**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА**

ІНСТИТУТ ФІЗИКО-МАТЕМАТИЧНИХ ТА КОМП’ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП’ЮТЕРНИХ НАУК

**Веб-орієнтована система для інтелектуального аналізу валютних коливань**

**Дипломна робота**

**Рівень вищої освіти - другий (магістерський)**

Виконав:

студент VI курсу, групи 644м

спеціальності

8.122 Комп’ютерні науки та інформаційні*\_\_* технології\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(шифр і назва спеціальності)

Бурла Є.К.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_\_д.ф.-м.н. проф Ушенко Ю.О.,\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_к.ф.-м.н. Довгунь Ю.Я.\_\_\_\_

(прізвище та ініціали)

Рецензент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали, кафедра, університет)

**До захисту допущено:**

**Протокол засідання кафедри № \_\_\_**

від „\_\_\_” \_\_\_\_\_\_\_\_\_ 2017 р.

зав. кафедри\_\_\_\_\_\_\_\_\_ д.ф.-м.н., проф. Ушенко Ю.О.

Чернівці – 2017

**АНОТАЦІЯ**

Пояснювальна записка містить 77 сторінок формату А4, кількість додатків – 1.

При розробці дипломної роботи використано список джерел, що налічує 16 назв.

У першому розділі магістерської описано теоретичні основи фінансового аналізу даних, клієнт серверну архітектуру та проведено огляд використаних технологій.

У другому розділі проведено огляд аналогів продукту, опис предметної області, а також огляд практичного застосування методів отримання даних з мережі Інтернет. Також в цьому розділі описана методика підключення API та інструкція користувача.

Третій розділ присвячений охороні праці в галузі.

**ABSTRACT**

The explanatory note contains 77 pages of A4 format, the number of applications is 1.

During the development of the diploma project is used the list of sources, which has 16 titles.

The first section of the Magister work describes the theoretical foundations of financial data analysis, client server architecture and review of used technologies.

The second section reviews the product's analogues, a description of the subject area, as well as an overview of the practical application of methods for obtaining data from the Internet. Also, this section describes the API connection method and user's manual.

The third section is devoted to labor protection in the industry.

**ЗМІСТ**

[ВСТУП 5](#_Toc501487761)

[І. ТЕОРЕТИЧНА ЧАСТИНА 7](#_Toc501487762)

[1.1 Теоретичні основи фінансового аналізу 7](#_Toc501487763)

[1.2 Клієнт-серверна архітектура 11](#_Toc501487764)

[1.3. Огляд використаних технологій 15](#_Toc501487765)

[ІІ. ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА 29](#_Toc501487766)

[2.1 Огляд аналогів програмних продуктів 29](#_Toc501487767)

[2.2 Практична реалізація проекту 32](#_Toc501487768)

[2.3 Практичне застосування методів Web-mining 36](#_Toc501487769)

[2.4 Тестування 42](#_Toc501487770)

[2.5 Інструкція користувача 44](#_Toc501487771)

[ІІІ. ОХОРОНА ПРАЦІ 49](#_Toc501487772)

[ВИСНОВОК 54](#_Toc501487773)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 56](#_Toc501487774)

[ДОДАТОК А 58](#_Toc501487775)

# ВСТУП

**Актуальність теми.** На сьогоднішній день розвитку людства та глобалізації економіки багато споживачів цікавляться курсами валют та їхнім прогнозуванням, для знаходження, яких використовують глобальну мережу Інтернет, адже це найпростіший та найзручніший спосіб знайти саме дані, без потреби витрачати час на похід до банківської установи.

Тому метою даного дипломного проекту було створити проект веб-сайту, за допомогою якого користувачі зможуть переглядати курси валют в зручний для них час з будь-якої точки земної кулі станом на сьогоднішній день або на будь-яку іншу дату, за певний діапазон часу та оформити їх у вигляді таблиці, побудувати графік валютних коливань, а також за допомогою функції прогнозування, яка базується на нелінійній регресії та показниках динаміки, спрогнозувати приблизні середні курси таких валют як: EUR, USD, GBP та RUB по відношенню до гривні на 3 місяці вперед на базі 12 попередніх місяців.

**Об’єктом дослідження** є фінансова система, а саме підсистема валютних операцій.

**Предметом дослідження** є динаміка курсу основних світових валют.

**Мета проекту:** розробити, програмно реалізувати, дослідити, оптимізувати та розмістити веб-сайт для відображення, пошуку та прогнозування курсів валют.

**Новизна предмету даної розробки** полягає в тому, що на фоні розроблених на даний час аналогів, цей продукт матиме можливість працювати з багатьма валютами, відображати курси валют на діапазони дат, також відображати курси валют на декілька років, графічно відображати коливання курсів валют, спрогнозувати курси на 3 місяці вперед та можливість конвертування курсів валют із заданням потрібної кількості з високою точністю.

Для досягнення мети даного дипломного проекту необхідне вирішення наступних завдань:

* опрацювати теоретичні відомості теорії фінансів;
* забезпечити отримання даних в реальному часі;
* реалізувати конвертування валют;
* виконати аналіз валютних коливань;
* реалізувати візуалізацію досліджуваних даних за допомогою графіків;
* виконати прогнозування даних за допомогою нелінійної регресії та показників динаміки;
* забезпечити розгортання веб-сайту на Heroku Application Server.

# І. ТЕОРЕТИЧНА ЧАСТИНА

## 1.1 Теоретичні основи фінансового аналізу

Валютний курс – вираз ціни грошової одиниці однієї країни в грошових одиницях іншої. Фіксація валютного курсу здійснюється або відповідно до золотого паритету (гарантованим золотим змістом національної грошової одиниці), або за міжнародною угодою. При класичному золотому стандарті, тобто при вільному розміні валют на золото в центральному банку, валютний курс встановлювався в пропорціях до їх золотого вмісту.

Для конвертованих валют в основі курсу лежить валютний паритет. Проте курси валют майже ніколи не збігаються з їхнім валютним паритетом. В умовах міжнародної торгівлі і інших зовнішньоекономічних акцій співвідношення надходжень і платежів в іноземній валюті і, отже, попит і пропозиція іноземної валюти не перебуває у рівновазі. При активному платіжному балансі курси іноземних валют на валютному ринку даної країни падають, а курс національної грошової одиниці підвищується. Зворотне відбувається у разі, коли країна має пасивний платіжний баланс. Тому в більшості країн разом з твердим офіційним курсом національної валюти також існує вільний. За офіційного паритету здійснюються розрахунки центральних національних банків та інших валютно-фінансових установ між різними країнами і з міжнародними організаціями. Розрахунки між приватними особами і організаціями, виходять із зовнішньоторговельних і зовнішньоекономічних зв'язків та здійснюються по вільному курсу.

До середини 80-х років у МВФ та інших фінансово-кредитних інституціях було досягнуто консенсусу про те, що для нових економік найкращим виявився режим фіксованого, або принаймні “керованого”, валютного курсу насамперед через те, що такий режим дисциплінує центральні банки цих країн та утримує інфляцію на рівні менш ніж 10%. До середини 90-х цей “вашингтонський консенсус” було дискредитовано, передусім валютною кризою у Мексиці наприкінці 1994 року. Почав зароджуватися новий, до того ж абсолютно відмінний від попереднього консенсус. Полягав він у тому, що постбреттонвудському світі високомобільного капіталу лише “автоматичність” могла запобігти валютній кризі. Тут малися на увазі режими, що автоматично пристосовувалися до ринкових сил, без втручання держави: або режими повністю гнучких валютних курсів, або режими фіксованих валютних курсів, що були конституційно обмежені “валютними радами”.

Режими валютного курсу:

* що коливається — вільно змінюється під впливом попиту і пропозиції і заснований на використанні ринкового механізму.
* плаваючий — різновид валютного курсу, що коливається, зумовлюючий використання механізму валютного регулювання. Так, для обмеження різких коливань курсів національних валют, що викликають несприятливі наслідки валютно-фінансових та економічних відносин, країни, що увійшли в Європейську валютну систему, ввели в практику узгодження відносних меж взаємних коливань валютного курсу.
* фіксований — офіційно встановлене співвідношення між національними валютами, засноване на визначеннях в законодавчому порядку валютних паритетах. Він допускає закріплення змісту національних грошових одиниць безпосередньо в золоті або доларах США при строгому обмеженні коливань ринкових курсів валют в межах одного відсотка.

Міжнародна торгівля ведеться в грошових одиницях. У кожній країні існує своя валюта. І перш, ніж купити товари, послуги або фінансові активи за кордоном, треба спочатку звернутись на зовнішній валютний ринок, де валюту можна купити чи обміняти на іншу. Основний обмін валют для міжнародної торгівлі проводиться в найбільших фінансових центрах світу.

Курси, по яких обмінюються валюти, визначаються їх попитом і пропозицією. Наприклад, на початку 1986 року американський долар коштував 10 французьких франків, два роки опісля він коштував вже 6 франків. По мірі зміни обмінних курсів валют змінюються ціни на товари цих країн — імпортерів та експортерів.

Обмінні курси впливають на зовнішньоторговельний баланс країни і впливають на хід міжнародних фінансових операцій. Але незалежно від того, чи є обмінні курси валют високими або низькими, чи перевищує вартість експорту вартість імпорту, або навпаки, експортуються ті товари, які країна може виготовити і які мають порівняно низьку альтернативну вартість, та імпортуються ті, що виготовляються за кордоном за аналогічних умов.

Основним поняттям, створеним для пояснення валютних курсів є паритет купівельної спроможності, ПКС (purchasing power parity - РРР), для формулювання якого зазвичай привертають так званий закон однієї ціни: ціна товару в одній країні повинна бути рівна ціні товару в іншій країні; а оскільки ці ціни виражаються в різних валютах, це співвідношення цін і визначає курс обміну однієї валюти на іншу.

Нехай Pd - внутрішня ціна (domestic price) даного товару, а Pf - його ціна за кордоном, в сусідній країні (foreign price). Ці ціни являють собою кількість валют, національної для даної країни та іноземної, які дають за одиницю товару всередині країни та закордоном. Відношення цін і буде тим курсом, за яким одну валюту стануть обмінювати на іншу заради купівлі даного товару.

