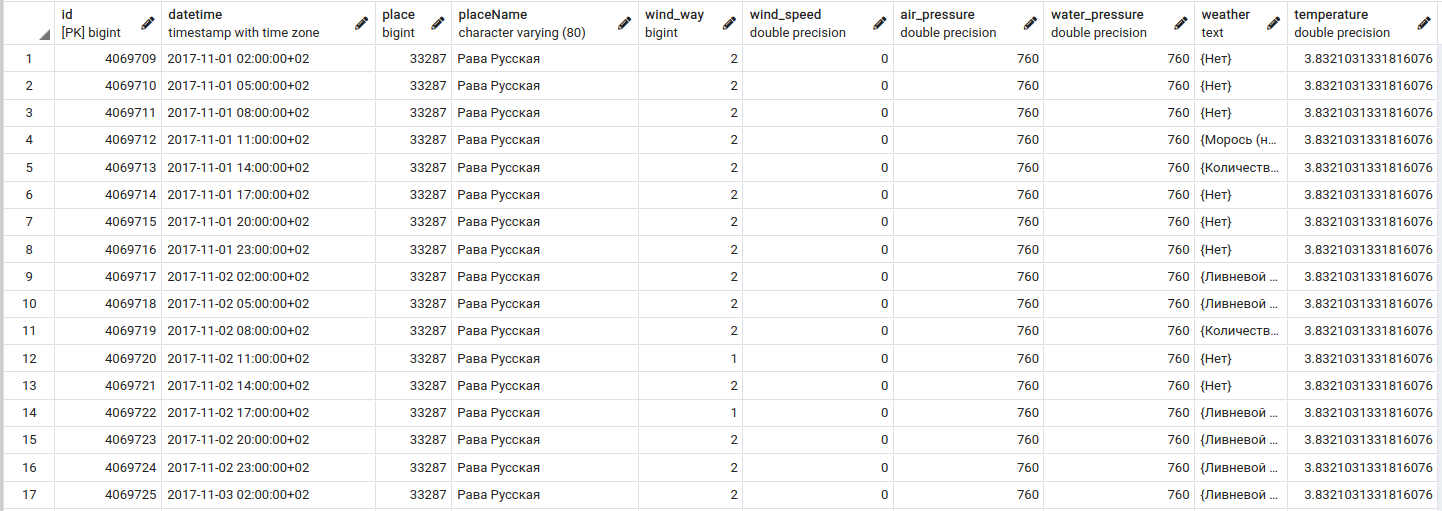


Рисунок 3.6 – Вікно таблиці метеоданих в базі даних

Дана таблиця створена засобами фреймворка Django і виконується наступним кодом.

class Meteodata(models.Model):

id = models.BigAutoField("id", primary\_key=True)

datetime = models.DateTimeField("Дата и время")

place = models.BigIntegerField("Код места")

placeName = models.CharField("Место", max\_length=80)

temperature = models.FloatField("Температура воздуха")

wind\_way = models.BigIntegerField("Направление ветра")

wind\_speed = models.FloatField("Скорость ветра")

air\_pressure = models.FloatField("Атмосферное давление")

water\_pressure = models.FloatField("Давление моря")

weather = models.TextField("Погодное явление")

class Meta:

verbose\_name = "Метеоданные"

verbose\_name\_plural = "Метеоданные"

Детально по кожному полю таблиці метеоданих:

* Id це поле целочисленном типу зберігає унікальний номер запису.
* Datetime це підлозі в якому зберігається дата і времм коли було зроблено запис.
* Place це поле в якому зберігається назва регіону в якому був зроблений запис.
* placeNumber це поле в якому зберігається номер регіону в якому був зроблений запис.
* temperature поле температури яка була зафіксована в момент запису.
* wind\_way поле зберігає напрямок вітру.
* wind\_speed поле зберігає значення швидкості вітру.
* air\_pressure поле зберігає значення атмосферного тиску.
* water\_pressure поле зберігає значення атмосферного тиску на рівні моря.
* weather поле зберігає погодне явище відбувалося в момент запису.

Відповідно до цієї таблиці створена таблиця для записів яка має однаковий набір полів, рис 3.7.

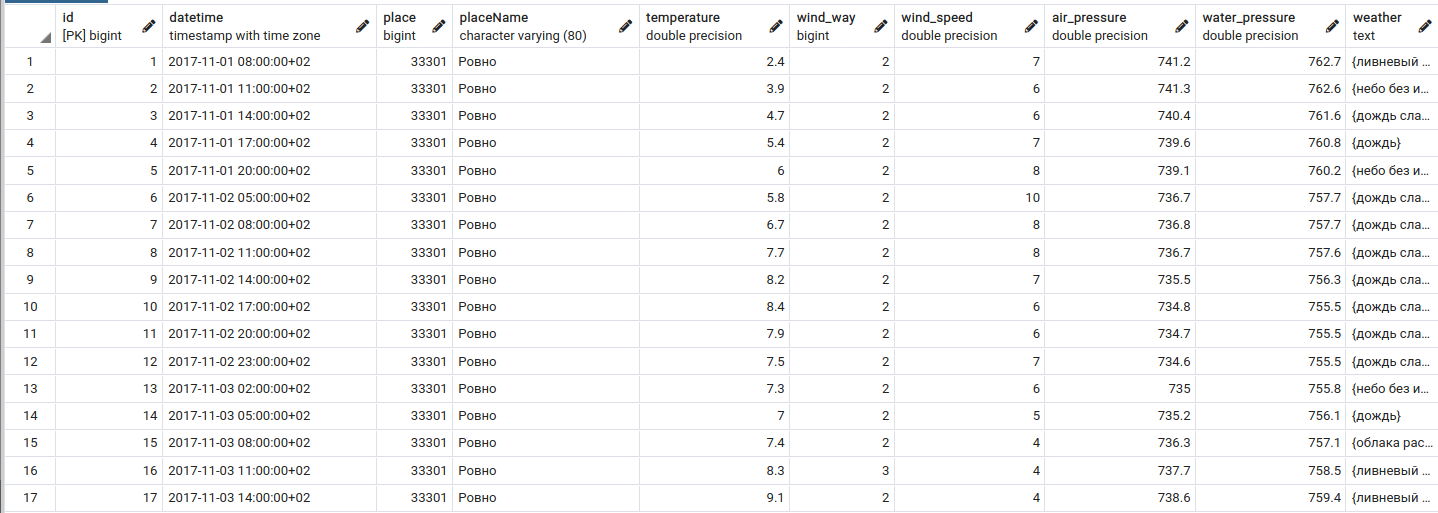


Рисунок 3.7 – Таблиця очищених метеоданих

Таблиця прогнозів про погоду містить збігаються поля з таблицею з інформацію про погоду.

Таблиця зберігає список знайдених аномалій містить наступні поля:

* Id поле зберігає унікальний номер запису.
* Meteodata\_Id містить унікальний номер запису з таблиці мететоданних.
* Fieldname містить назву поля в якому була обноружена аномалія.
* Value зберігає в собі значення поля в якому була виявлена аномалія.
* Anomaly містить кратне опис аномалії або її тип.

Дана табіца створюється наступним кодом:

class MeteodataAnomalies(models.Model):

id = models.BigAutoField("id", primary\_key=True)

meteodata\_id = models.BigIntegerField("Код записи")

fieldname = models.TextField("Название поля")

value = models.TextField("Значение")

anomaly = models.TextField("Аномалия")

class Meta:

verbose\_name = "Аномалии"

verbose\_name\_plural = "Аномалии"

Також варто звернути увагу що були отримані раніше таблиці є додатковими, крім них Django створює радий основних таблиць, це таблиці зберігають в собі дані про підключених до системи з'єднань, їх унікальні коди, паролі адміністратора, таблиці логування дій в системі, таблиця логування помилок в системі, таблиця підключених модулів і т.д.

База даних вкрай чутливий елемент системи і требудет до себе великої уваги. Так як система працює з великими даними повинна бути можливість легко взаємодіяти з нею, здійснювати запис в базу даних з високою швидкістю. Для цього раздаботанние таблиці даних повинні бути схожі, а дані в них однорідні. Дане завдання оптимізації процесів вирішується на рівні програмного інтерфейсу надається фреймворком Django 3, а також додатковими бібліотеками, які максимально спрощують створення системи.

Як і було позначена раніше Django генерує таблиці в базі даних, для цього використовується python код який описує структуру яку повинна мати таблиця. Для цього використовується наступні типи даних:

* BinaryField. Створює BLOB-поле для зберігання двійкових даних (наприклад, зображень, аудіо чи інших мультимедійних об'єктів)
* BooleanField. Створює логічне поле для зберігання значень True / False (або 0/1)
* DateField. Створює поле дати для зберігання дат.
* FloatField. Створює стовпець для зберігання чисел з плаваючою комою.
* TextField. Створює текстове поле для зберігання тексту.
* BigIntegerField. Створює велике ціле число, щоб вмістити числа від –9223372036854775808 до 9223372036854775807. Цей діапазон може змінюватися залежно від марки БД
* EmailField. Примушує текст – це дійсний електронний лист із внутрішнім Django EmailValidator, щоб визначити, що є, а що не є дійсним. Працює так само, як за замовчуванням CharField має максимальну довжину 254 символи, а також примушує рядок – це дійсний електронний лист.

Для аналізу ієрархії таблиць бази даних іноді використовуються діаграм зв'язку таблиць, це необов'язкові метод аналізу який потрібен тільки для наочності та полегшеного розуміння структури бази даних. Детально описуючи його можна виділити наступне.

Всередині діаграми бази даних у кожній зв'язку є три окремих елемента: кінцеві точки, стиль лінії і зв'язані таблиці.

Кінцеві точки. Кінцеві точки лінії показують вид зв'язку: "один до одного" або "один до багатьох".

Якщо на одній кінцевій точці зв'язку знаходиться ключ, а на іншій – цифра вісім, то це зв'язок «один до багатьох». Якщо в зв'язку по одному ключу на кожній кінцевій точці, то це зв'язок «один до одного».

Стиль лінії. Різновид лінії (не її кінцеві точки) показує, перевіряє чи СКБД посилальну цілісність для зв'язку при додаванні нових даних в таблицю, пов'язану з допомогою зовнішнього ключа.

Якщо зв'язок намальована у вигляді суцільної лінії, це означає, що СКБД перевіряє посилальну цілісність для зв'язку при додаванні або зміні рядків у таблиці, пов'язаної з допомогою зовнішнього ключа.

Якщо пунктирна лінія, це означає, що СКБД не перевіряє посилальну цілісність для зв'язку при додаванні або зміні рядків у таблиці, пов'язаної з допомогою зовнішнього ключа.

Зв'язані таблиці. Лінія зв'язку показує, що дві таблиці, зв'язані з допомогою зовнішнього ключа.

Для зв'язку «один до багатьох» таблиця, зв'язана з допомогою ключа, – це таблиця близько цифри на лінії.

Якщо обидві кінцеві точки лінії приєднані до однієї таблиці, це означає зворотну зв'язок.

Для будь-якої бази даних можна створити будь-яку необхідну кількість діаграм; кожна з таблиць бази даних може використовуватися у будь-якій кількості діаграм.

Таким чином, для візуалізації різних частин бази даних або для акцентування різних аспектів її конструювання можна створювати різні діаграми. Наприклад, можна створити велику діаграму, в якій будуть відображатися всі таблиці і стовпчики, а також меншу діаграму, в якій будуть відображатися всі таблиці, але не буде стовпців.

Всередині діаграми бази даних кожна таблиця має три окремих елементів: заголовка вікна, список вибору рядків і набір стовпців властивостей. Рядок заголовка в рядку заголовка відображається ім'я таблиці.

Якщо таблицю було змінено, але ще не збережено, то після імені таблиці з'являється зірочка (\*), що показує наявність незбережених змін. Область виділення рядка.

Щоб вибрати стовпець бази даних в таблиці, клацніть область виділення рядка. Якщо стовпець є первинним ключем таблиці, то в цьому списку відображається символ ключа.

Стовпці властивостей. Набір стовпців властивостей видно на поданні таблиці.

Таблицю можна переглянути в будь-якому з п'яти різних уявлень, що дозволяють підібрати підходящий розмір і розміщення елементів діаграми.

## Опис розробленого інтерфейсу до інформаційної системи

Інформаційну систему розроблено в кращих традиціях web-архітектури, за рахунок чого сайт складається з адміністраторської частини та клієнтської.

Адміністратор системи може додавати будь-яку інформацію на сайт, змінювати поточну, адмініструвати облікові записи та інше.

Панель адміністратора виконана у вигляді переліку, що спрощує навігацію, та дозволяє поступово виконувати різні дії з контентом.

Головне вікно панелі адміністратора зображено на рис. 3.8.

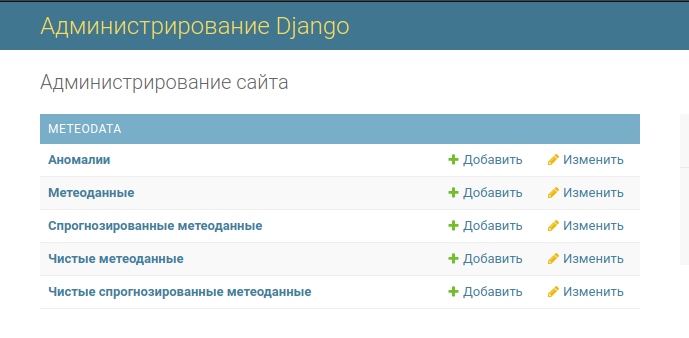


Рисунок 3.8 – Вікно панелі адміністратора

Головними складовими функціями адміністратора є додання нового контенту.

Для цього розроблено спеціалізовані форми, які дозволяють спрощувати цей процес та роботи його зрозумілим для кожного, рис. 3.9.

Це виконано з розрахунку на те, що в більшості метеорологічних центрів, адмініструванням контенту займаються прості метеорологи, які не мають спеціалізованого досвіду роботи з адміністрування сайтів.

Отже це було одним з головних критеріїв під час розробки панелі адміністратора.

Вона повинна бути як можливо спрощеною та відповідати усім вимогам до відповідного програмного засобу.

Форма додання нового запису метеоданих включає в себе:

* Id це поле целочисленном типу зберігає унікальний номер запису;
* Datetime це підлозі в якому зберігається дата і времм коли було зроблено запис;
* Place це поле в якому зберігається назва регіону в якому був зроблений запис;
* placeNumber це поле в якому зберігається номер регіону в якому був зроблений запис;
* temperature поле температури яка була зафіксована в момент запису;
* wind\_way поле зберігає напрямок вітру;
* wind\_speed поле зберігає значення швидкості вітру;
* air\_pressure поле зберігає значення атмосферного тиску;
* water\_pressure поле зберігає значення атмосферного тиску на рівні моря.

Основні джерела містять інформацію подібному форматі, різні інтернет ресурси закордонних спеціалізованих метеоцентри, що базуються в провідних університетах і великих компаніях. Це оптимальний набір даних для аналізу системою, який дає можливість легко збирати дані. Також різні архіви метеоданих містять всі вибрані для аналізу даної системою дані, що є важливим критерієм вибору, бо дає доступ до майже нескінченним ресурсів зборів Європейського метеоцентри.

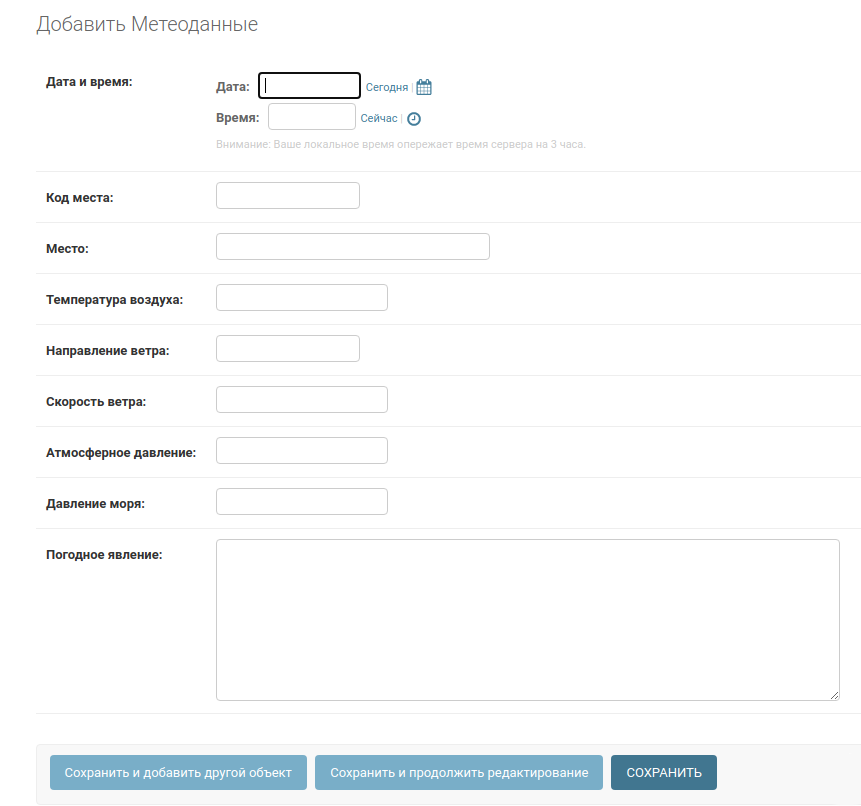


Рисунок 3.9 – Вікно додання нового запису метеоданих

Якщо необхідно відредагувати вже існуючий запис, то його форма буде максимально відповідати нормам змін. Усі додані до системи записи виводяться користувачам у вигляді переліків, що є нормою для перегляду великої кількості контенту схожого типу. Вікно форми редагування інформації зображено на рис. 3.10.

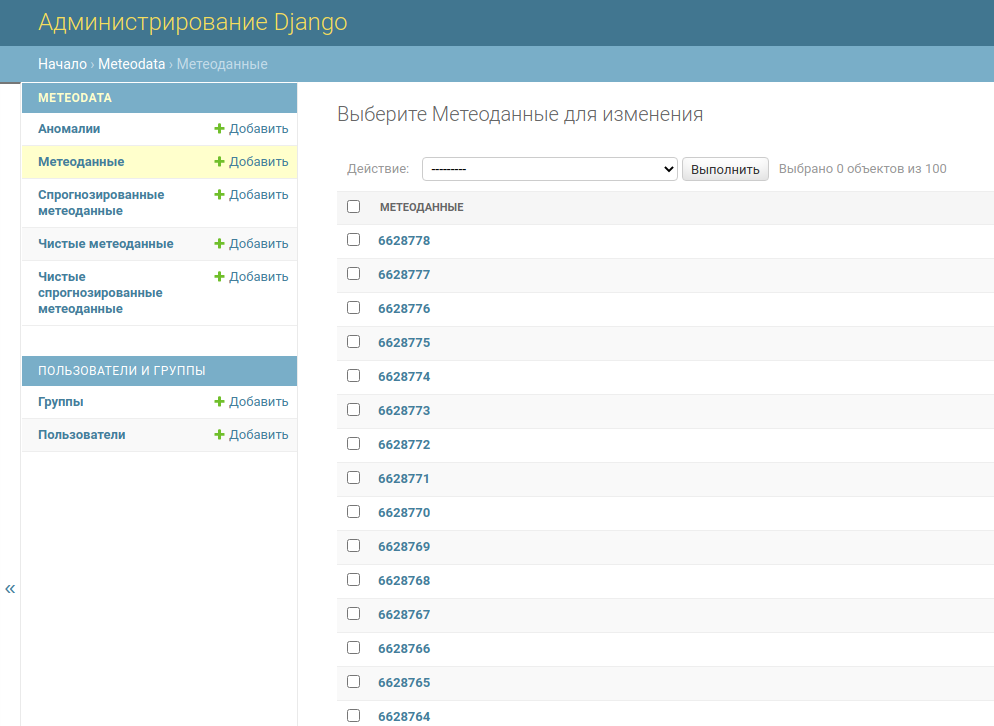


Рисунок 3.10 – Вікно редагування існуючих турів

Також не менш необхідною формою є форма для додавання, редагування та видалення аномалій. Дана форма складається зі спеціальних полів і в неї необхідно внести наступні дані:

* Id поле зберігає унікальний номер запису.
* Meteodata\_Id містить унікальний номер запису з таблиці метеоданнх.
* Fieldname містить назву поля в якому була обноружена аномалія.
* Value зберігає в собі значення поля в якому була виявлена аномалія.
* Anomaly містить кратне опис аномалії або її тип.

Усі інші форми редагування чи додання інформації розроблені аналогічним образом. Отже їх опис не є обов’язковим. Після розгляду функціональної складової панелі адміністратора, необхідно зробити опис користувацької частини.

Інтерфейс сайту виконаний в стилі подібному месенджер Telegram, який є еталонним з боку простоти дизайну і його приємності оці. Стиль спокійний і «однознанчний», його сособенності в тому що полььзователю зрозуміло як користуватися елементами інтерфейсу і якого «поведеднія» від них чекати.

Базова сторінка є сторінкою перегляду метеоданих, вона складається з декількох соновном елементів. Перший елемент необхідний лоя роботи з системою це вершняя панель переходу між сторінками. На цій панелі розташовані три кнопки для переходу між сторінками перегляду метеоданих, перегляду прогнозів і перегляду аномалій шляхом натискання кнопок «перегляд метеоданих», «прогнозування метеоданих» і «Аналіз метеоданих» відповідно. Кнопка «Оновити дані» запускає основний алгоритм обробки даних, це основна кнопка по роботі з системою, рис. 3.11.

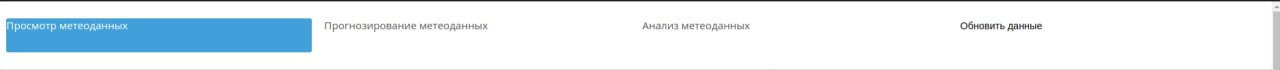


Рисунок 3.11 – Приклад зовнішнього вигляду верхній панелі управління

Другим важливим елементом на кожній сторінці явлется основний блок, в ньому може розташовуватися найрізноманітніша інформація, в даному випадку в ньому розташовані таблиці з даними для вільного їх перегляду користувачем. Використовувані таблиці мають панель управління, кожна робоча таблиця своя і саму таблицю в якій відображені дані. Дані в таблиці розбиті по стовпцях з іменами параметрів виведені в рядках нижче, при натисканні на назву парамтера проводиться сортування, а при подвійному натисканні зворотна сортування по параметру тексту, утримуючи кнопки заголовка. Інформація в таблиці виводитися в своїх окремих осередках, в центрі кожної з яких розміщений текст зі значенням параменту стовпця в цьому рядку записи, рис.3.12.

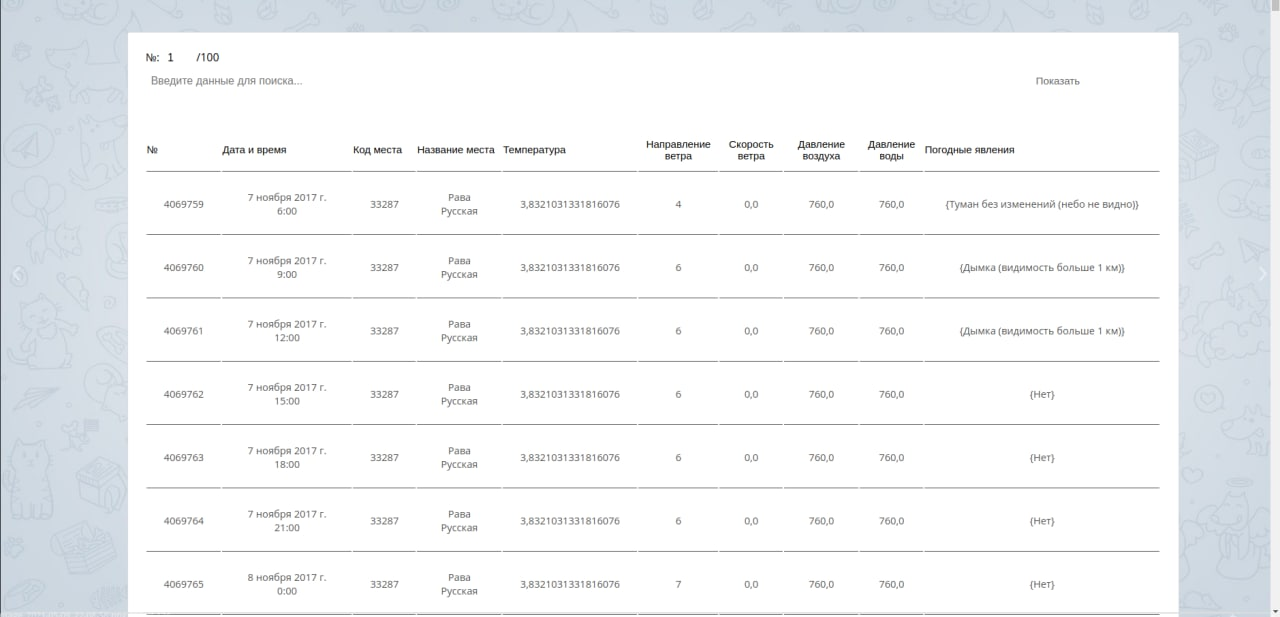


Рисунок 3.12 – Таблиця для відображеня даних.

Панель управління таблицею складається з рядка відображення поточної сторінки і максимального номера станиці. Даний елемент являє собою акуратну рядок тексту.

Також на панелі управління розташована рядок пошуку і фільтрації даних, що виводяться і кнопка застосувати.

Частково прозора рядок введення необхідна для пошуку в даних і їх фільтрації, для цього достатньо ввести якийсь текст і натиснути кнопку застосувати. Пошук осущетсвляется за всіма параметрами записи, рис. 3.13.



Рисунок 3.13 – Приклад зовнішнього вигляду панелі фільтрації таблиць.