З 04 квітня 2014 року офіційні курси гривні до іноземних валют та курсу банківських металів визначаються Національним банком України на поточний робочий день, і діють з часу їх встановлення.

Різниця між курсом покупки і курсом продажу, називається курсовим спредом і є джерелом прибутку для суб’єкта, що здійснює валютне котирування. Валютне котирування має такі види:

* одиницям національної валюти відповідає 10, 100 тощо, одиниць іноземною валюти, тобто базою котирування є іноземна валюта: USD/UAH = 5,4350-5,4850 база котирування – USD, валюта котирування – UAH;
* пряме котирування (американський варіант) – визначає, скільком обернене котирування (європейський варіант) – визначає, скільком одиницям іноземної валюти відповідає 10, 100 і т.д. одиниць національної валюти, тобто базою котирування є національна валюта: UAH/USD = 0,1823-0,1840, база котирування – UAH, валюта котирування – USD;

Очевидно, що між рівнями курсів, обчислених за прямим (К) і оберненим котируванням (R), виконується співвідношення:



У більшості країн застосовують пряме котирування.

* крос-котирування — визначає курс двох валют одна до одної через курс кожної з них відносно третьої валюти, часто до долара США.

Функції валюти:

1. Інструмент обслуговування міжнародних економічних відносин, міжнародний платіжно-купівельний засіб.
2. Інтернаціональна міра вартості, цінності товарів.
3. Міра вартості, цінності національних грошових одиниць.

Конвертованість валюти - це її здатність до вільного обміну на інші валюти за визначеним курсом. Види конвертованості валюти:

* зовнішня конвертованість - можливість використання національної валюти у міжнародних розрахунках;
* внутрішня конвертованість - можливість купівлі-продажу іноземної валюти в обмін на національну (і навпаки) всередині країни;
* конвертованість за капітальними операціями - відсутність обмежень на платежі та трансферти за міжнародними операціями, пов'язаними з рухом капіталу;
* конвертованість за поточними операціями - відсутність обмежень на платежі та трансферти за поточними операціями.

Фактори, що впливають на конвертованість валюти:

* економічний потенціал країни;
* стабільність її внутрішнього грошового обігу;
* ступінь розвитку національних грошових ринків і ринків капіталу;
* ступінь відкритості національної економіки та її інтегрованості у світові ринки;
* характер валютних обмежень.

Валютні обмеження - це встановлена у законодавчому, адміністративному порядку система правил і норм з регламентації операцій з іноземною валютою, платіжними документами в іноземній валюті, які здійснюються юридичними і фізичними особами.

Форми валютних обмежень:

* наявність декількох обмінних курсів залежно від типу економічних операцій;
* використання двосторонніх платіжних угод (клірингів). У цьому випадку платежі, отримані від іноземного партнера, не можуть бути направлені на придбання товарів в іншій країні;
* вимоги національної влади щодо продажу центральному банку або репатріації вітчизняними фірмами-експортерами своїх валютних надходжень;
* ліцензування експорту та імпорту, а також специфічні правила і норми в галузі руху капіталів і продажу (наприклад, регламентування іноземних інвестицій, вивозу прибутків, отримання зовнішніх кредитів, експорту, імпорту тощо). Повністю конвертовані валюти мають: країни з розвинутою та відкритою економікою; країни, які є великими експортерами нафти; невеликі країни, що виконують функції офшорних зон фінансової інфраструктури та є зручними "гаванями" для філій транснаціональних банків.

## 1.2 Клієнт-серверна архітектура

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

* набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
* набір клієнтів, які використовують сервіси, що надаються серверами;
* мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів (рис.1).

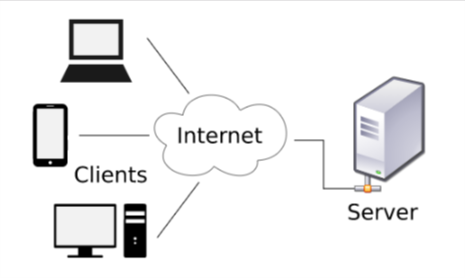


Рис. 1. Модель клієнт-сервер.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери — це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми — і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Обов'язки та взаємодія.Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

* рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
* прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
* рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

* модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
* модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду веб-сторінок називаються браузерами (веб-оглядачами). Найпоширеніші браузери: Google Chrome, Internet Explorer, Firefox, Safari і Opera.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме - автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

Для запиту до веб-сервера клієнтська програма повинна задати місцезнаходження комп'ютера, на якому розміщується серверна програма, назву потрібного документа і, можливо, інші дані, які специфікують запит. Мережа забезпечує знаходження сервера і передачу йому клієнтського запиту. Серверні програми обробляють цей запит, відповідь пересилається по мережі клієнтові.

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка застосунку. Програми проміжного рівня можуть функціювати під управлінням спеціальних серверів застосунків, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення — звичайний браузер. Це позбавляє його необхідності завантажувати та інстaлювати спеціальні програми (хоча інколи така необхідність все-таки виникає). Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

Приклад трирівневої архітектури. На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними — СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-засоби можуть бути найрізноманітнішими. Так, для створення серверних застосунків, крім PHP, широко застосовуються Java, Perl, Python. Взагалі, технології створення розподілених, зокрема веб-застосунків, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а також про .NET — порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки XML (Extensible Markup Language).

## 1.3. Огляд використаних технологій

Для автоматизації зборки проекту використовувався Maven. Maven – це засіб автоматизації роботи з програмними проектами, який використовується для управління (management) та складання (build) програм.

Вся структура проекту описується в файлі pom.xml (POM – Project Object Model), який повинен знаходитись в кореневій теці проекту. Ключовим поняттям Maven являється артефакт – це, по суті, будь-яка бібліотека, збережена в репозиторії. Це може бути будь-яка залежність (dependency) або плагін.

Dependencies – це ті бібліотеки, які безпосередньо використовуються в проекті для компіляції коду або його тестування.

Плагіни – використовуються Maven при складанні проекту чи для інших цілей (деплоймент, створення файлів проекту для Eclipse тощо).

Для опису програмного проекту, який потрібно побудувати, Maven використовує конструкцію відому як Project Object Model (POM), залежності від зовнішніх модулів, компонентів та порядку побудови. Виконання певних, чітко визначених задач – таких, як компіляція коду та пакетування відбувається шляхом досягнення заздалегідь визначених цілей. Ключовою особливістю Maven є його мережева готовність.

Двигун ядра може динамічно завантажувати плагіни з репозиторію, того самого репозиторію, що забезпечує доступ до багатьох версій різних Java-проектів з відкритим кодом, від Apache та інших організацій та окремих розробників. Цей репозиторій та його реорганізований наступник – Maven-2 репозиторій, намагається бути де-факто механізмом для дистрибуції Java програм, але прийняття його в такій якості йде повільно.

Maven забезпечує підтримку побудови не просто перебираючи файли з цього репозиторію, але й завантажуючи назад артефакти у кінці побудови. Локальний кеш звантажених артефактів діє, як первісний засіб синхронізації виходу проектів на локальній системі.

Maven базується на плагін-архітектурі, що дозволяє зробити використання будь-якої програми контрольованим через стандартний вхід. Теоретично, це могло б дозволити будь-кому писати плагіни для інтерфейсу з інструментами для побудови (компілятори, тестери тощо) для будь-якої мови. В дійсності, підтримка і використання для мов відмінних від Java були мінімальною.

Життєвий цикл мавена доволі очікуваний:

* перевірка – перевіряє коректність метаінформації проекту;
* компіляція – компілює сирцевий код;
* тест – проганяє тести класів з попереднього кроку;
* упаковка – запаковує скомпільовані класи в зручно переміщуваний формат (jar або war, наприклад);
* інтеграційний тест – відправляє запаковані класи в середу інтеграційного тестування і проганяє тести;
* перевірка – провіряє коректність пакету і задоволені вимоги якості;
* встановлення – заганяє пакет в локальний репозиторій, звідки він (пакет) буде доступний для використання як залежність в інших пакетах;
* розгортання – відправляє пакет на віддалений сервер, звідки інші розробники можуть його отримати й використати.

Розглянемо під’єднані за допомогою Maven бібліотеки, які були використанні в проекті:

Java EE API – включає в себе декілька технологій, які розширяють деякі функціональні можливості Java SE API-інтерфейсів. Наприклад, включає в себе: «javax.servlet.\*», «javax.websocket.\*», «javax.faces.\*», «javax.faces.component.\*», «javax.enterprise.inject.\*», «javax.validation.\*», «javax.persistence.\*» та інші[5].

Java Mail API – надає платформо-незалежні та протокол-незалежні фреймворки для mail та messaging додатків.

JSON – (англ. JavaScript Object Notation, укр. об’єктний запис JavaScript, вимовляється джейсон) – це текстовий формат обміну даними між комп’ютерами. JSON базується на тексті, і може бути з легкістю прочитаним людиною. Формат дозволяє описувати об’єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). Розробив і популяризував формат Дуглас Крокфорд (рис.2).

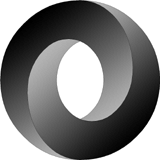


Рис.2. Логотип JSON

JSON знайшов своє головне призначення у написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAJ, виступає як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером[11]. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об’єкти.

За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш придатним для серіалізації складних структур.

Якщо говорити про веб-застосунки, в такому ключі він доречний в задачах обміну даними як між браузером і сервером (AJAX), так і між самими серверами (програмні HTTP-інтерфейси). Формат JSON так само добре підходить для зберігання складних динамічних структур в реляційних базах даних або файловому кеші.

Приклад використання JSON:

var ajaxData = '{"name": "wiki", "fname": "pedia", "rates": [1, 4, 5, 6]}';

var ajaxObj = JSON.parse(ajaxData);

console.log(ajaxObj.name + ajaxObj.rates[2]);

JSON будується на двох структурах:

* набір пар ім'я/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативним масивом;
* впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список, або послідовність.

Це універсальні структури даних. Теоретично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Оскільки JSON використовується для обміну даними між різними мовами програмування, то є сенс будувати його на цих структурах.