Описані вище елементи інтерфейсу повторюються на кожній сторінці системи і в детальному описі не потребують. Інтерфейс виконаний в мінімалістичному дизайні щоб мінімально відволікати користувача від роботи з системою. Система розроблялася з ідеєю максимально швидкої роботи з нею [22].

## Практичне використання інформаційної системи

З метою перевірки роботоспосібності розробленої інформаційної системи, необхідно побудувати тестовий приклад, який дозволить проаналізувати ефективність роботи системи, а також забезпечить зворотній зв’язок щодо розробленого функціоналу. Спочатку необхідно обрати набір вхідних критерієв, на яких буде базуватися тестовий приклад, рис. 3.14.

Для базового прикладу функціональності програми досить перейти на сторінку перегляду метеоданих і натиснути на кнопку «Оновити дані». Після цього і будуть завантажені всі початкові дані необхідні для роботи алгоритмів і настройки всіх використовуваних нейромереж. З боку користувача потрібно тільки очікування, весь подальший процес польностью автоматизований і не вимагає додаткових втручань людини. Першим етапом буде то що система завантажить інформацію про погоду, в разі якщо дані частково були завантажені раніше то система оновить їх вибравши тільки ті що з'явилися з дати останнього запису в системі. Далі дані передаються на обробку нейромережі з пошуку аномалій, та записує їх в два списки «чистий», в якому знаходяться всі записи які пройшли перевірку і сирої, в нього входять всі отримані інформацію про погоду, також знайдені аномалії заносяться в окрему таблицю в базі.

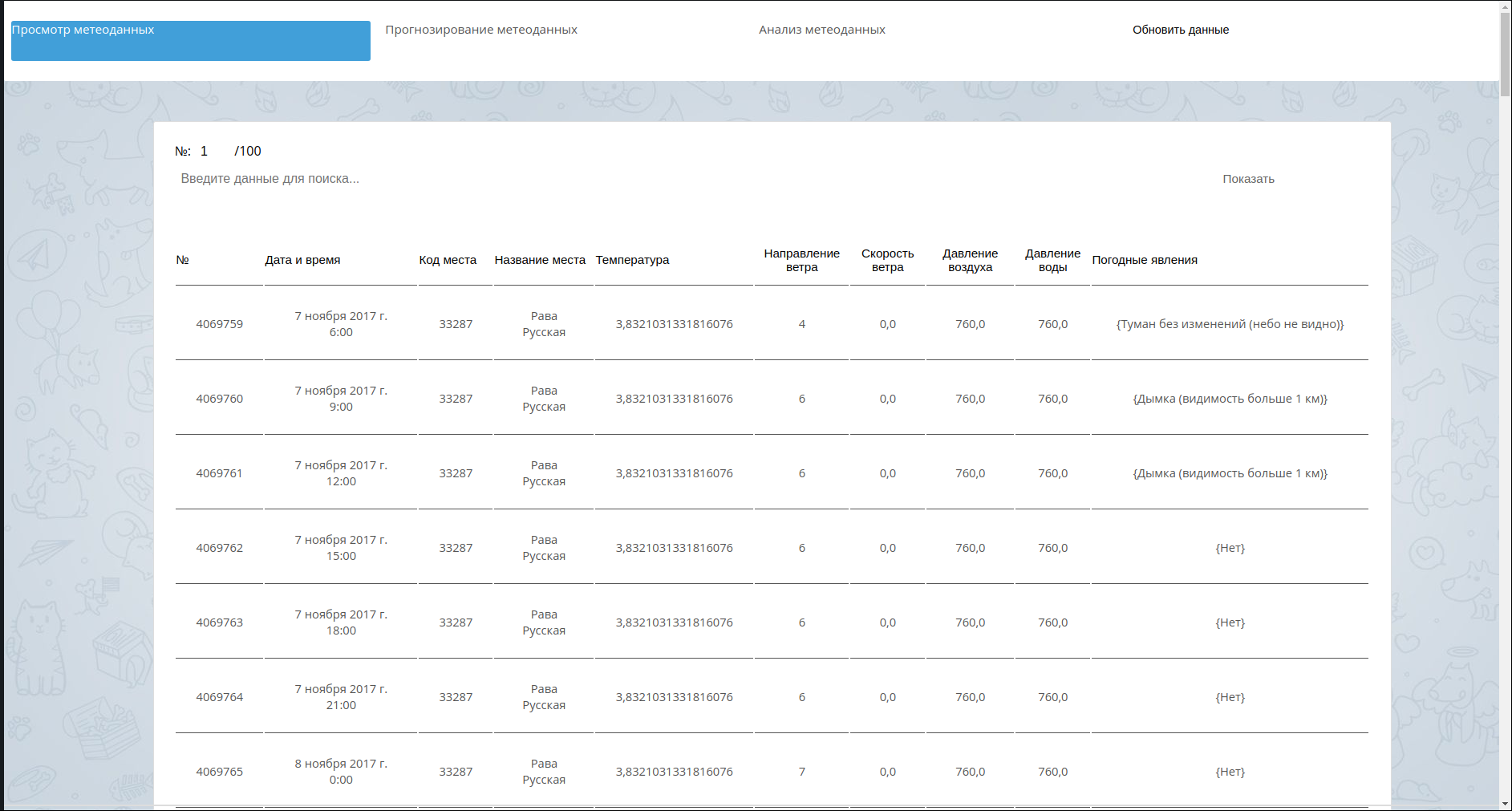


Рисунок 3.14 – Вікно перегляду метеоданих

Наступним етапоя вляется підготовка нейромереж для прогнозування, їх створюється дві копії, одна для повного списку а друга пада тільки для чистого списку. Після підготовки нейромереж обидві пари проходять свої списки і будують прогноз метеоданих і погодних явищ. В процесі система збирає статистику алгоритмів прогнозування для подальшого порівняння ефективності алгоритмів. В кінці цього процесу відбувається запість розрахованих прогнозів базу даних, прогноз пари робила розрахунок на чистих даних записується в список чистого прогнозва, прогноз другої пари в повний список прогнозів відповідно. Закінчення алгоритму в тому що користувача переносять на відповідну сторінку, на якій зображена стовпчаста діаграма з коефіцієнтами точності алгоритмів, кількість отриманих записів і число знайдених в них аномалій а також час скільки зайняв весь процес розрахунку.

Всі дані які були отримані в процесі записані у відповідні таблиці і можуть бути переглянуті користувачем пізніше. Головна перевага даної системи в тому що вона дозволяє легко перевірити алгоритми прогнозування і пошуку аномалій. При постановці подібних задачь користувачеві досить натиснути одну кнопку а для перевірки різних алгоритмів обробки і аналізу досить підмінити заздалегідь заготовлені файли нейромереж.

## Висновки до третього розділу

Проаналізувавши різні методи розробки інформаційних систем, а також засоби реалізації, було спроектовано сучасне програмне рішення, яке може впроваджуватися в метеорологічні центри та використовуватися як самостійна система, або як додатковий модуль з розширенням функціональних можливостей системи. При проектуванні інформаційної системи було визначено головні можливості системи, а також її функціональні особистості.

Головною складовою інформаційної системи є модуль обробки та аналізу даних, конкретно системи пошуку аномалій та побудудвання прогнозу. Система фільтрації та пошуку побудована з використанням різних пошукових критеріїв, які оброблюються різними спеціалізованими туристичними базами даних.

Додатковим програмним модулем є реалізація системи збору метеорологічних даних. Модуль опитує зовнішні джерела на предмет нових метеоданих та завантажуе їх по конкретним запитам.

Розроблена інформаційна система може використовуватися в якості самостійного програмного засобу аналізу та спрощення роботи метеорологів, а також як додатковий програмний модуль, який може бути впроваджено до існуючої інформаційної системи метеорологічних центрів.

# ВИСНОВКИ

В результаті отриманих знань, і додатково вивчених матеріалів були виконані всі вимоги щодо розробки програмного комплексу для збору, обробки та аналізу метеоданих. Розроблене програмне забезпечення дозволяє збирати метеодані з різних джерел, шукати в них аномалії та робити по даним прогноз. Під час написання роботи, було виконано:

* аналіз літературних джерел;
* аналіз предметної області;
* аналіз впливу інформаційних технологій на робочі процеси метеорологів;
* обґрунтувано впровадження інформаційних систем в сферу метеорологіх;
* порівняльний аналіз схожих програмних систем;
* визначення програмні засоби реалізації інформаційної системи;
* проектування інформаційної системи;
* розробку інформаційної системи для сфери метеорології.

Розробку програмного комплексу виконано з використанням мови програмування Python та спеціалізованого фреймворку для роботи з web-системами Django. В якості системи керування базами даних використано PostgreSQL.

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 8302:2015 «Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання» – Чинний від 04.03.2016. – Київ: УкрНДНЦ, 2016. – 20 с.
2. Антоні С. Опануй самостійно програмування за 21 день / С. Антоні. – Москва: Вильямс, 2017. – 562 с.
3. Аронов І.З. Сучасні проблеми безпеки технічних систем і аналізу ризику / І.З. Аронов. – Москва: Інтуит. – 451 с.
4. Астелс Д. Керівництво з екстремального програмування / Д. Астелс. – Москва: Вільямc, 2018. – 468 с.
5. Багриновский К.А. Нові інформаційні технології / К.А. Багриновский, Є.Ю. Хрустальов. – Москва: ЕКО, 2015. – 212 с.
6. Гмурман В.Є. Теорія ймовірностей і математична статистика / В.Є. Гмурман. – Москва: Вища. Шк, 2020. – 479с.
7. Бахтізін В.В. Технології розробки програмного забезпечення / В.В. Бахтізін, Л.А. Глухова. – Мінськ: БДУІР, 2016. – 267 с.
8. Бенін Д.М. Системи підтримки прийняття рішень / В.Л. Сніжко, Д.М. Бенін – Москва: Тріада, 2019. – 165 с.
9. Бхаграва А. Грокаем алгоритми / А. Бхаграва – СПб .: ПИТЕР, 2020. – 288с
10. Вірт Н. Алгоритми і структури даних. / Н. Вірт – Москва: Свет, 2015. – 651 с.
11. Владимиров В.А. Управління ризиком / В.А. Владимиров, Ю.Л. Воробйов, Г.Г. Малинецкий. – Москва: ЮНИТИ, 2016. – 352 с.
12. Шаруп Ч. Х. Укус пітона / Ч. Х. Шаруп – Москва: ЮНИТИ, 2019 – 152с.
13. Дронов В. Django 3 практика створення веб-сайтів на python / В. Дронов – СПб.: «БХВ-Петербург», 2019 – 712с
14. Прохоренок Н.А. Python 3 и PyQt 5. Разработка приложений / Н.А. Прохоренок, В.А. Дронов. – СПб.: БХВ-Петербург, 2016. – 832c.
15. Padmanabhan T.R. Programming with Python / T.R. Padmanabhan. – СПб.: Springer, 2016. – 349p.
16. Эрик М. Изучаем Python. Программирование игр, визуализация данных, веб-приложения / М. Эрик. – СПб.: Питер, 2017. – 496c.
17. Коггзолл Д. PHP 5. Полное руководство / Д. Коггзолл. – Москва: Диалектика, 2006. – 752 с.
18. Бейлин Л. Изучаем PHP и MySQL / Л. Бейлин. – Москва: Эксмо, 2010. – 1060 с.
19. Lane D. Web Database Application with PHP and MySQL / D. Lane. – New Jersey: O'Reilly, 2004. – 816 p.
20. Прохоренок Н.А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н.А. Пархоменко. – СПб.: Питер, 2015. – 768 с.
21. Роббинс Д. HTML5, CSS3 и JavaScript. Исчерпывающее руководство / Д. Роббинс. – Москва: Эксмо, 2014. – 528 с.
22. Шварц Б. MySQL. Оптимизация производительности / Б. Шварц. – Москва: Символ-Плю, 2010. – 483 с.
23. Гольцман В. MySQL 5.0 / В.Гольцман. – Москва: Питер, 2009. – 764 с.
24. Яргер Р.Д. MySQL и mSQL. Базы данных для небольших предприятий и Интернета / Р.Д. Яргер. – Москва: Символ-Плюс, 2010. – 929 с.
25. Уильман Л. MySQL. Руководство по изучению языка / Л.Уильман. – Москва: ДМК Пресс, 2004. – 764 с.
26. Аткинсон Л. MySQL. Библиотека профессионала / Л.Аткинсон. – Москва: Вильямc, 2002. – 1493 с.
27. Артеменко Ю. Н. MySQL. Справочник по язику / Ю.Н. Артеменко. –Москва: Вильямс, 2005. – 843 с.
28. Никсон Р. «Learning PHP, MySQL, JavaScript, CSS & HTML5 A Step-by-Step Guide to Creating Dynamic Websites» / Р. Никсон. – O'Reilly Media, 2012. – 730 с.
29. Rocha G.D. Learning SQLite for iOS / G.D. Rocha. – NY.: [Packt Publishing](https://www.packtpub.com/application-development/learning-sqlite-ios), 2016. – 579 р.
30. Owens M. The Definitive Guide to SQLite / M. Owens, G. Allen. – NY.: Apress, 2010. – 739 р.

ДОДАТОК А

Лістинг коду інформаційної системи

from ..models import Meteodata, ClearMeteodata, ForecastMeteodata, ClearForecastMeteodata, MeteodataAnomalies

from django.db.models import Q

class DataReader():

def \_\_init\_\_(self):

self.\_rows\_count = 50

self.\_meteodata\_page = 1

self.\_meteodata\_search = ''

self.\_meteodata = Meteodata.objects.all()

self.\_clear\_meteodata\_page = 1

self.\_clear\_meteodata\_search = ''

self.\_clear\_meteodata = ClearMeteodata.objects.all()

self.\_forecast\_page = 1

self.\_forecast\_search = ''

self.\_forecast = ForecastMeteodata.objects.all()

self.\_clear\_forecast\_page = 1

self.\_clear\_forecast\_search = ''

self.\_clear\_forecast = ClearForecastMeteodata.objects.all()

self.\_anomaly\_page = 1

self.\_anomaly\_search = ''

self.\_anomaly = MeteodataAnomalies.objects.all()

self.\_firstOrderName = ''

self.\_secondOrderName = ''

self.\_thirdOrderName = ''

self.\_fothOrderName = ''

self.\_fifthOrderName = ''

print(type(self.\_meteodata))

self.\_updateData(1)

self.\_updateData(2)

self.\_updateData(3)

self.\_updateData(4)

self.\_updateData(5)

print(type(self.\_meteodata))

def set\_page(self,tableId, num):

print('set\_page')

print(type(self.\_meteodata))

if tableId == 1:

self.\_meteodata\_page = num

self.\_updateData(1)

elif tableId == 2:

self.\_clear\_meteodata\_page = num

self.\_updateData(2)

elif tableId == 3:

self.\_forecast\_page = num

self.\_updateData(3)

elif tableId == 4:

self.\_clear\_forecast\_page = num

self.\_updateData(4)

elif tableId == 5:

self.\_anomaly\_page = num

self.\_updateData(5)

def search(self, tableId, search):

print('search')

print(type(self.\_meteodata))

temp = 0

if tableId == 1:

self.\_meteodata\_page = 1

if search != '':

self.\_meteodata\_search = search

self.\_meteodata = self.\_meteodata.filter(

Q(id\_\_contains=search) |

Q(datetime\_\_contains=search) |

Q(place\_\_contains=search) |

Q(placeName\_\_contains=search) |

Q(temperature\_\_contains=search) |

Q(wind\_way\_\_contains=search) |

Q(wind\_speed\_\_contains=search) |

Q(air\_pressure\_\_contains=search) |

Q(water\_pressure\_\_contains=search) |

Q(weather\_\_contains=search)

)

else:

self.\_meteodata\_search = ''

self.\_meteodata = Meteodata.objects.all()

self.\_updateData(1)

elif tableId == 2:

self.\_clear\_meteodata\_page = 1

if search != '':

self.\_clear\_meteodata\_search = search

self.\_clear\_meteodata = self.\_clear\_meteodata.filter(

Q(id\_\_contains=search) |

Q(datetime\_\_contains=search) |

Q(place\_\_contains=search) |

Q(placeName\_\_contains=search) |

Q(temperature\_\_contains=search) |

Q(wind\_way\_\_contains=search) |

Q(wind\_speed\_\_contains=search) |

Q(air\_pressure\_\_contains=search) |

Q(water\_pressure\_\_contains=search) |

Q(weather\_\_contains=search)

)

else:

self.\_clear\_meteodata\_search = ''

self.\_clear\_meteodata = ClearMeteodata.objects.all()

self.\_updateData(2)

elif tableId == 3:

self.\_forecast\_page = 1

if search != '':

self.\_forecast\_search = search

self.\_forecast = self.\_forecast.filter(

Q(id\_\_contains=search) |

Q(datetime\_\_contains=search) |

Q(place\_\_contains=search) |

Q(placeName\_\_contains=search) |

Q(temperature\_\_contains=search) |

Q(wind\_way\_\_contains=search) |

Q(wind\_speed\_\_contains=search) |

Q(air\_pressure\_\_contains=search) |

Q(water\_pressure\_\_contains=search) |

Q(weather\_\_contains=search)

)

else:

self.\_forecast\_search = ''

self.\_forecast = ForecastMeteodata.objects.all()

self.\_updateData(3)

elif tableId == 4:

self.\_clear\_forecast\_page = 1

if search != '':

self.\_clear\_forecast\_search = search

self.\_clear\_forecast = self.\_clear\_forecast.filter(

Q(id\_\_contains=search) |

Q(datetime\_\_contains=search) |

Q(place\_\_contains=search) |

Q(placeName\_\_contains=search) |

Q(temperature\_\_contains=search) |

Q(wind\_way\_\_contains=search) |

Q(wind\_speed\_\_contains=search) |

Q(air\_pressure\_\_contains=search) |

Q(water\_pressure\_\_contains=search) |

Q(weather\_\_contains=search)

)

else:

self.\_clear\_forecast\_search = ''

self.\_clear\_forecast = ClearForecastMeteodata.objects.all()

self.\_updateData(4)

elif tableId == 5:

self.\_anomaly\_page = 1

if search != '':

self.\_anomaly\_search = search

self.\_anomaly = self.\_anomaly.filter(

Q(id\_\_contains=search) |

Q(meteodata\_id\_\_contains=search) |

Q(fieldname\_\_contains=search) |

Q(value\_\_contains=search) |

Q(anomaly\_\_contains=search)

)

else:

self.\_anomaly\_search = ''

self.\_anomaly = MeteodataAnomalies.objects.all()

self.\_updateData(5)

def get\_max\_pages(self, tableId):

print('get\_max\_pages')

print(type(self.\_meteodata))

size = 0

if tableId == 1:

size = self.\_meteodata.values('id').count()

elif tableId == 2:

size = self.\_clear\_meteodata.values('id').count()

elif tableId == 3:

size = self.\_forecast.values('id').count()

elif tableId == 4:

size = self.\_clear\_forecast.values('id').count()

elif tableId == 5:

size = self.\_anomaly.values('id').count()

val = int(size // self.\_rows\_count)

if size % self.\_rows\_count != 0:

val += 1

return val

def sort(self, tableId, dataName):

print('sort data')

print(type(self.\_meteodata))

if tableId == 1:

if self.\_firstOrderName == '' or self.\_firstOrderName != dataName:

self.\_meteodata = self.\_meteodata.order\_by(dataName)

self.\_firstOrderName = dataName

else:

self.\_meteodata = self.\_meteodata.order\_by(str('-' + self.\_firstOrderName))

elif tableId == 2:

if self.\_secondOrderName == '' or self.\_secondOrderName != dataName:

self.\_clear\_meteodata = self.\_clear\_meteodata.order\_by(dataName)

self.\_secondOrderName = dataName

else:

self.\_meteodata = self.\_meteodata.order\_by(str('-' + self.\_secondOrderName))

elif tableId == 3:

if self.\_thirdOrderName == '' or self.\_thirdOrderName != dataName:

self.\_forecast = self.\_forecast.order\_by(dataName)

self.\_thirdOrderName = dataName

else:

self.\_forecast = self.\_forecast.order\_by(str('-' + self.\_thirdOrderName))

elif tableId == 4:

if self.\_fothOrderName == '' or self.\_fothOrderName != dataName:

self.\_clear\_forecast = self.\_clear\_forecast.order\_by(dataName)

self.\_fothOrderName = dataName

else:

self.\_clear\_forecast = self.\_clear\_forecast.order\_by(str('-' + self.\_fothOrderName))

elif tableId == 5:

if self.\_fifthOrderName == '' or self.\_fifthOrderName != dataName:

self.\_anomaly = self.\_anomaly.order\_by(dataName)

self.\_fifthOrderName = dataName

else:

self.\_anomaly = self.\_anomaly.order\_by(str('-' + self.\_fifthOrderName))

def \_updateData(self, tableId):

print('updateData')

print(type(self.\_meteodata))

if tableId == 1:

self.\_meteodata = self.\_meteodata[(self.\_meteodata\_page \* self.\_rows\_count):(self.\_meteodata\_page \* self.\_rows\_count) + self.\_rows\_count]

elif tableId == 2:

self.\_clear\_meteodata = self.\_clear\_meteodata[(self.\_clear\_meteodata\_page \* self.\_rows\_count):(self.\_clear\_meteodata\_page \* self.\_rows\_count) + self.\_rows\_count]

elif tableId == 3:

self.\_forecast = self.\_forecast[(self.\_forecast\_page \* self.\_rows\_count):(self.\_forecast\_page \* self.\_rows\_count) + self.\_rows\_count]

elif tableId == 4:

self.\_clear\_forecast = self.\_clear\_forecast[(self.\_clear\_forecast\_page \* self.\_rows\_count):(self.\_clear\_forecast\_page \* self.\_rows\_count) + self.\_rows\_count]

elif tableId == 5:

self.\_anomaly = self.\_anomaly[(self.\_anomaly\_page \* self.\_rows\_count):(self.\_anomaly\_page \* self.\_rows\_count) + self.\_rows\_count]

print(type(self.\_meteodata))

from django.db import models

class Meteodata(models.Model):

id = models.BigAutoField("id", primary\_key=True)

datetime = models.DateTimeField("Дата и время")

place = models.BigIntegerField("Код места")

placeName = models.CharField("Место", max\_length=80)

temperature = models.FloatField("Температура воздуха")

wind\_way = models.BigIntegerField("Направление ветра")

wind\_speed = models.FloatField("Скорость ветра")

air\_pressure = models.FloatField("Атмосферное давление")

water\_pressure = models.FloatField("Давление моря")

weather = models.TextField("Погодное явление")

def \_\_str\_\_(self):

return self.id

class Meta:

verbose\_name = "Метеоданные"

verbose\_name\_plural = "Метеоданные"

class ClearMeteodata(models.Model):

id = models.BigAutoField("id", primary\_key=True)

datetime = models.DateTimeField("Дата и время")

place = models.BigIntegerField("Код места")

placeName = models.CharField("Место", max\_length=80)

temperature = models.FloatField("Температура воздуха")

wind\_way = models.BigIntegerField("Направление ветра")

wind\_speed = models.FloatField("Скорость ветра")

air\_pressure = models.FloatField("Атмосферное давление")

water\_pressure = models.FloatField("Давление моря")

weather = models.TextField("Погодное явление")

def \_\_str\_\_(self):

return self.id

class Meta:

verbose\_name = "Чистые метеоданные"

verbose\_name\_plural = "Чистые метеоданные"

class ForecastMeteodata(models.Model):

id = models.BigAutoField("id", primary\_key=True)

datetime = models.DateTimeField("Дата и время")

place = models.BigIntegerField("Код места")

placeName = models.CharField("Место", max\_length=80)

temperature = models.FloatField("Температура воздуха")

wind\_way = models.BigIntegerField("Направление ветра")

wind\_speed = models.FloatField("Скорость ветра")

air\_pressure = models.FloatField("Атмосферное давление")

water\_pressure = models.FloatField("Давление моря")

weather = models.TextField("Погодное явление")

def \_\_str\_\_(self):

return self.id

class Meta:

verbose\_name = "Спрогнозированные метеоданные"

verbose\_name\_plural = "Спрогнозированные метеоданные"

class ClearForecastMeteodata(models.Model):

id = models.BigAutoField("id", primary\_key=True)

datetime = models.DateTimeField("Дата и время")

place = models.BigIntegerField("Код места")

placeName = models.CharField("Место", max\_length=80)

temperature = models.FloatField("Температура воздуха")

wind\_way = models.BigIntegerField("Направление ветра")

wind\_speed = models.FloatField("Скорость ветра")

air\_pressure = models.FloatField("Атмосферное давление")

water\_pressure = models.FloatField("Давление моря")

weather = models.TextField("Погодное явление")

def \_\_str\_\_(self):

return self.id

class Meta:

verbose\_name = "Чистые спрогнозированные метеоданные"

verbose\_name\_plural = "Чистые спрогнозированные метеоданные"

class MeteodataAnomalies(models.Model):

id = models.BigAutoField("id", primary\_key=True)

meteodata\_id = models.BigIntegerField("Код записи")

fieldname = models.TextField("Название поля")

value = models.TextField("Значение")

anomaly = models.TextField("Аномалия")

class Meta:

verbose\_name = "Аномалии"

verbose\_name\_plural = "Аномалии"

def \_\_str\_\_(self):

return self.name

def get\_absolute\_url(self):

return reverse("\_detail", kwargs={"pk": self.pk})

import psycopg2

from datetime import date

class DBControl():

def \_\_init\_\_(self):

self.\_connect()

self.\_tables = ('meteodata', 'forecastmeteodata', 'clearmeteodata', 'clearforecastmeteodata', 'meteodataanomalies')

self.\_curTable = 0

self.\_lastQuere = ''