У JSON використовуються такі їхні форми:

* об'єкт – це невпорядкована множина пар ім’я/значення. Об'єкт починається з символу { і закінчується символом }. Кожне значення слідує за : і пари ім’я/значення відділяються комами;
* масив – це впорядкована множина значень. Масив починається символом [ і закінчується символом ]. Значення відділяються комами;
* значення може бути рядком в подвійних лапках, або числом, або логічними true чи false, або null, або об’єктом, або масивом. Ці структури можуть бути вкладені одна в одну;
* рядок – це впорядкована множина з нуля або більше символів юнікода, обмежена подвійними лапками, з використанням escape-послідовностей, що починаються із зворотної косої риски. Символи представляються простим рядком.

Тип Рядок (String) дуже схожий на String в мовах C і Java. Число теж дуже схоже на C- або Java-число, за винятком того, що вісімкові та шістнадцяткові формати не використовуються. Пропуски можуть бути вставлені між будь-якими двома лексемами.

Наступний приклад показує JSON представлення об'єкта, що описує людину. У об’єкті є рядкові поля імені і прізвища, об'єкт, що описує адресу, і масив, що містить список телефонів.

{ "firstName": "Іван",

"lastName": "Коваленко",

"address": {

"streetAddress": "вул. Грушевського 14, кв.101",

"city": "Київ",

"postalCode": 21000

},

"phoneNumbers": [

"044 123-1234",

"050 123-4567"

]

}

Наступний фрагмент коду JavaScript, що описанний в проекті показує, як клієнт може використати XMLHttpRequest для запиту об’єктів з серверу:

$.ajax({

   url : '/rest/currencies/' + path,

    method : method,

    data : dataSendObj,

    dataType : 'json'

});

де url – шлях на який буде йти запит, method – метод з яким буде відправлятися запит (GET, POST, PUT, DELETE та інші), data – інформація яку відправляє кліент на сервер, dataType – тип даних який клієнт очікує від сервера.

REST(Representational State Transfer) **–** це архітектурний підхід в дизайні розподілених додатків, наприклад, веб-сервісів. Основними цілями використання цього підходу являються: розширюваність (в плані введення нових компонентів), уніфікований інтерфейс доступу, незалежне розгортання компонентів (рис.3).



Рис.3. Логотип REST API

REST-архітектура має декілька обмежень. По-перше, дана архітектура описує взаємодію клієнта і сервера, відповідно може використовуватись не тільки в випадках, в яких є клієнт і сервер. По-друге, вона не має на увазі залежності від користувацького контексту. По-третє, уніфікований інтерфейс, не дозволяє надсилати довільні запити. REST-архітектура побудована на основі протоколу HTTP, використовуючи для відправки запитів цього протоколу: GET, POST, PUT, DELETE, для визначення дії, яку необхідно виконати з вхідними параметрами.

Основним поняттям, на якому будується REST, являється поняття ресурсу (деякого джерела даних), який задається унікальним ідентифікатором (наприклад, в HTTP це URI). При маніпуляції з ресурсами використовується деякий стандартизований інтерфейс доступу.

Ресурси можуть містити підресурси (дочірні ресурси), які містять інформацію про деяку частину ресурсу[12]. Одним з найбільш розповсюджених застосувань REST являються RESTfull веб-сервіси.

RESTfull Web Services (веб API) – це веб-сервіс, побудований на основі HTTP і принципів REST. Це набір ресурсів, які характеризуються наступними ознаками:

* базовий URI веб-сервісу, наприклад: http://yoursite.yourdomain/ resources/».
* підтриманими типами даних.
* набором операцій підтриманими сервісом, відповідним HTTP-методам (GET, POST, PUT, DELETE та інші).
* API повинен бути основаним на гіпертексті (тобто HTML).

Таким чином працюють різні HTTP-методи для ресурсів і під ресурсів.

Батьківський ресурс: «http://yoursite.yourdomain/resources/resource1/»:

* GET: вертає список дочірніх ресурсів, і можливо деякі інші їх особливості;
* POST: створює новий дочірній елемент колекції resource1;
* PUT: повністю заміщає ресурс resource1;
* DELETE: видаляє ресурс resource1.

Дочірній ресурс: «http://yoursite.yourdomain/resources/resource1/ element1»:

* GET: вертає детальний опис дочірнього ресурсу зі всіма його особливостями;
* POST: часто не використовується для листових вузлів. Працює з підресурсом element1, як з колекцією, добавляє до нього дочірній ресурс;
* PUT: заміщає вказаний дочірній ресурс, а в випадку, якщо його не існує - створює;
* DELETE: видаляє ресурс element1.

Для RESTfull веб-сервісу, основними даними являються URI ресурсу на сервері, і який HTTP метод буде використовуватися. JAX-RS визначає анотації за допомогою котрих можна отримувати цю інформацію. Кожен клас, який являється ресурсом, потрібен мати, хоча б, одну з цих анотацій.

Анотація @Path вказує URI ресурсу. Данна анотація була об’явлена в javax.ws.rs.Path, і може використовуватись для того, щоб змінювати класи і методи. Вона приймає один параметр: строку, яка являється шаблоном URI ресурсу. Шаблон URI вказує місцезнаходження ресурсу. Як показано в прикладі, такий шаблон може містити наступні компоненти: необроблені складові шляху і ідентифікатори параметрів, заключенні в фігурні скобки.

JAX-RS **–** Java API для RESTfull Web Services. Це Java запрограмована мова API, яка підтримує створення веб сервісів (web services) згідно з REST (Representational State Transfer) архітектурним патерном. JAX-RS використовує анотації, впроваджені ще в Java SE 5, для спростування веб сервісів для клієнтів (рис.4).

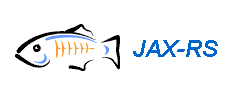


Рис.4. Логотип JAR-RS

З версії 1.1, JAX-RS є офіційно частиною Java EE 6. Примітною особливістю є те, що для початку роботи з JAX-RS не потрібно проводити конфігурацію. Для середовищ відмінних від Java EE 6 потрібно провести конфігурацію в файлі web.xml.

@Path("/resources/{resource}/{element}")

Наведенному шаблону буде, наприклад, відповідати URI /resources/database/users. Котрий буде означати, запит к таблиці users з БД. Таким чином, значения resource буде рівне database, а значення element рівне users. В залежності від того, що позначається анотацією, шаблон буде застосовуватися к повному шляху, або к шляху відносно батьківського ресурсу. Дане правило ілюструє наступний приклад:

package demo.jaxrs:

import javax.ws.rs.Path;

@Path("/colorservice/")

public class ColorService {

public ColorService(){}

@GET

@Path("/{colorCode }/")

public Color getColor(@PathParam("colorCode") String code) {

}

В JAX-RS є 5 анотацій, які відносяться до HTTP методів: javax.ws.rs.DELETE, javax.ws.rs.GET, javax.ws.rs.POST, javax.ws.rs.PUT, javax.ws.rs.HEAD. Для того, щоб вказати, що даний метод класу повинен обробляти запити відправленні деяким HTTP методом, необхідно помітити даний метод класу анотацією, наприклад @GET або @DELETE. Кореневий ресурсний клас, являється точкою входу в RESTful веб-сервісу на JAX-RS. Він позначений анотацією @Path, яка вказує URI кореневого ресурсу сервісу. Його методи або напряму реалізують операції з ресурсом, або представляють доступ до підресурсів.

Вимоги до кореневого ресурсного класу:

* даний клас повинен позначатися анотацією @Path. Вказаний шлях являє собою кореневий URI для всіх ресурсів, які обробляються даним сервісом. Якщо клас кореневого ресурсу вказує на те що його шлях і один з його методів позначений анотацією GET, то GET-запит викличе цей метод. Якщо підресурс вказує свій URI{id}, то повний шаблон шляху к підресурсу буде resource/{id} і він буде оброблювати запити типу resource/{id};
* клас повинен мати конструктор об’явлений як public, для виклику його в режимі runtime. В runtime повинно бути можливим надати всі вхідні параметри конструктора. В якості параметрів конструктора, можуть використовуватись параметри, помічені JAX-RS анотаціями, такими як @HeaderParam, @PathParam, @FormParam.

Основа Jersey – це більше, ніж реалізація посилання JAX-RS. Джерсі надає свій власний API, який розширює набір інструментів JAX-RS з додатковими функціями та утилітами для подальшого спрощення RESTful обслуговування та розвитку клієнтів. Джерсі також виставляє численні розширення SPI так, щоб розробники могли поширювати Джерсі, щоб максимально відповідати їх потребам (рис.5).



Рис.5. Логотип Jersey

Цілі проекту Jersey можна підсумувати в наступних пунктах:

* Відстежування JAX-RS API і надавання регулярних випусків якісних посилань, які поставляються разом із GlassFish;
* Забезпечити API для поширення Jersey та побудови спільноти користувачів і розробників; і, нарешті
* Зробити легким створення RESTful Web-сервісів, використовуючи Java і віртуальну машину Java.

Наступні компоненти є частиною Джерсі:

* Основний сервер: для створення REST-послуг на основі анотації (jersey-core, jersey-server, jsr311-api);
* Основний клієнт: допомагає спілкуватися з сервісами REST (джерсі-клієнт);
* Підтримка JAXB;
* Підтримка JSON;
* Інтеграційний модуль для Spring and Guice.

Підключення Jersey:

<repository>

<id>snapshot-repository.java.net</id>

<name>Java.net Snapshot Repository for Maven</name>

<url>https://maven.java.net/content/repositories/snapshots/</url>

<layout>default</layout>

</repository>

jQuery **-** популярна JavaScript-бібліотека з відкритим сирцевим кодом. Згідно з дослідженнями організації W3Techs, JQuery використовується понад половиною від мільйона найвідвідуваніших сайтів. jQuery є найпопулярнішою бібліотекою JavaScript, яка посилено використовується на сьогоднішній день (рис.6).