# removers

def removeMeteodata(self, id):

self.\_curs.execute('DELETE from public.meteodata\_meteodata WHERE id ={}'.format(str(id)))

def removeForecast(self, id):

self.\_curs.execute('DELETE from public.meteodata\_forecastmeteodata WHERE id ={}'.format(str(id)))

def removeClearMeteodata(self, id):

self.\_curs.execute('DELETE from public.meteodata\_clearmeteodata WHERE id ={}'.format(str(id)))

def removeClearForecast(self, id):

self.\_curs.execute('DELETE from public.meteodata\_clearforecastmeteodata WHERE id ={}'.format(str(id)))

def removeAnomalies(self, id):

self.\_curs.execute('DELETE from public.meteodata\_meteodataanomalies WHERE id ={}'.format(str(id)))

# inserters

def insertMeteodata(self, data):

\_id = int(self.getMaxId(self.\_tables[2]) + 1)

self.\_curs.execute("INSERT INTO public.meteodata\_meteodata VALUES(DEFAULT, '" + data[0] + "', " + data[1] + ", '" + data[2] + "', " + data[3] + ", " + data[4] + ", " + data[5] + ", " + data[6] + ", " + data[7] + ", '{" + data[8] + "}')")

def insertForecast(self, data):

\_id = int(self.getMaxId(self.\_tables[3]) + 1)

self.\_curs.execute("INSERT INTO public.meteodata\_forecastmeteodata VALUES(DEFAULT, '" + data[0] + "', " + data[1] + ", '" + data[2] + "', " + data[3] + ", " + data[4] + ", " + data[5] + ", " + data[6] + ", " + data[7] + ", '{" + data[8] + "}')")

def insertClearMeteodata(self, data):

\_id = int(self.getMaxId(self.\_tables[2]) + 1)

self.\_curs.execute("INSERT INTO public.meteodata\_clearmeteodata VALUES(DEFAULT, '" + data[0] + "', " + data[1] + ", '" + data[2] + "', " + data[3] + ", " + data[4] + ", " + data[5] + ", " + data[6] + ", " + data[7] + ", '{" + data[8] + "}')")

def insertClearForecast(self, data):

\_id = int(self.getMaxId(self.\_tables[3]) + 1)

self.\_curs.execute("INSERT INTO public.meteodata\_clearforecastmeteodata VALUES(DEFAULT, '" + data[0] + "', " + data[1] + ", '" + data[2] + "', " + data[3] + ", " + data[4] + ", " + data[5] + ", " + data[6] + ", " + data[7] + ", '{" + data[8] + "}')")

def insertAnomalies(self, data):

\_id = int(self.getMaxId(self.\_tables[0]) + 1)

self.\_curs.execute("INSERT INTO public.meteodata\_meteodataanomalies VALUES(DEFAULT," + data[0] + ", '{" + data[1] + "}', '{" + data[2] + "}', '{" + data[3] + "}')")

# updaters

def updateMeteodata(self, data):

self.\_curs.execute("UPDATE public.meteodata\_meteodata SET datetime='" + data[0] + "', place=" + str(data[1]) + ", \"placeName\"" + data[2] + "', temperature=" + str(data[3]) + ", wind\_way=" + str(data[4]) + ", wind\_speed=" + str(data[5]) + ", air\_pressure=" + str(data[6]) + ", water\_pressure=" + str(data[7]) + ", weather='{" + data[8] + "}' WHERE id=" + str(data[0]))

def updateForecast(self, data):

self.\_curs.execute("UPDATE public.meteodata\_forecastmeteodata SET datetime='" + data[0] + "', place=" + str(data[1]) + ", \"placeName\"" + data[2] + "', temperature=" + str(data[3]) + ", wind\_way=" + str(data[4]) + ", wind\_speed=" + str(data[5]) + ", air\_pressure=" + str(data[6]) + ", water\_pressure=" + str(data[7]) + ", weather='{" + data[8] + "}' WHERE id=" + str(data[0]))

def updateClearMeteodata(self, data):

self.\_curs.execute("UPDATE public.meteodata\_clearmeteodata SET datetime='" + data[0] + "', place=" + str(data[1]) + ", \"placeName\"" + data[2] + "', temperature=" + str(data[3]) + ", wind\_way=" + str(data[4]) + ", wind\_speed=" + str(data[5]) + ", air\_pressure=" + str(data[6]) + ", water\_pressure=" + str(data[7]) + ", weather='{" + data[8] + "}' WHERE id=" + str(data[0]))

def updateClearForecast(self, data):

self.\_curs.execute("UPDATE public.meteodata\_clearforecastmeteodata SET datetime='" + data[0] + "', place=" + str(data[1]) + ", \"placeName\"" + data[2] + "', temperature=" + str(data[3]) + ", wind\_way=" + str(data[4]) + ", wind\_speed=" + str(data[5]) + ", air\_pressure=" + str(data[6]) + ", water\_pressure=" + str(data[7]) + ", weather='{" + data[8] + "}' WHERE id=" + str(data[0]))

def updateAnomalies(self, data):

self.\_curs.execute("UPDATE public.meteodata\_meteodataanomalies SET meteodata\_id=" + str(data[1]) + ", fieldname='{" + data[2] + "}', value='{" + data[3] + "}', anomaly='{" + data[4] + "}' WHERE id=" + str(data[0]))

# getters

def getMeteodata(self, filter='id, datetime, place, \"placeName\", temperature, wind\_way, wind\_speed, air\_pressure, water\_pressure, weather', where=''):

self.\_curTable = 0

tmp = 'SELECT {} FROM public.meteodata\_meteodata {}'.format(filter, where)

self.\_lastQuere = tmp

self.\_curs.execute(tmp)

records = self.\_refactorRecords(self.\_curs.fetchall())

return records

def getForecast(self, filter='id, datetime, place, \"placeName\", temperature, wind\_way, wind\_speed, air\_pressure, water\_pressure, weather', where=''):

self.\_curTable = 1

tmp = 'SELECT {} FROM public.meteodata\_forecastmeteodata {}'.format(filter, where)

self.\_lastQuere = tmp

self.\_curs.execute(tmp)

records = self.\_refactorRecords(self.\_curs.fetchall())

return records

def getClearMeteodata(self, filter='id, datetime, place, \"placeName\", temperature, wind\_way, wind\_speed, air\_pressure, water\_pressure, weather', where=''):

self.\_curTable = 3

tmp = 'SELECT {} FROM public.meteodata\_clearmeteodata {}'.format(filter, where)

self.\_lastQuere = tmp

self.\_curs.execute(tmp)

records = self.\_refactorRecords(self.\_curs.fetchall())

return records

def getClearForecast(self, filter='id, datetime, place, \"placeName\", temperature, wind\_way, wind\_speed, air\_pressure, water\_pressure, weather', where=''):

self.\_curTable = 4

tmp = 'SELECT {} FROM public.meteodata\_clearforecastmeteodata {}'.format(filter, where)

self.\_lastQuere = tmp

self.\_curs.execute(tmp)

records = self.\_refactorRecords(self.\_curs.fetchall())

return records

def getAnomalies(self, filter='id, meteodata\_id, fieldname, value, anomaly', where=''):

self.\_curTable = 2

tmp = 'SELECT {} FROM public.meteodata\_meteodataanomalies {}'.format(filter, where)

self.\_lastQuere = tmp

self.\_curs.execute(tmp)

records = self.\_refactorRecords(self.\_curs.fetchall())

return records

# Additional functions

def getTableNames(self):

return self.\_tables

def getMaxId(self, table):

if table in self.\_tables:

self.\_curs.execute('SELECT MAX(id) FROM public.{}'.format(table))

elif isinstance(table, int):

if table >= 0 and table < len(self.\_tables):

self.\_curs.execute('SELECT MAX(id) FROM public.{}'.format(self.\_tables[table]))

else:

return 0

records = self.\_curs.fetchall()

if records[0][0] is None:

return 1

else:

return int(records[0][0])

def getHederLabels(self):

if self.\_curTable == 0:

return ['№', 'Дата и время', 'Код места', 'Название места', 'Температура', 'Направление ветра', 'Скорость ветра', 'Давление воздуха', 'Давление воды', 'Погодные явления']# meteodata\_meteodataanomalies

elif self.\_curTable == 1:

return ['№', 'Дата и время', 'Код места', 'Название места', 'Температура', 'Направление ветра', 'Скорость ветра', 'Давление воздуха', 'Давление воды', 'Погодные явления']# payment\_table

elif self.\_curTable == 2:

return ['№', '№ метеоданных', 'Имя поля', 'Значение', 'Описание аномалии']# MeteodataTable

def \_refactorRecords(self, records):

result = list()

for row in records:

result\_row = list()

for element in row:

if isinstance(element, datetime):

result\_row.append(str(element))

elif isinstance(element, str):

if element.isdigit():

result\_row.append(float(element))

else:

if element[0] == '{':

result\_row.append(element[1: -1])

else:

result\_row.append(element)

elif isinstance(element, list):

result\_row.append(element[0])

elif isinstance(element, int):

result\_row.append(element)

elif isinstance(element, float):

result\_row.append(float(element))

result.append(result\_row)

return result

def save(self):

self.\_conn.commit()

def \_connect(self):

self.\_conn = psycopg2.connect(dbname='Weather', user='postgres', password='12345', host='localhost')

self.\_curs = self.\_conn.cursor()

def \_\_del\_\_(self):

self.\_conn.commit()

self.\_curs.close()

self.\_conn.close()

import requests

from bs4 import BeautifulSoup as bs

from bs4 import NavigableString

from calendar import monthrange

import psycopg2

import time

import re

from datetime import date

class MyParser:

def \_\_init\_\_(self):

self.\_rawData = []

self.\_urls = ['https://meteopost.com/weather/archive/']

def StartParse(self):

citySet, data = self.raw\_update1()

return self.prepare\_data1(citySet, data)

def raw\_update1(self, url='https://meteopost.com/weather/archive/', startParseRequest='', endParseRequest=''):

resultList = []

monthSet = {'value': 0, 'text': 0}

yearSet = {'value': 0, 'text': 0}

citySet = {'value': 0, 'text': 0}

badCitySet = {'value': 0, 'text': 0}

# get city

r = requests.get(url)

html = bs(r.content, 'html.parser')

badCitySet['value'], badCitySet['text'] = self.\_getSetData(name='city', html=html, part=0)

citySet['value'], citySet['text'] = self.\_getSetData(name='city', html=html, part=1)

# get month and year

yearSet['value'], yearSet['text'] = self.\_getSetData(name='y', html=html, part=0)

monthSet['value'], monthSet['text'] = self.\_getSetData(name='m', html=html, part=0)

r.close()

startParseFlag = False

for year in yearSet['value']:

y = year

for month in monthSet['value']:

m = month

days = str(monthrange(int(y), int(m))[1])

for city in citySet['value']:

city = city

req = 'd=1&m={}&y={}&city={}&arc=2&days={}'.format(m, y, city, days)

print(req)

hd = {

"Host": "meteopost.com",

"User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:84.0) Gecko/20100101 Firefox/84.0",

"Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8",

"Accept-Language": "en-US,en;q=0.5",

"Accept-Encoding": "gzip, deflate, br",

"Content-Type": "application/x-www-form-urlencoded",

"Content-Length": "39",

"Origin": "https://meteopost.com",

"DNT": "1",

"Connection": "keep-alive",

"Referer": "https://meteopost.com/weather/archive/",

"Cookie": "\_ga=GA1.2.1952077172.1608990993; \_gid=GA1.2.2059205127.1609852737; \_gat=1",

"Upgrade-Insecure-Requests": "1",

"TE": "Trailers"

}

if req == startParseRequest or startParseRequest == '':

startParseFlag = True

if not startParseFlag:

continue

if req == endParseRequest:

return

time.sleep(0.15) # not to ddos timer)

r = requests.post(url, data=req.encode(), headers=hd, stream=True)

html = bs(r.content, 'html5lib')

items = html.find\_all("table")

html = items[3]

items = html.find\_all("tr")

for i in range(len(items)):

s = ''

tr = items[i]

if len(tr) > 5:

for child in tr:

x = child.center

try:

img = x.img

num = 0

if img['src'] == '/pic/180.png':

num = 0

elif img['src'] == '/pic/225.png':

num = 1

elif img['src'] == '/pic/270.png':

num = 2

elif img['src'] == '/pic/360.png':

num = 3

elif img['src'] == '/pic/0.png':

num = 4

elif img['src'] == '/pic/45.png':

num = 5

elif img['src'] == '/pic/90.png':

num = 6

elif img['src'] == '/pic/135.png':

num = 7

elif img['src'] == '/pic/00.png':

num = 8

s = s + '\_' + str(num)

except:

try:

b = x.b

s = s + '\_' + b.string

except:

# set \_ symbol for delimiter

if not isinstance(x, type(None)):

s = s + '\_' + x.get\_text()

else:

s = s + '\_=' # if data missing = symbol is placeholder

print('SSS - ', s)

resultList.append(s + '&&' + req + '\n')

r.close()

return citySet, resultList

def \_getSetData(self, name='', html='', part=0):

items = html.find\_all(attrs={'name': name})

temp1 = []

temp2 = []

for child in items[part].children:

if not isinstance(child, NavigableString):

temp1.append(child['value'])

temp2.append(child.string)

return temp1, temp2

def prepare\_data1(self, citySet, data):

resultList = []

r = re.compile('^days=[0-9](2)[\n]$')

i = 0

while True:

line = data[i]

nextLine = data[i+1]

temp = str(line[-1:-9:-1])

if temp[::-1] not in ('days=28\n', 'days=29\n', 'days=30\n', 'days=31\n'):

print(list(temp[::-1]))

data[i] = data[i].rstrip() + nextLine

data.pop(i+1)

i = i + 1

if i + 1 == len(data):

break

i = 0

headers = []

while True:

line = data[i]

baseList = line.split('&&')

if line[0] == '&':

print(baseList[1])

i = i + 1

continue

dataList = baseList[0].split('\_')

querryList = baseList[1].split('&')

if dataList[0] == '':

dataList.pop(0)

if line[:3] == '\_Де':

headers = dataList

i = i + 1

continue

d = dict()

for j in range(len(dataList)):

if dataList[j] != '=':

d.update({headers[j]: dataList[j]})

else:

d.update({headers[j]: 0})

for item in querryList:

temp = item.split('=')

d.update({temp[0]: temp[1]})

if isinstance(d['Темп. Возд'], str):

d['Темп. Возд'] = d['Темп. Возд'][:-1]

if isinstance(d['Скор ветра'], str):

d['Скор ветра'] = d['Скор ветра'].split(' ')

d['Скор ветра'] = d['Скор ветра'][0]

if d['День'] == '0':

d['День'] = '26'

xx = d['Час'].split(':')

if int(xx[0]) > 24:

d['Час'] = '12:00'

resultList.append(d)

i = i + 1

#11-3

#4-10

if i == len(data):

break

return citySet, resultList

from .Deamon.ForecastDeamon import ForecastDeamon

from .MeteodataMiner import MeteodataMiner

from .DataReader import DataReader

class MainMenu:

Deamon = ForecastDeamon()

Miner = MeteodataMiner()

Reader = DataReader()

currentContext = 1

def get\_context():

if MainMenu.currentContext == 1:

context = {

"meteodata\_page": MainMenu.Reader.\_meteodata\_page,

"meteodata\_search": MainMenu.Reader.\_meteodata\_search,

"meteodata\_max\_pages": MainMenu.get\_max\_pages(1),

"meteodata\_top\_labels": MainMenu.get\_top\_labels(),

"meteodata": MainMenu.Reader.\_meteodata,

"clear\_meteodata\_page": MainMenu.Reader.\_clear\_meteodata\_page,

"clear\_meteodata\_search": MainMenu.Reader.\_clear\_meteodata\_search,

"clear\_meteodata\_max\_pages": MainMenu.get\_max\_pages(2),

"clear\_meteodata": MainMenu.Reader.\_clear\_meteodata

}

elif MainMenu.currentContext == 2:

context = {

"forecast": MainMenu.Reader.\_forecast,

"forecast\_page": MainMenu.Reader.\_forecast\_page,

"forecast\_search": MainMenu.Reader.\_forecast\_search,

"forecast\_max\_pages": MainMenu.get\_max\_pages(3),

"forecast\_top\_labels": MainMenu.get\_top\_labels(),

"clear\_forecast": MainMenu.Reader.\_clear\_forecast,

"clear\_forecast\_page": MainMenu.Reader.\_clear\_forecast\_page,

"clear\_forecast\_search": MainMenu.Reader.\_clear\_forecast\_search,

"clear\_forecast\_max\_pages": MainMenu.get\_max\_pages(4),

}

elif MainMenu.currentContext == 3:

context = {

"anomaly": MainMenu.Reader.\_anomaly,

"anomaly\_page": MainMenu.Reader.\_anomaly\_page,

"anomaly\_search": MainMenu.Reader.\_anomaly\_search,

"anomaly\_max\_pages": MainMenu.get\_max\_pages(5),

"anomaly\_top\_labels": MainMenu.get\_top\_labels()

}

context.update(

{

"rows\_count": MainMenu.Reader.\_rows\_count,

"menu\_status": MainMenu.get\_menu\_status(),

"submenu\_status": MainMenu.get\_submenu\_status()

}

)

return context

def set\_page(num, tableId):

MainMenu.Reader.set\_page(num, tableId)

def search(search):

MainMenu.Reader.search(search)

def set\_context(num):

if num >= 1 and num <= 3:

MainMenu.currentContext = num

def data\_update():

print('data\_update')

MainMenu.Miner.updateMeteodata()

def forecast\_update():

print('forecast\_update')

MainMenu.Deamon.update()

def anomaly\_update():

print('anomaly\_update')

MainMenu.Deamon.scanForAnomalies()

def get\_menu\_status():

print('get\_menu\_status')

if MainMenu.currentContext == 1:

return ['button-active', 'button', 'button']

elif MainMenu.currentContext == 2:

return ['button', 'button-active', 'button']

elif MainMenu.currentContext == 3:

return ['button', 'button', 'button-active']

def get\_submenu\_status():

print('get\_submenu\_status')

if MainMenu.currentContext == 1:

if MainMenu.Miner.meteodataUpdateStatus():

return 'hide'

else:

return 'active'

elif MainMenu.currentContext == 2:

if MainMenu.Deamon.forecastUpdateStatus():

return 'hide'

else:

return 'active'

elif MainMenu.currentContext == 3:

if MainMenu.Deamon.anomaliesScanStatus():

return 'hide'

else:

return 'active'

def get\_top\_labels():

print('get\_top\_labels')

if MainMenu.currentContext == 1:

return ['№', 'Дата и время', 'Код места', 'Название места', 'Температура', 'Направление ветра', 'Скорость ветра', 'Давление воздуха', 'Давление воды', 'Погодные явления']

elif MainMenu.currentContext == 2:

return ['№', 'Дата и время', 'Код места', 'Название места', 'Температура', 'Направление ветра', 'Скорость ветра', 'Давление воздуха', 'Давление воды', 'Погодные явления']

elif MainMenu.currentContext == 3:

return ['№', '№ метеоданных', 'Имя поля', 'Значение', 'Описание аномалии']

def get\_max\_pages(tableId):

return MainMenu.Reader.get\_max\_pages(tableId)

def \_make\_forecast\_test():

print('make\_forecast\_test')

MainMenu.Deamon.\_make\_test()

def sort(dataName):

print('sort data')

MainMenu.Reader.sort(MainMenu.currentContext, dataName)

from django.shortcuts import render, redirect

from django.views.generic import ListView

from django.views.generic.base import View

from .models import Meteodata, ForecastMeteodata, MeteodataAnomalies

from django.db.models import Q

from .moduls import MainMenu

class MeteodataView(ListView):

def get(self, request):

MainMenu.set\_context(1)

context = MainMenu.get\_context()

return render(request, "meteodata/meteodataview.html", context=context)

class MeteodataChangePage(View):

def post(self, request):

d = request.POST

print(d)

MainMenu.set\_context(1)

MainMenu.set\_page(int(d['page']))

MainMenu.search(d['search'])

try:

context = MainMenu.get\_context()

return render(request, "meteodata/meteodataview.html", context=context)

except:

return redirect('/meteodata/')

class MeteodataUpdateDataPage(View):

def post(self, request):

d = request.POST

print(d)

MainMenu.set\_context(1)

if not MainMenu.\_is\_data\_updated:

MainMenu.\_data\_update()

return redirect('/meteodata/')

class ForecastView(ListView):

def get(self, request):

MainMenu.set\_context(2)

context = MainMenu.get\_context()

return render(request, "meteodata/forecastview.html", context=context)

class ForecastChangePage(View):

def post(self, request):

d = request.POST

print(d)

MainMenu.set\_context(2)

MainMenu.set\_page(int(d['page']))

MainMenu.search(d['search'])

try:

context = MainMenu.get\_context()

return render(request, "meteodata/forecastview.html", context=context)

except:

return redirect('meteodata/forecast/')

class ForecastUpdateDataPage(View):

def post(self, request):

d = request.POST

print(d)

MainMenu.set\_context(2)

if not MainMenu.\_is\_forecast\_updated:

MainMenu.\_make\_forecast()

return redirect('meteodata/forecast/')

class ForecastUpdateTestResultPage(View):

def post(self, request):

d = request.POST

print(d)

MainMenu.set\_context(2)

MainMenu.\_make\_forecast\_test()

return redirect('meteodata/forecast/')

class AnomalyView(ListView):

def get(self, request):

MainMenu.set\_context(3)

context = MainMenu.get\_context()

return render(request, "meteodata/analysisview.html", context=context)

class AnomalyChangePage(View):

def post(self, request):

d = request.POST

print(d)

MainMenu.set\_context(3)

MainMenu.set\_page(int(d['page']))

MainMenu.search(d['search'])

try:

context = MainMenu.get\_context()

return render(request, "meteodata/analysisview.html", context=context)

except:

return redirect('meteodata/anoomaly/')

class AnomalyUpdateDataPage(View):

def post(self, request):

d = request.POST

print(d)

MainMenu.set\_context(3)

if not MainMenu.is\_anomaly\_updated:

MainMenu.\_make\_anomaly()

return redirect('meteodata/anoomaly/')