Рис.6. Логотип jQuery

Синтаксис jQuery розроблений, щоб зробити орієнтування у навігації зручнішим завдяки вибору елементів DOM, створенню анімації, обробки подій, і розробки AJAX-застосунків[13]. jQuery також надає можливості для розробників, для створення плагінів у верхній частині бібліотеки JavaScript. Використовуючи ці об'єкти, розробники можуть створювати абстракції для низькорівневої взаємодії та створювати анімацію для ефектів високого рівня. Це сприяє створенню потужних і динамічних веб-сторінок.

jQuery можна підключити двома способами. Перший спосіб – завантажити потрібну бібліотеку з офіційного сайту та підключити її наступним чином:

<script type="text/javascript" src="jquery.js"></script>

Другий спосіб, завантажувати бібліотеку при загрузці сторінки. Коли користувач переходить на сайт, бібліотека завантажується з сайту в браузер та зберігається там, тобто, при повторному переході на сайт, вона не буде знову завантажуватись, а буде братися з браузера. можна підключити таким чином:

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">

</script>

Основне завдання jQuery — це надавати розробнику легкий та гнучкий інструментарій кросбраузерної адресації DOM об'єктів за допомогою CSS та XPath селекторів. Також даний фреймворк надає інтерфейси для Ajax-застосунків, обробників подій і простої анімації. Принцип роботи jQuery полягає в використанні класу (функції), який при звертанні до нього повертає сам себе. Таким чином, це дозволяє будувати послідовний ланцюг методів.

$('#test') //знаходимо елемент з id="test"

.text('Клікни по мені') //встановлюємо текст елемента

.addClass('myAlert') //додаємо клас "myAlert"

.css('color','red') //встановлюємо колір тексту червоним

.attr('alert','Привіт, світе!') // додаємо атрибут "alert" із значенням

.bind( // додаємо в обробник події click функцію, яка відкриє модальне

'click',// вікно із текстом, що вказаний в атрибуті "alert" ("Привіт, світе!")

function(){alert($(this).attr('alert'))} );

AJAX **-** підхід до побудови користувацьких інтерфейсів веб-застосунків, за яких веб-сторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама звідти довантажує потрібні користувачу дані. AJAX — це не самостійна технологія, а швидше концепція використання декількох суміжних технологій[14] (рис.7).



Рис.7. Логотип AJAX

UI в проекті створювався за допомогою Bootstrap Framework. Bootstrap - це безкоштовний набір інструментів з відкритим сирцевим кодом, призначений для створення веб-сайтів та веб-застосунків, який містить шаблони CSS та HTML для типографіки, форм, кнопок, навігації та інших компонентів інтерфейсу, а також додаткові розширення JavaScript. Він спрощує розробку динамічних веб-сайтів і веб-застосунків (рис.8).



Рис.8. Логотип Bootstrap Framework

Bootstrap – це клієнтський фреймворк, тобто інтерфейс для користувача, на відмінно від серверної сторони, яка знаходиться на сервері. Репозиторій з даним фреймворком є одним з найбільш популярніших на GitHub. Серед інших, його використовують NASA та MSNBC.

Chartjs– JavaScript-бібліотекою, за допомогою якої можна створювати різні типи діаграм, при цьому використовуються можливості HTML5-елементу – canvas[16] (рис.9).

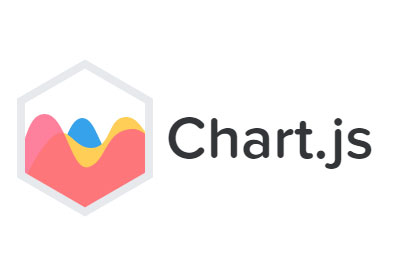


Рис.9. Логотип Chartjs

Underscore**.** В його склад входять більше 80 функцій, які спростовують роботу з колекціями, функціональними зв’язками, javascript-шаблонізаціями та іншими (рис.10).



Рис.10. Логотип Underscore

LocalStorage та SessionStorage**.** LocalStorage - «ключ / значення» зберігається в браузері, поки користувач не очистить його локальне сховище. Можна кілька днів не заходити на веб-проект і після повернення на сайт побачити раніше заповнену форму. Розмір localstorage: рекомендується надавати одному домену як мінімум 5 мегабайт. Більшість браузерів використовують саме цей ліміт. SessionStorage - «ключ / значення» зберігається в браузері поки відкрита вкладка. Тобто можна вільно переміщатися по сайту, оновлювати сторінку, але не закривати вкладку або вікно браузера.

Отже, програмний продукт «ExchangeRates» створений з використанням таких мов програмування як Java та JavaScript, а також мови розмітки HTML5 та каскадних таблиць стилів CSS3. Запити з клієнтську частину на серверну відправлялись за допомогою AJAX. RestFull веб сервіс, а саме Jersey фреймворк обробляв ці запити. В якості шаблонізатора був використаний Underscore.js. Весь інтерфейс базується на Bootsrap. Також передбачено можливість зберігати дані в LocalStorage та SessionStorage.

# ІІ. ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

## 2.1 Огляд аналогів програмних продуктів

При створенні веб-додатків, веб-сайтів, програмних продуктів тощо, дуже важливим для розробника є аналіз вже існуючих аналогів даної розробки. Це надасть можливість зрозуміти як буде виглядати UI продукту, які можливості та відмінності від аналогів він буде мати.

На самому початку розробник повинен проаналізувати всі «сильні» та «слабкі» сторони розробки, провести тестування та відокремити для себе важливі сторони.

Розробка, даного проекту полягає у написанні веб-сайту, який б дозволяв всім отримувати швидко та точно отримувати курси потрібної валюти на потрібний момент часу, побудувати графік так проаналізувати по ньому яким чином відбувалась зміна вартості потрібної валюти.

Отже, для виконання даного дипломного проекту було проведено пошук аналогів для вищезазначеної програми.

Серед ряду існуючих сьогодні аналогів даного програмного продукту було вибрано найбільш популярні, та за функціональними можливостями схожі розроблений в подальшому програмний продукт.

Аналогами є, наприклад, інформаційні модулі таких сайтів як «http://privatbank.ua/» та «http://minfin.com.ua/».

Інформаційні модулі з сайту Мінфіну (рис 11) надає інформацію про курси таких валют як: долар, євро, рубль, польський злотий, швейцарський франк та англійський фунт (рис. 11).

Також є можливість поспостерігати, як змінювалися курси валют за останній тиждень за допомогою графіка.

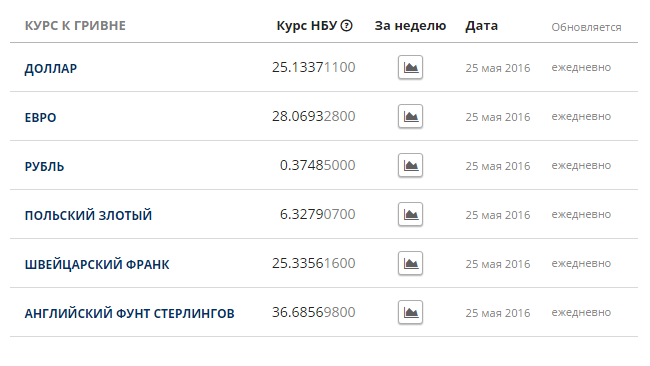


Рис.11. Інформаційний модуль курсу валют НБУ

Інформаційний модуль з Приватбанку (рис. 12) підтримує лише базові можливості, надає курси лише на сьогоднішній день у відділеннях Приватбанку та в НБУ (рис.12).

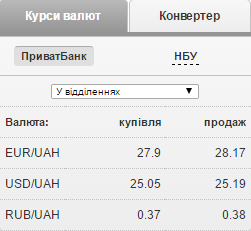


Рис. 12. Інформаційний модуль курсу валют Приватбанку

Інформаційний модуль сайту bank.gov.ua, надає курси НБУ лише на поточну дату (рис.13).



Рис.13. Інформаційний модуль курсу валют сайту bank.gov.ua

Інформаційний модуль сайту [travelex.com](http://www.travelex.com), надає лиш засіб для конвертування поточних курсів (рис.14).

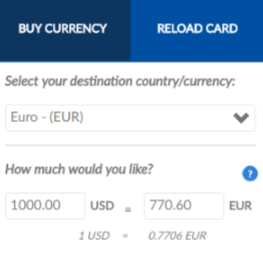


Рис.14. Інформаційний модуль курсу валют сайту travelex.com

Найбільшим недоліком існуючих розробок є відсутність вибору конкретної дати, адже можливо користувачу потрібно дізнатися, які були курси валют місяць, рік або більше часу тому для проведення аналізу, написання звітів, доповідей тощо. Другим недоліком є те, що неможливо зробити вибірку курсів на заданий діапазон дат, що може використовуватися для побудови статистичних вибірок тощо. Наступним недоліком є відсутність можливості графічно відобразити за вказаним діапазоном дат коливання курсів валют в інформаційному модулі від НБУ, а також повністю відсутність графіків в інформаційному модулі від Приватбанку. Незручним є те, що для того щоб дізнатися курси в різних банках треба переходити по їх порталам з метою відображення даних. Також нема приблизного прогнозування курсів валют. В цих інформаційних модулях також менша кількість доступних валют.

## 2.2 Практична реалізація проекту

Реалізація проекту була проведена в декілька етапів:

* ініціалізація:

– аналіз ідеї, розробка проектного завдання;

– визначення цілей, обмежень і пріоритетів проекту;

* планування:

– визначення об’єму проекту;

– виявлення всіх можливих ризиків, пов’язаних з реалізацією проекту і розробка способів їх вирішення;

* реалізація:
* отримання технічного завдання;
* створення дизайну та навігації (створення ескізу сторінки сайту, вибір місць розташування елементів, оформлення шрифтів і заголовків і багато іншого.);
* верстання;
* розташування сайту на локальному сервері (Tomcat);
* проектування RestFull веб-сервісів;
* проектування роутингу на Rest;
* програмування AJAX-запитів, які будуть відправлятися на Rest API;
* програмування сервісів для обробки запитів до Rest API, програмування надсилання запитів на API для отримання даних про курси валют, програмування сутностей (класів, які будуть описувати курс конкретної валюти та класів які будуть описувати помилки);
* програмування логіки додатку на Java EE8
* програмування шаблонізаторів для таблиць та графіків;
* програмування скриптів;
* тестування.
* налаштування конфігурації хостингу та розташування сайту на Heroku..

Користувацький інтерфейс(UI) – оснований на Bootstrap Framework. Весь інтерфейс був побудований на системі сітки. Головної її перевагою є те, що за її допомоги досягається повна адаптивність сайту. Система сіток основана на 12-х колонках, які масштабуються відповідно до розміру пристрою або до розміру оглядового вікна. В проекті знаходиться єдиний html-файл, які містить в себе всі контейнери: Home, Current Currencies, Rates By Date, Chart, Converter, Prediction, About та інші.

Весь роутинг зроблений за допомогою хешу, анімації, були створенні за допомогою чистого CSS3 та звичайно ж HTML5. Запити до REST API що знаходиться на сервері веб проекту здійснюються за допомогою AJAX, задля уникання лишніх завантажень/перезавантажень цілої сторінки. AJAX перезавантажує лиш окремий потрібний модуль.

Створене REST API приймає такі методи як GET, PUT, POST, DELETE. Воно основане на Jersey framework. Коли запит надходить до REST, він аналізує його та передає керування на відповідно анотований метод. Цей метод стягує з запиту за допомогою анотацій @FormParam потрібні параметри які були реалізовані та відправленні. Ці дані опрацьовуються, детально перевіряється їхня валідність, та передаються у відповідний сервісний клас. В сервісному класі відбуваються запити до API, а також обробка отриманої відповіді з API та її конвертація в JSON об’єкт. Дані з JSON об’єкт синтаксично аналізуються в классі JSONParsing. Тут вони відповідно до методу заносяться в списки для подальшої обробки, та ще раз проходять валідацію. Якщо при валідації виникає помилка – вона виловлюється, приводиться в User Friendly форму та вертається на клієнтську частину в JSON форматі, де вона виводиться за допомогою компонента Notify.

Прогнозування відбувається наступним чином. До API надсилаються запити за 1 календарний рік. Далі отримані дані обробляються по аналогії зі всіма, розбиваються на місяці, та обчислюється середнє значення за місяць. Далі за допомогою класів ExponentialRegression, PowerRegression та DynamicsIndicatorsService обраховуються всі формули для нелінійної регресії, а також для показників динаміки, та прогнозовані дані (на 3 місяці) відправляються на клієнт.

В програмі реалізовано 3 види прогнозування:

* прогнозування за допомогою нелінійної регресії (експоненційної регресії;
* прогнозування за допомогою нелінійної регресії (експоненційної регресії;
* прогнозування за допомогою показників динаміки.

Показники динаміки характеризують зміну значення з часом. Порівняння проводиться в формі різниці (абсолютні показники динаміки) або відносини (відносні показники динаміки). Порівняння поточного рівня з попереднім дає ланцюгові показники, порівняння з початковим (базисним) рівнем дає базисні показники.

Основна мета регресійного аналізу полягає у визначенні аналітичної форми зв'язку, в якій зміна результативного ознаки зумовлено впливом одного або декількох факторних ознак, а безліч всіх інших чинників, також впливають на результативну ознаку, приймається за постійні і середні значення.

Для вибраного алгоритму прогнозування за допомогою нелінійної регресії – рахуються дані, знаходяться значення рівняння експоненціальної та степеневих регресії, знаходяться їхні коефіцієнти детермінації. Ці коефіцієнти порівнюються, та, в залежності від того який з них більший – та регресія використовується для прогнозування даних в подальшому. Чим ближче значення коефіцієнту детермінації до 1, тим кращою є побудована регресія. Далі, за вибраною регресією прогнозуються данні, підставленням нових місяців в потрібне рівняння регресії.

Всі таблички з проекту виводяться за допомоги шаблонізатора underscore.js для уникнення перезавантаження головної сторінки лишніми даними.

Всі графіки будуються за допомогою бібліотеки Chart.js, Якщо в діапазон дат початкова дата знаходиться після кінцевої дати діапазну – на кліент вертається повідомлення про помилку, яка обробляється компонентом Notify.js. Також отриманий графік досліджується на клієнтській частині на мінімальне, максимальне та середнє значення. Ці дані відображаються під графіком в компоненті Card (рис.15).

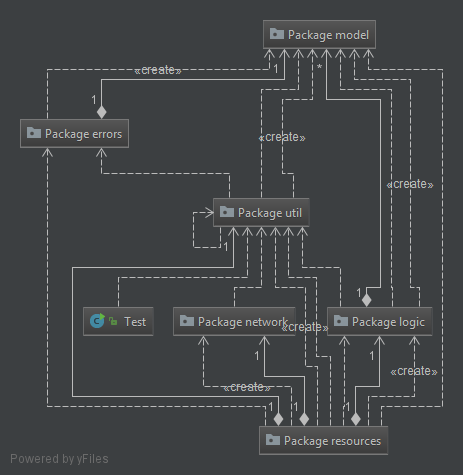


Рис.15. UML діаграма класів

## 2.3 Практичне застосування методів Web-mining

Web-mining – процес вибірки даних з веб-ресурсів, який, як правило, має більш практичну ніж теоретичну значущість. Основна ціль – збір даних (parsing) з подальшим збереженням в потрібному форматі. Фактично, задача зводиться до написання HTML парсерів.

Існує декілька підходів до вилучення даних:

1. Аналіз DOM-дерева, використання XPath.
2. Парсинг рядків.
3. Використання регулярних виразів.
4. XML парсинг.

Аналіз DOM-дерева uрунтується на аналізі DOM-дерева при використанні, якого дані можна отримати напряму по ідентифікатору, імені або інших атрибутів елемента дерева (таким елементом може служити параграф, таблиця, блок тощо). Крім того, якщо елемент не позначений ідентифікатором – до нього можна дістатися по унікальному шляху, спускаючись вниз по DOM-дереву. Наприклад:

body -> p[10] -> a[1] -> текст посилання.

Або пройти по списку однотипних елементів, наприклад:

body -> links -> 5 элемент -> текст посилання

Перевагами цього підходу є:

* можна отримати дані будь-якого типу і будь-якого рівня складності;
* знаючи, де знаходиться елемент, можна отримати його значення, прописавши шлях до нього.

Недоліки цього підходу:

* різні HTML/JavaScript Engines по-різному генерують DOM дерево, тому треба прив’язатись до конкретної системи.
* шлях елемента може змінюватися, тому, як правило, такі парсери розраховані на короткий період збору даних;
* DOM-шлях може бути складний і не завжди однозначний.

Цей підхід можна використовувати разом з бібліотекою Microsoft.mshtml, яка, по суті являється core елементом в Internet Explorer[7].

Data Extracting SDK використовує Microsoft.mshtml для аналізу DOM дерева, але є надбудовою до бібліотеки для зручності роботи:

UriHtmlProcessor proc = new UriHtmlProcessor(new Uri(<http://habrahabr.ru/new/page1/)>);

proc.initialize();

var links – from 1 in proc.Links

where 1.Class == “topic” && EndsWithInt(1.Href) == true

select new ResultItem {

Link = 1.Href,

TopicName = 1.Text.ToWindows1251()

};

Наступним етапом еволюції аналізу DOM-дерева є використання XPath – тобто, шляхів, які широко використовують при синтаксичному аналізі XML-даних. Суть даного підходу в тому, що з допомогою якогось простого синтаксису описати шлях до елементу без необхідності поступово рухатись вниз по DOM-дереву. Даний підхід використовує всім відома бібліотека jQuery та бібліотека HtmlAgilityPack:

HtmlDocument doc = new HtmlDocument();

doc.Load(“file.html”);

foreach(HtmlNode link in doc.DocumentElement.SelectNodes(“//a[@href”]) {

HtmlAttribute att = link[“href”];

att.Value = FixLink(att);

}

doc.Save(“file.html”);

Парсинг рядків*.* Не дивлячись на те, що цей підхід не можна застосовувати для написання важливих парсерів, іноді дані відображаються за допомогою деякого шаблону, тоді значення параметрів стандартні, а змінюються тільки їх значення. В такому випадку дані можуть бути отримані без аналізу DOM-дерева, а шляхом парсингу рядків, наприклад як це зроблено в Data Exctracting SDK:

Дані:

Компанія: Microsoft

Штаб-квартира: Редмунд

Код:

String data = “<p>Компания: Microsoft</p><p>Штаб-квартира: Редмунд</p>”;

String company = data.GetHtmlString(“Компания: ”, “</p>”);

String location = data.GetHtmlString(“Штаб-квартира: ”, “</p>”);

// Результат

// company = “Microsoft”

// location = “Редмунд”

Використання набору методів для аналізу рядків іноді більш ефективний аніж аналіз DOM-дерева або XPath.

Регулярні вирази і парсинг XML. Дуже зустрічається, коли HTML повністю парсили з допомогою регулярних виразів. Це невірний підхід, так як таким чином можна отримати більше проблем, ніж користі.

Регулярні вирази необхідно використовувати тільки для вилучення даних, які мають строгий формат – електронні адреси, телефони тощо, в рідких випадках – адреси, шаблонні дані.

Ще одним неефективним підходом є розглядати HTML як XML-дані. Причина в тому, що HTML не часто буває валідним, тобто, таким, що його можна розглядати як XML-дані. Бібліотеки, які реалізують таким підходом більше часу виділяють перетворенню HTML в XML, і вже потім безпосередньому парсингу даних. Тому краще уникати цей підхід[8].

Візуальний підхід. В даний момент, візуальний підхід знаходиться на початковій стадії розвитку. Суть підходу в тому, що користувач зміг без використання програмної мови чи API налаштувати систему для отримання потрібних даних будь-якої складності. Вважаю, що парсери майбутнього будуть візуальними.

Проблеми при парсингу HTML даних – використання JavaScript/AJAX/ асинхронних завантажень дуже ускладнюють написання парсерів; різні системи для рендеру HTML можуть видавати різні DOM-дерева; великі об’єми даних потребують писати розподіленні парсери, що призводить до додаткових затрат на синхронізацію.

Не можна одночасно виділити підхід, який буде 100% застосовуватись у всіх випадках, тому сучасні бібліотеки для парсинга HTML даних, як правило, комбінують, різні підходи.

Отримання даних за допомогою API. API – це, в першу чергу, інтерфейс. Інтерфейс який дозволяє розробникам використовувати готові блоки для побудови додатку. В випадку з розробкою мобільних додатків, в ролі API може виступати бібліотека для роботи з «розумним будинком» – всі нюанси реалізовані в бібліотеці.