ДОДАТОК Б

Слайди презентації



Рисунок Б1 – Титульний слайд презентації



Рисунок Б2 – Слайд з визначенням предметної області

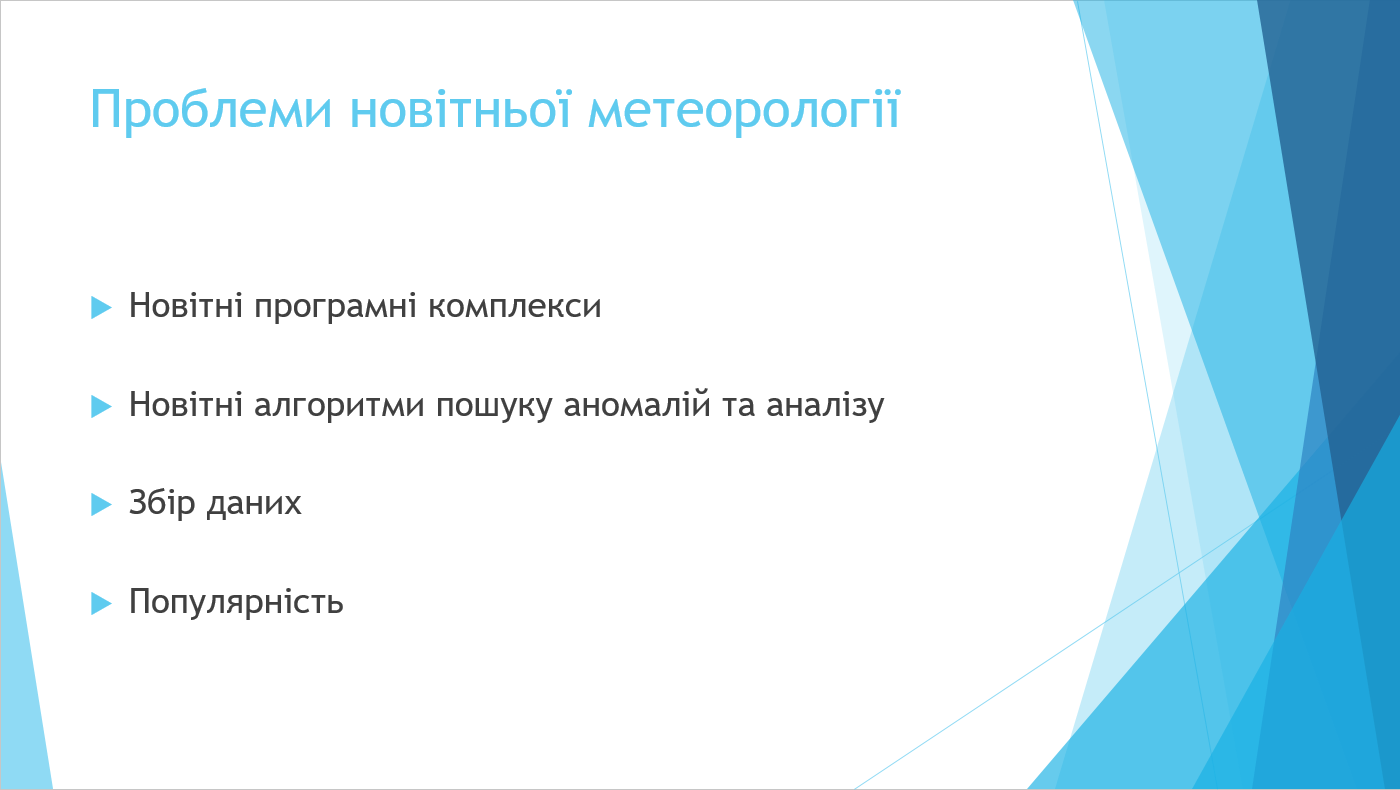


Рисунок Б3 – Слайд з визначеними проблемами предметної області

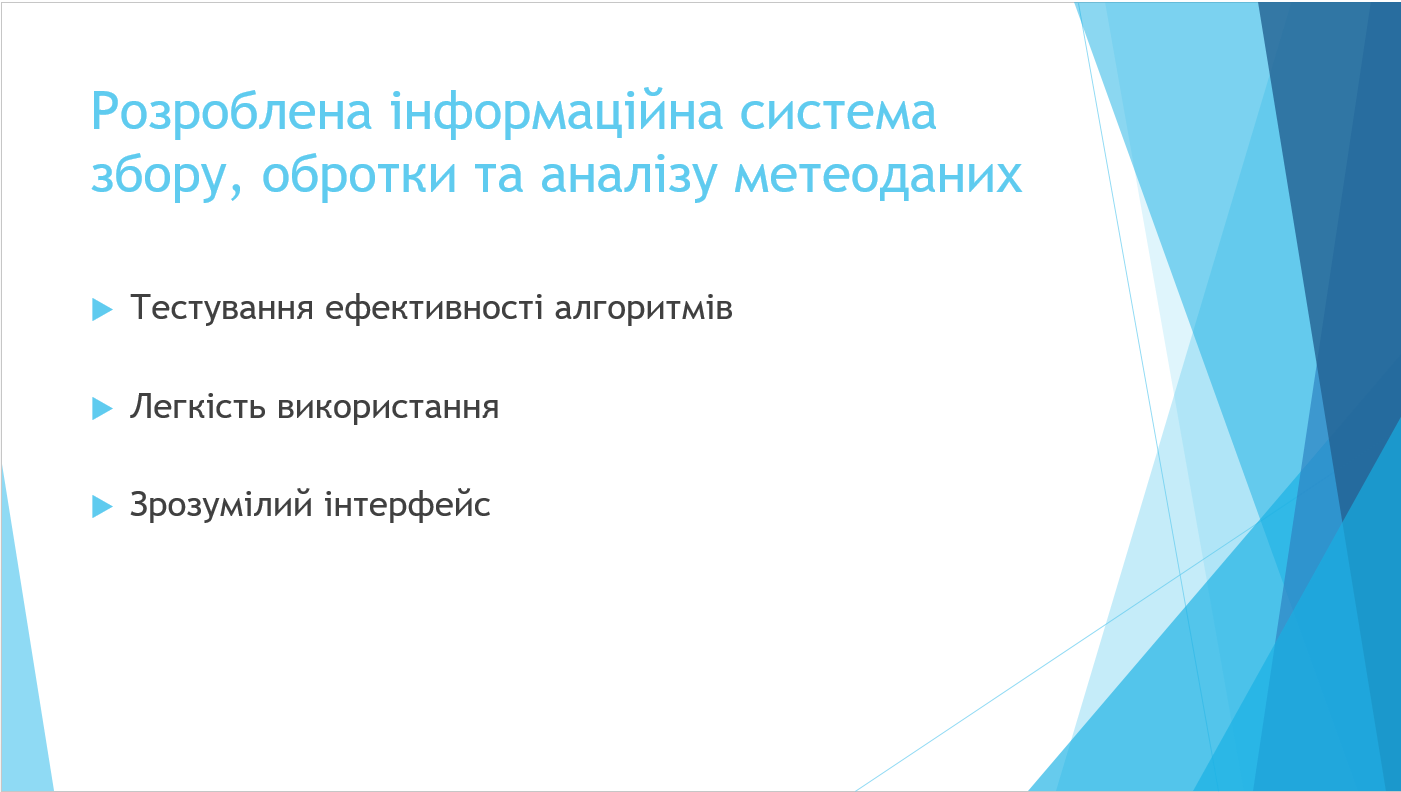


Рисунок Б4 = Слайд з описом функціоналу розробленої системи

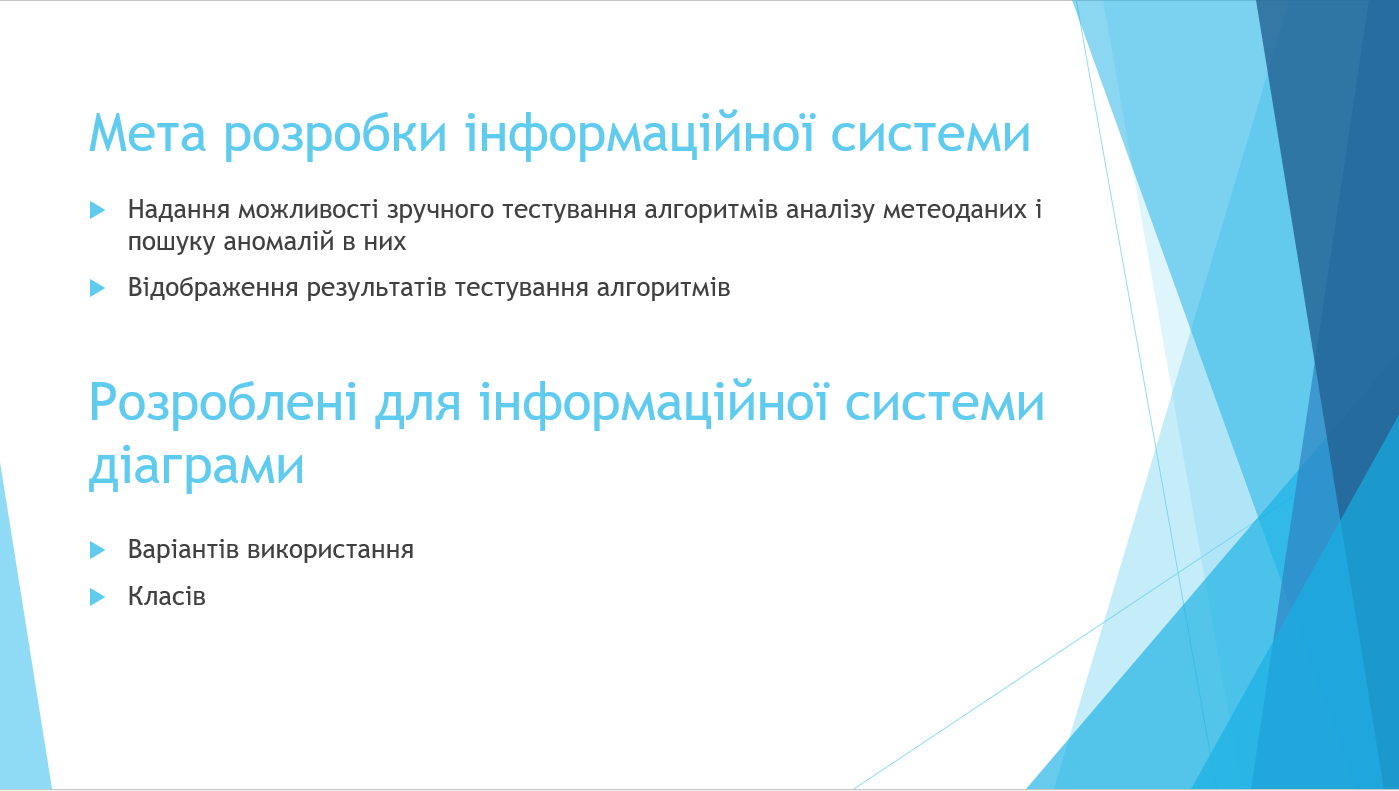


Рисунок Б5 – Слайд описом мети та описом розроблених діаграм

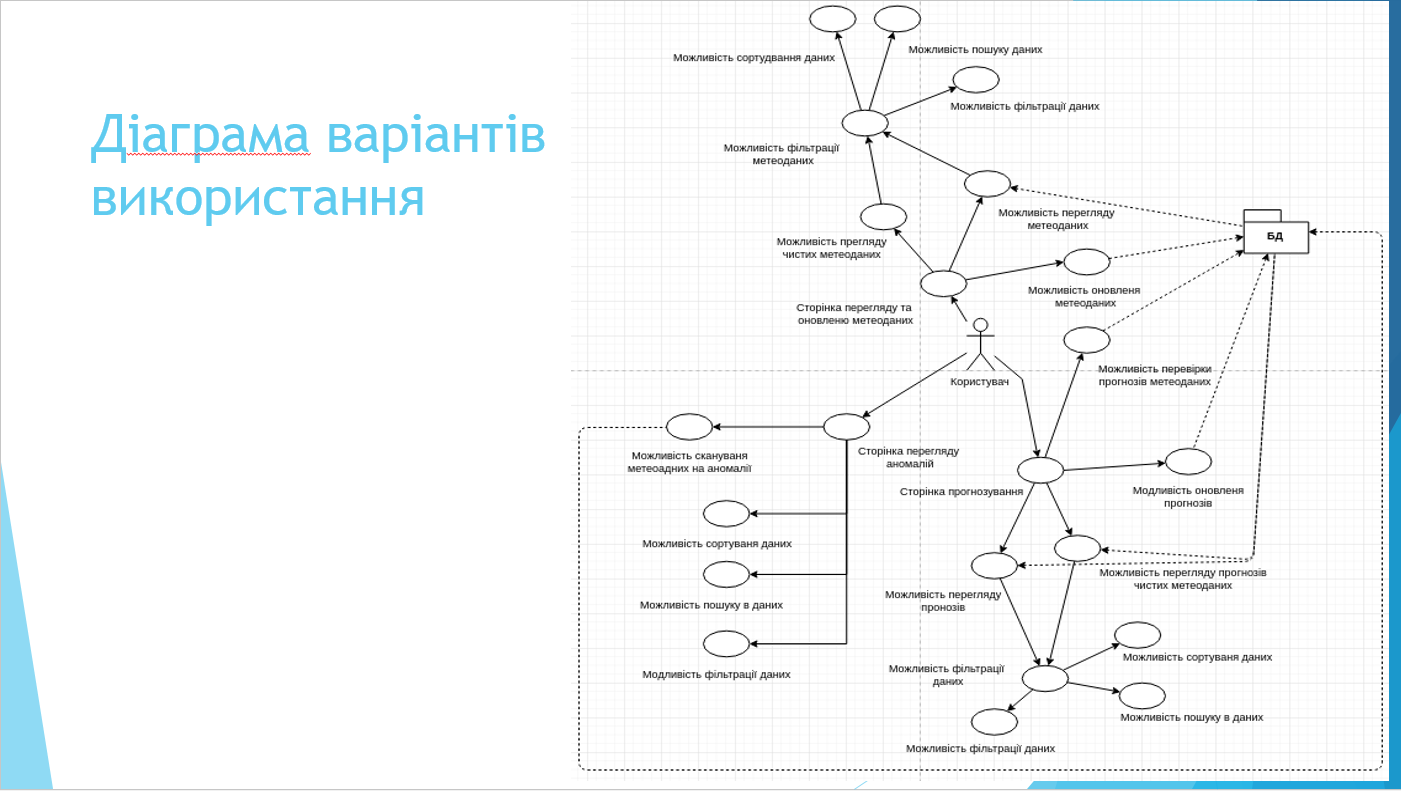


Рисунок Б6 – Слайд з діаграмою варіантів використання

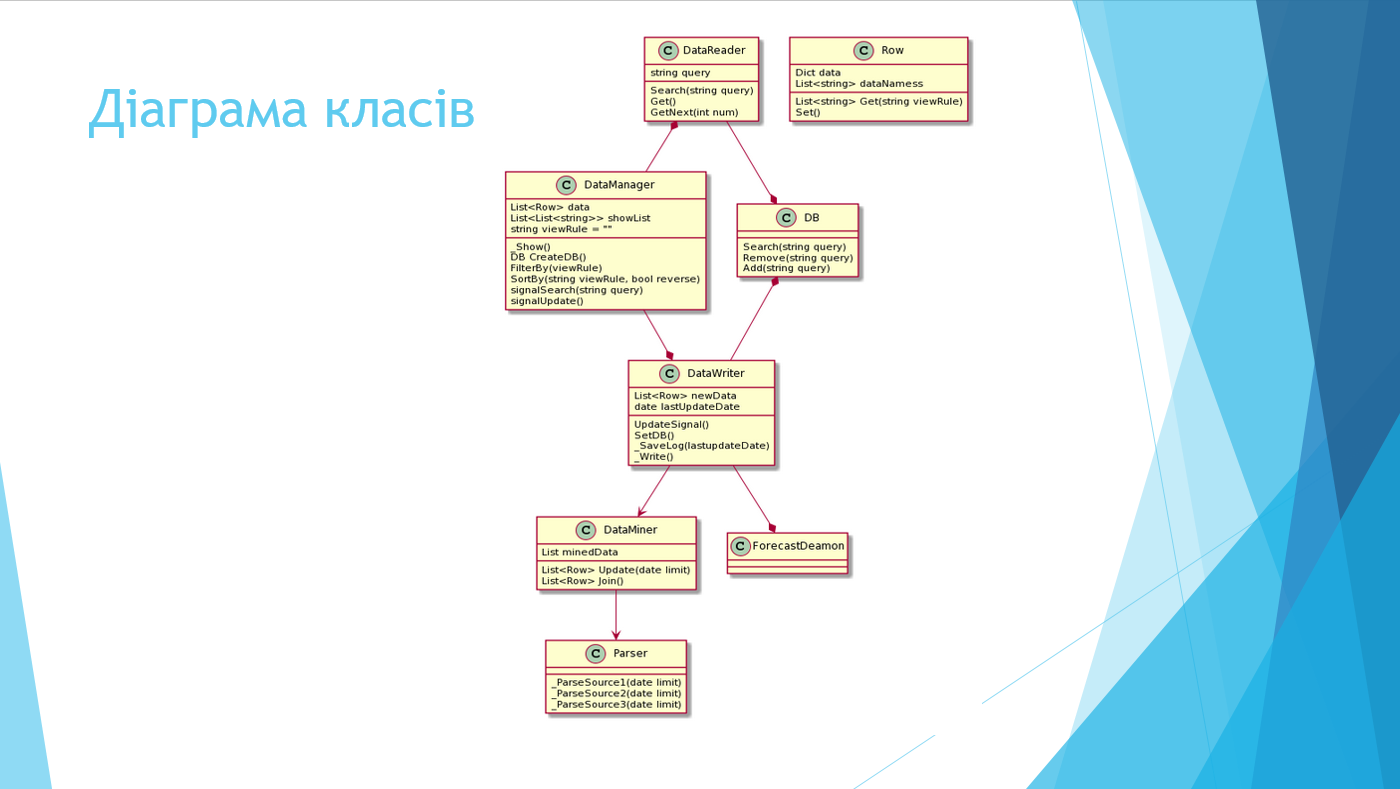


Рисунок Б7 – Слайд з діаграмою класів

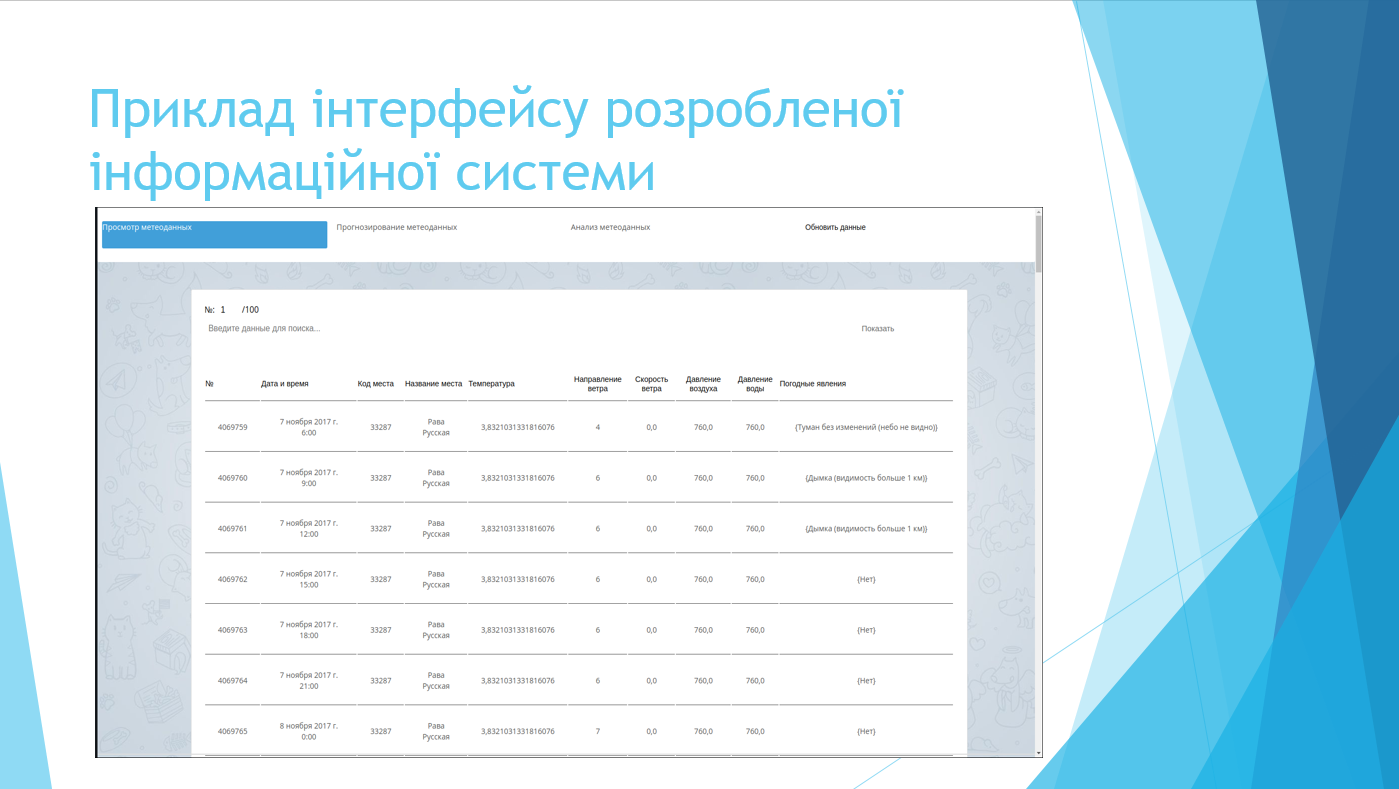