У випадку розробки веб-додатків, API може видавати дані в іншому від стандартного HTML форматі, завдяки цьому їм буде зручно користуватися при написанні власних додатків. Сторонні загальнодоступні API частіше всього видають дані в одному з двох форматів: JSON або XML. На випадок, якщо розробник ще не вирішив зробити API для свого додатку, йому треба запам’ятати, що JSON набагато більш лаконічний і простий в прочитанні, ніж XML, а сервіси, які надають доступ до даних в XML-форматі, поступово зникають.

Проаналізуємо API веб-додатків на прикладах. Деякі додатки – наприклад, Github – мають власний API, яким можуть скористатися інші розробники. То, як вони будуть користуватися ним залежить від можливостей, які надає API і від того, наскільки добре працює фантазія в розробників. API Гітхаба дозволяє, наприклад, отримувати інформацію про користувача, його аватарі, підписників, репозиторіїв і багато іншої корисної і цікавої інформації.

Якщо взяти, до прикладу, API Твітера, то інтерфейс цього сервісу може надавати всю інформацію про твіти користувача, його читачів і тих, кого він читає тощо. Це лиш маленька частина всіх можливостей, якими кожний бажаючий може відтворити, використовуючи API стороннього сервісу або створюючи свій власний.

На основі API будуються такі проекти, як карти 2GIS, мобільні додатки, десктопні клієнти для Twitter і Vkontakte. Всі їх функції стали можливими лиш тому, що власні сервіси мають якісні та детально документовані API.

Стандартний запит від стороннього API виглядає приблизно так:

Curl https://api.github.com/users/Flush95

На випадок, якщо хтось не знає, зазначу, що curl не має жодного відношення до API і використовується в операційних системах для відправки і отримання даних через термінал.

Аналогічно можна надсилати запит на будь-якій мові.

Як видно з цього блоку, відповідь містить в собі логін, аватар, посилання на профіль на сайті і в API, статус користувача, кількість публічних репозиторіїв та іншу корисну і цікаву інформацію.

Створити повноцінний API для свого додатку – половина всієї роботи. Щоб звернутися до API, в першу чергу потрібно відправити HTTP-запити з метою отримання потрібну інформації. Значно розумніше буде створити бібліотеку для роботи з інтерфейсом, в якій будуть описані всі необхідні способи отримання і відправки інформації за допомогою API.

Таким чином, якщо розробник створює власний API, йому треба задуматись, можливо йому треба створити такі ж бібліотеки для роботи з додатком на найбільш розповсюджених мовах. І треба бути готовим, що хтось може створити власну бібліотеку для роботи з API, який створив розробник додатку.

Для підключення API МінФіну потрібно:

* надати МінФіну свою назву та адресу свого сайту для формування ключа доступу (токена), заповнюючи форму (рис.16);
* отримати токен (40-а символьний) по повідомленню на свій email;
* налаштувати систему отримання даних;
* врахувати, що кількість запитів не повинно перевищувати 1 запит раз в 5 хвилин;
* вказати посилання на Фінансовий портал «МінФін» на своєму сайті в місці, де будуть показуватись курси валют. Форма реєстрації показана нижче (рис.16).

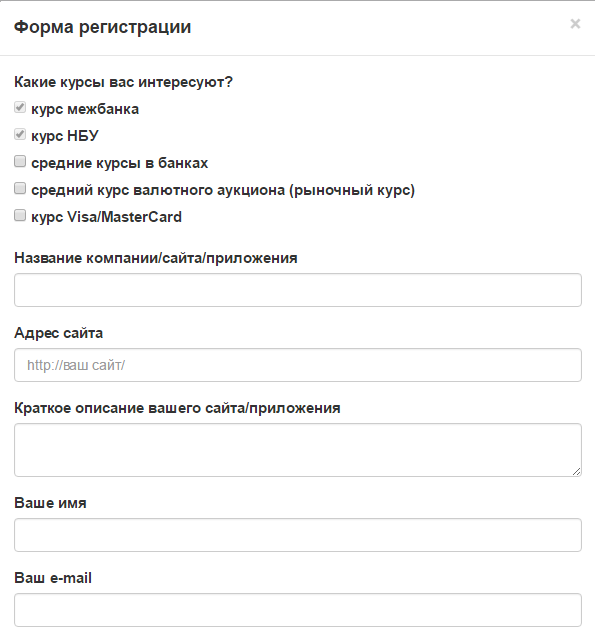


Рис.16. Форма реєстрації для отримання ключа

Результат запиту інформації с API:

{"usd":{"date":"2015-06-25 00:00:00", "ask":"21.17802000", "bid":"21.17802000", "trendAsk":"-0.34770000","trendBid":"-0.34770000", "currency":"usd"}, "eur":{"date":"2015-06-25 00:00:00", "ask":"23.74691400", "bid":"23.74691400", "trendAsk":"-0.37050300", "trendBid":"-0.37050300", "currency":"eur"} ,"rub":{"date":"2015-06-25 00:00:00", "ask":"0.39164000", "bid":"0.39164000", "trendAsk":"-0.00545000", "trendBid":"-0.00545000", "currency":"rub"}},

де:

* Ключ – валюта;
* date – дата;
* bid – продаж;
* ask – купівля;
* trendBid - тренд продажу;
* trendAsk - тренд купівлі.

Для отримання даних з minfin.com.ua необхідно надіслати параметр заголовка user-agent. Формат user-agent може бути таким:

"[ApplicationName]/[Version] ([http://link-to-your-application.com])",

де:

* ApplicationName – назва додатку;
* Version – версія;
* http://link-to-your-application.com – посилання на додаток/сайт, чи коротке описання. Наприклад, "FinApplicationBot/1.0 (<http://test.com)>";
* встановити user-agent в Java можна таким чином: System.setProperty("http.agent", "").

Рекомендується не використовувати user-agent браузерів(крім тих випадків, коли запит виконується з-під браузера, наприклад за допомогою JS).

## 2.4 Тестування

Тестування проекту здійснювалося за допомогою модульного тестування. Модульне тестування (англ. Unit testing) — це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні — інтерфейс, клас. Модульні тести, або unit-тести, розробляються в процесі розробки програмістами та, іноді, тестувальниками білої скриньки (white-box testers). Зазвичай unit-тести застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

Метою модульного тестування є ізоляція кожної частини програми та впевненість у тому, що кожна окрема частина є коректною. Модульний тест забезпечує жорсткий «контракт», за яким має працювати тестований код. Як результат, це надає деякі переваги. Модульне тестування допомагає знайти помилки раніше в циклі розробки ПЗ, що робить розробку дешевшою та швидшою.

Модульне тестування дозволяє програмісту, коли він буде змінювати код (проводити рефакторинг) бути впевненим, що модуль працює вірно (це — регресивне тестування). Оскільки модульне тестування вимагає написання тестів для всіх функцій та методів у програмі, помилки швидко локалізуються та виправляються.

Модульні тести являють собою специфічний вид документації до системи. Розробники можуть подивитися на модульний тест, щоб дізнатися про функції, що виконує модуль, та як його застосовувати.

Unit-тест перевіряє критичні характеристики модулю. Відповідність чи невідповідність цим характеристикам демонструє коректність модуля. Модульний тест «документує» ці критичні характеристики, але не треба покладатися лише на код в документуванні ПЗ під час розробки.

Слід відзначити, що звичайна письмова документація дуже повільно реагує на зміни в коді, тоді як модульні тести завжди відображають поточний стан модуля.

Код підключення Junit до Maven:

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

Тестування відбувалось на серверній частині. Для кожного модуля були написані тести, та було детально аналізовано їх результати. Тести в методах із запитами аналізували код відповіді. Якщо код відповіді 200 – тест пройдений. Якщо код відповідей з групи Bad Request 400 – тест не пройшов, помилка десь в запиті. Всі інші результати крім 200 та 400 рендеряться та вертаються на клієнт та відображаються в кастомному компоненті Notify.

Частина коду тестування методу checkSymbols:

public void testCheckSymbols() {

FormParamChecker checker = new FormParamChecker();

assert(checker.checkSymbols (“eu ”) == false);

assert(checker.checkSymbols (“euR ”) == true);

assert(checker.checkSymbols (“euro”) == false);

assert(checker.checkSymbols (“eur; usd”) == false);

assert(checker.checkSymbols (“eur,usd”) == true);

assert(checker.checkSymbols (“1”) == false);

}

Ось метод, що відповідає вимогам, які поставлено тестом:

public boolean checkSymbols(String symbols) {

if (symbols == null || symbols.isEmpty() || symbols.length() < 3)

return false;

else return true;

}

## 2.5 Інструкція користувача

При розробці інформаційної системи для аналізу валютних коливань були використані принципи компонентної розробки. Основні модулі проекту:

* Hot Currencies – зберігає вибрані вами валюти та завжди відображає курси по даним валютам в компоненті Current currencies (рис.17):

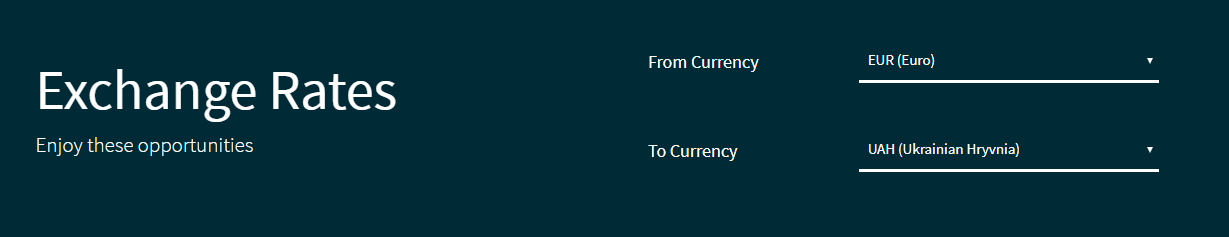


Рис.17. Компонент Hot Currencies

* Current Currencies – відображає курси на сьогоднішню дату та графік коливання курсів на тиждень для вибраних валют в компоненті Hot Currencies (рис.18):

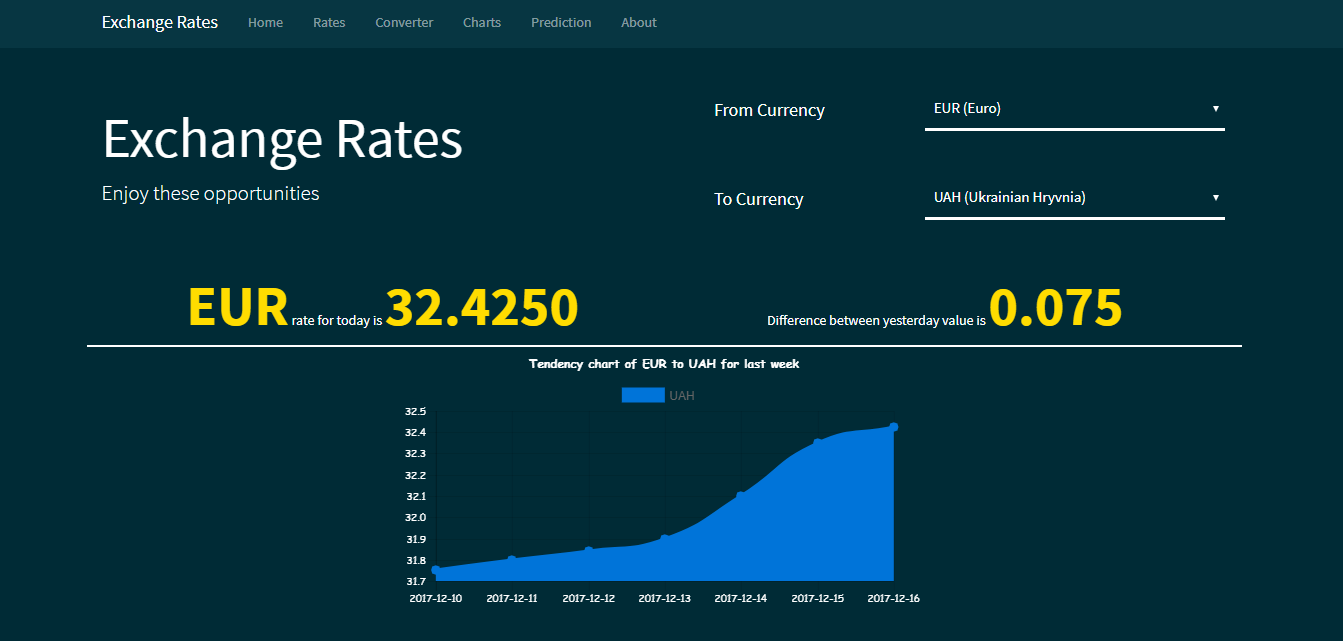


Рис.18. Компонент Current Currencies

* Rates By Date - компонент, за допомогою якого можна отримати курси вибраної валюти на потрібну дату. Якщо дату в цьому компоненті залишити пустою та відправити запит, курси будуть на поточну дату. Якщо не вибрати “Rates” – будуть відображені всі курси по відношенню до “Base Currency”(рис.19):

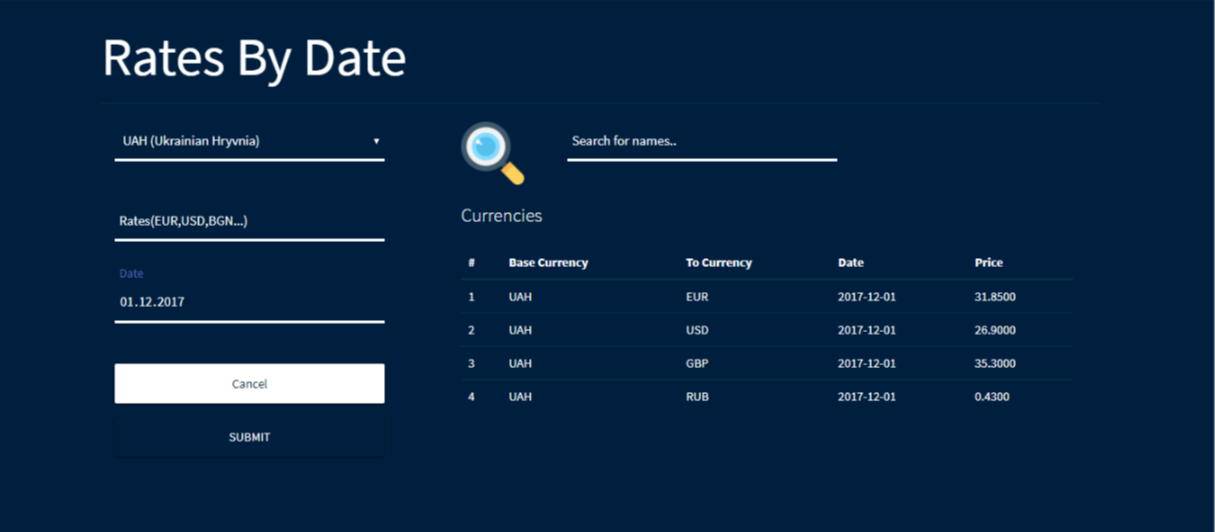


Рис.19. Компонент Rates By Date

* Search For Names - використовується для пошуку потрібних даних в отриманій таблиці (рис.20):

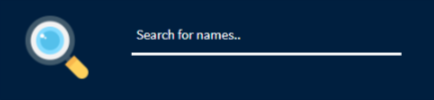


Рис. 20. Компонент Search For Names

* Converter – використовується для конвертування валют в потрібній кількості (рис.21):

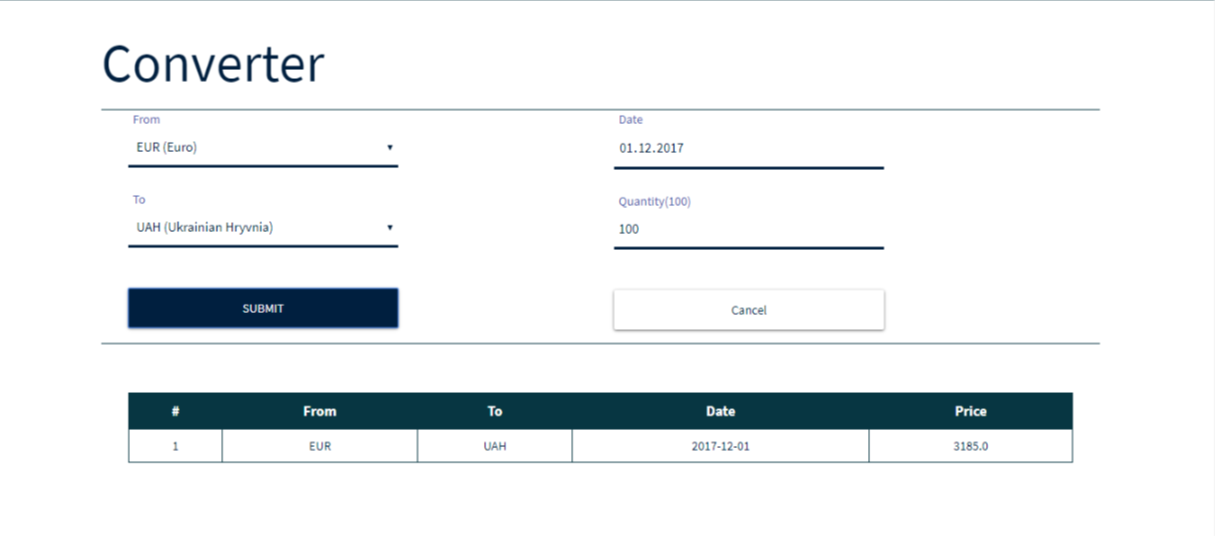


Рис. 21. Компонент Converter

* Chart – дозволяє побудувати графік на вибраний період, та визначає мінімальне, максимальне та середнє значення (рис.22):



Рис.22. Компонент Chart

* Prediction – використовується для прогнозування курсів валют. Діючі курси EUR -> UAH, USD -> UAH, GBP -> UAH та RUB -> UAH (рис.23).

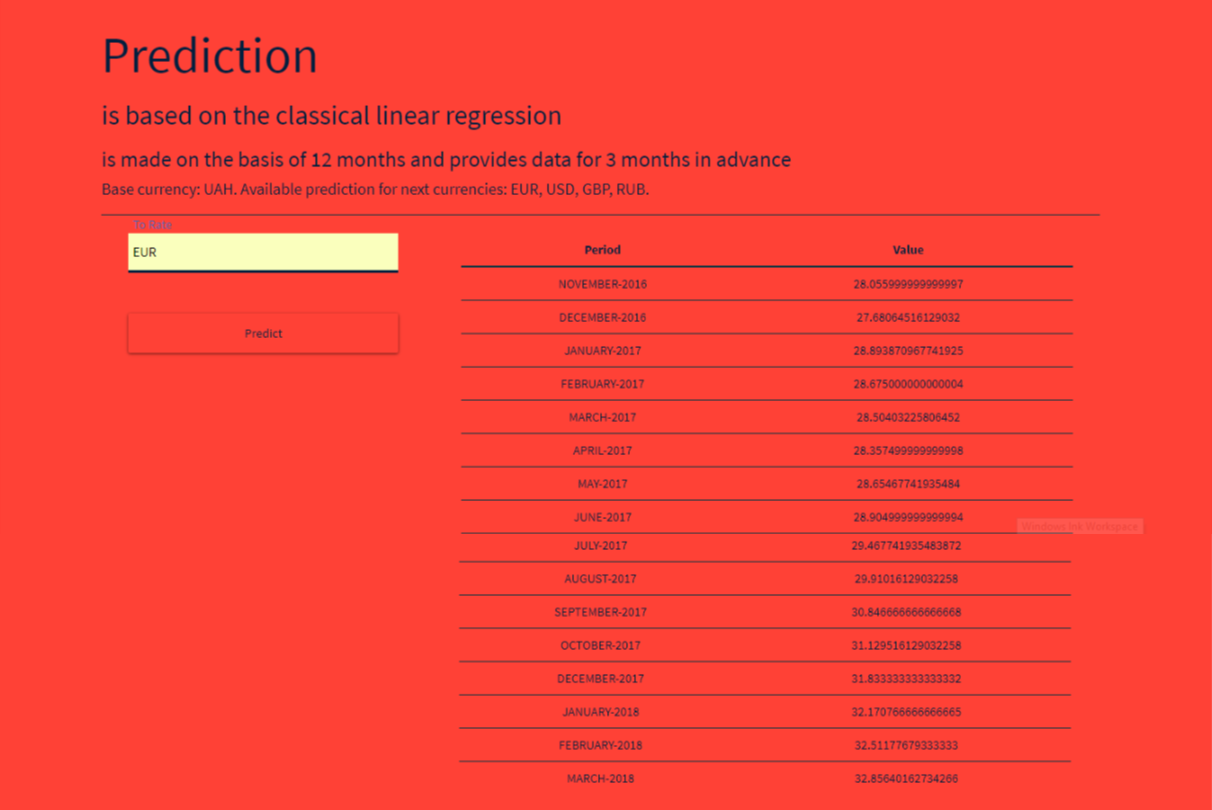


Рис.23. Компонент Prediction

* About – містить інформацію про сайт (рис.24):