Рисунок Б8 – Слайд з прикладом інтерфейсу дороти

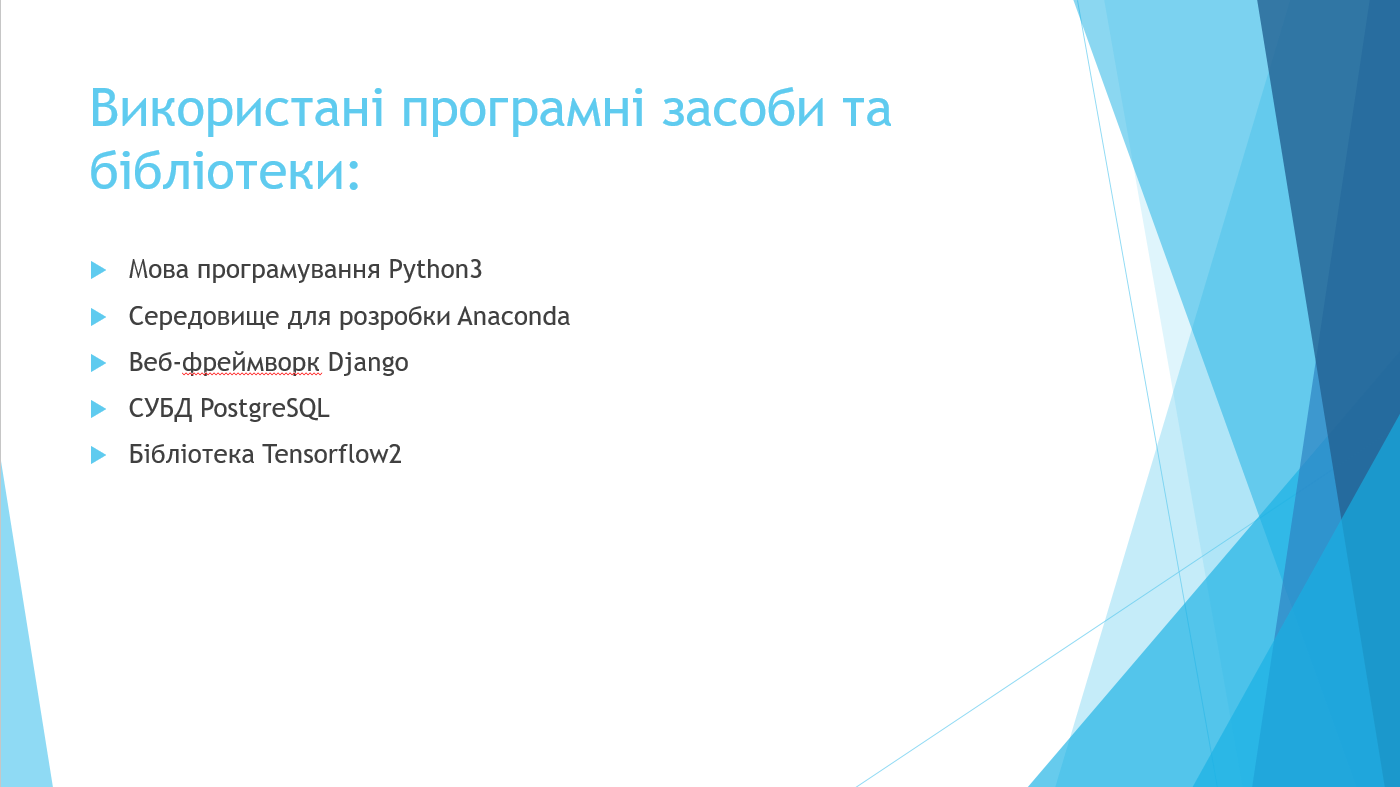


Рисунок Б9 – Слайд з визначенням засобів реалізації роботи

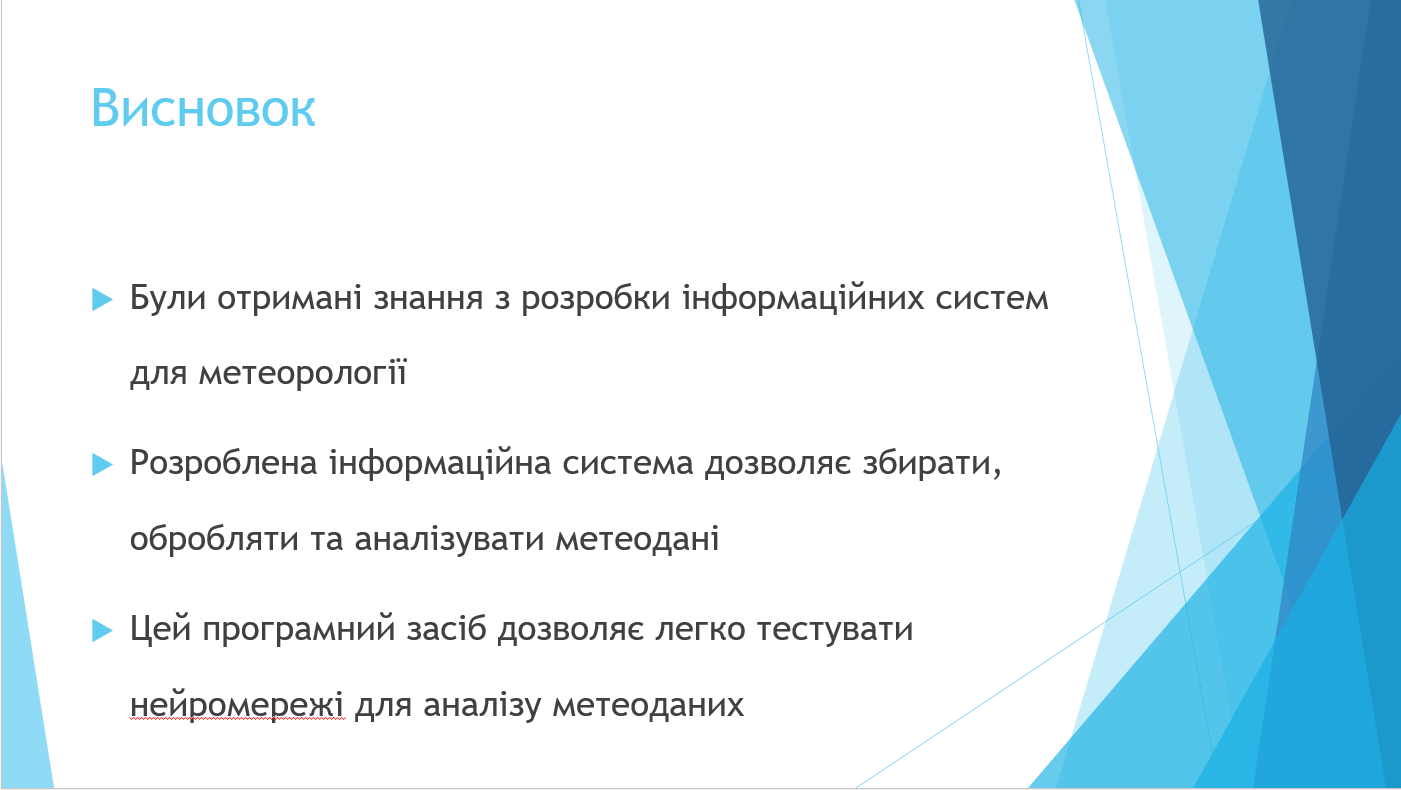


Рисунок Б10 – Слайд з загальними висновками



Рисунок Б11 – Слайд з висновками до роботи

ДОДАТОК В

Електронні матеріали

|  |  |  |
| --- | --- | --- |
| Зміст | Папка | Ім'я файла |
| Пояснювальна записка до роботи бакалавра |  | diploma.docх |
| Вихідні дані до роботи бакалавра |  | readme.txt |
| Докладний проект програми з кодами і поясненнями у середовищі проектування програмного продукту |  | project.zip |
| Демонстраційний ролик програмної системи (слайди, анімація тощо) |  | demo.pptх |