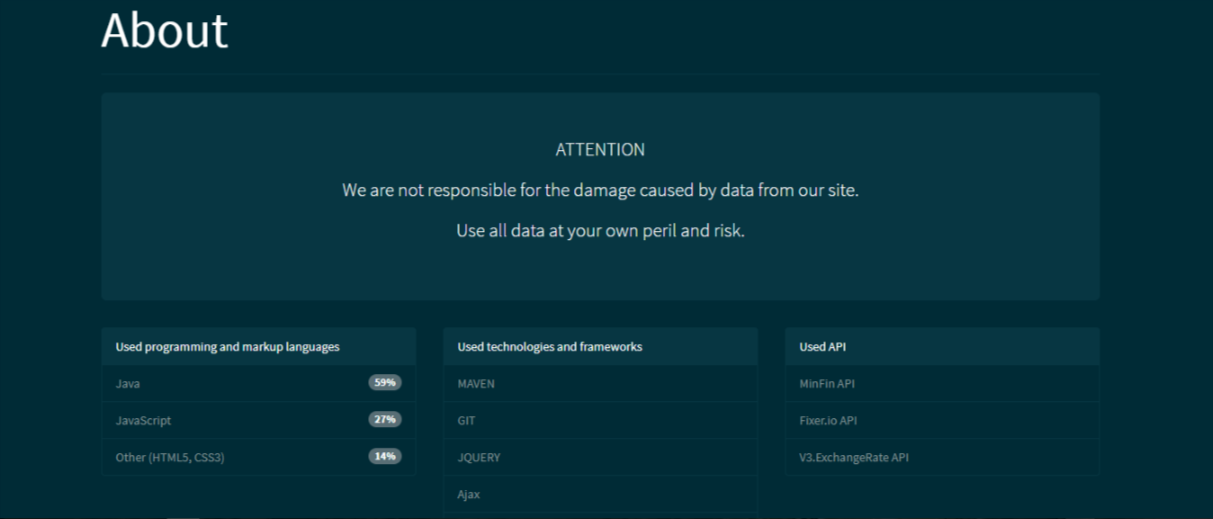


Рис.24. Компонент About

* Contact Me – містить контакти розробника (рис.25):

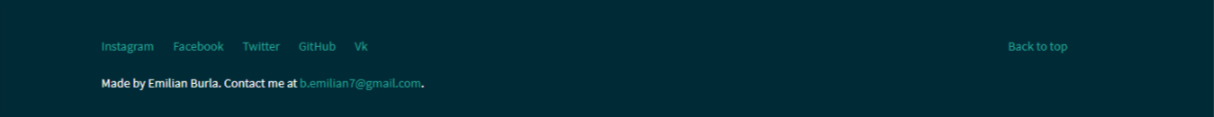


Рис.25. Компонент Contact Me

* Notify – компонент, який відображає повідомлення про стан запитів (рис.26):

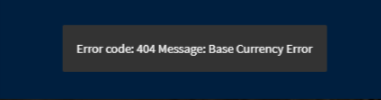


Рис. 26. Компонент Notify

Навігація по основним модулям проекту відбувається за допомогою хеш-роунтингу, що дозволяє миттєво переходити від одного модуля до іншого. У випадку виникнення будь-якої помилки, користувач буде інформуватись про це за допомогою компоненту Notify.

В результаті написання розділу 2:

* розглянуто аналоги даної інформаційної системи для аналізу валютних коливань, виявленню їх переваги та недоліки;
* описані етапи створення веб-сайту «ExchangeRates»;
* описано процедуру підключення API;
* створені діаграми класів та діаграма JavaScript-модулів;
* описані базові принципи прогнозування курсів;
* описано основні модулі програми та їх функції.

# ІІІ. ОХОРОНА ПРАЦІ

Охорона праці – це система законодавчих актів, соціально-економічних, органiзацiйних, технiчних, гiгiєнiчних i лiкувально-профiлактичних міроприємств i засобiв, забезпечуючих безпеку, охорону здоров’я i працездатнiсть людини в процесi працi.

Основоположне значення в галузі охорони праці має Закон України “Про Охорону праці”. Дія цього закону поширюється на всіх юридичних та фізичних осіб, які відповідно до законодавства використовують найману працю, та на всіх працівників.

Методологiчною основою охорони працi являється аналiз умов працi, технологiчного процесу, виробничого обладнання з точки зору можливостi виникнення небезпечних i шкiдливих виробничих факторiв.

Стратегiчним направленням розвитку охорони працi являється створення безпечної технiки i безпечних технологiй, комплексна автоматизація виробництва i на цiй основi забезпечення на всiх пiдприємствах умов, виключаючи виробничий травматизм, професiйнi захворювання i важку фiзичну працю.

Поліпшення умов та охорона праці стає одним із важливих напрямків підвищення матеріального та культурного рівня життя народу.

Зараз в Україні створена і діє ціла система законодавчих та нормативно-правових актів, що регламентують питання охорони праці в усіх галузях виробництва та сферах діяльності.

Державна політика в галузі охорони праці базується на принципах:

* пріоритет життя і здоров’я працівників по відношенню до результатів виробничої діяльності підприємства;
* повна відповідальність роботодавця за створення належних, безпечних і здорових умов праці;
* обов’язковий соціальний захист працівників, повне відшкодування шкоди особам, які потерпіли від нещасних випадків на виробництві та професійних захворювань.

Забезпечення здорових i безпечних умов працi покладається на Адмiнiстрацiю пiдприємства. Адмiнiстрацiя зобов’язана вводити сучаснi засоби технiки безпеки, попереджуючи виробничий травматизм i забезпечувати санiтарно-гiгiєнiчнi умови, попереджуючи виникнення професiйних захворювань робочих i службовцiв.

Адмiнiстрацiя пiдприємства зобов’язана забезпечити належне технiчне обладнання всiх робочих мiсць i створити на них умови працi, вiдповiднi правилам по охоронi працi. Адмiнiстрацiя розробляє інструкції, iнструктує працівників, контролює дотримання працiвниками інструкції по охоронi працi, проводить розслiдування нещасних випадкiв на виробництвi.

Трудове законодавство явилось основою для розробки типових правил внутрiшнього трудового розпорядку для робiтникiв i службовцiв підприємств, установ, органiзацiй.

Iнструкцiї по охоронi працi повиннi отримувати роздiли: загальн i вимоги безпеки; вимоги безпеки перед початком роботи; вимоги безпеки пiд час роботи; вимоги безпеки в аварiйнихситуац iях; вимоги безпеки по закiнченню роботи.

Виконання вимог iнструкцiї являється обов’язковим для працюючих. Iнструкцiї можуть бути виданi на руки, чи вивiшанi на робочих мiсцях, чи зберiгаютися у визначеному мiсцi, доступному для працюючих.

Розпочинаючи працювати на ПК, необхідно пам’ятати, що це дуже складна апаратура, яка потребує акуратного й обережного ставлення до неї, високої самодисципліни на всіх етапах її експлуатації. Напруга живлення ПК (220 В) є небезпечною для життя людини. Тому, незважаючи на те, що в конструкції комп’ютера передбачена достатня ізоляція від струмопровідних ділянок, необхідно знати та чітко виконувати ряд правил техніки безпеки.

Забороняється:

* торкатися екрана і тильного боку дисплея, проводів живлення та заземлення, з’єднувальних кабелів;
* порушувати порядок увімкнення й вимикання апаратних блоків;
* класти на апаратуру сторонні предмети;
* працювати на комп’ютері у вологому одязі та вологими руками;
* палити в приміщенні, де знаходяться комп’ютери;
* перед початком роботи на комп’ютері необхідно отримати дозвіл на роботу в уповноважених осіб педагогічно-лаборантського складу.

Під час роботи на комп’ютері необхідно:

* суворо дотримуватися інструкції з експлуатації апаратури;
* працювати на клавіатурі чистими сухими руками, не натискуючи на клавіші без потреби чи навмання;
* працюючи з дискетами, оберігати їх від ударів, дії магнітного поля й тепла, правильно вставляти дискети в дисковод;
* коректно завершувати роботу з тим чи іншим програмним засобом.

Під час роботи комп’ютера екран дисплея є джерелом електромагнітного випромінювання, яке руйнує зір, викликає втому, знижує працездатність. Через це треба, щоб очі користувача знаходилися на відстані 60-70 см від екрана, а безперервна робота за комп’ютером тривала не більше 25 хв. для дітей та 40-45 хв. для дорослих.

Будівлі та приміщення, де розміщені робочі місця операторів, мають бути не нижче II ступеня вогнестійкості згідно з ДБН В.1.1.7-2002.

Проходи до засобів пожежогасіння мають бути вільними.

Організація робочого місця оператора повинна забезпечувати відповідність усіхелементівробочого місця та їх розташування ергономічним вимогам ГОСТ 12.2.032–78 “ССБТ.

Відстань від екрана до ока працівника визначається згідно з вимогами ДСанПіН 3.3.2.007-98.

Не допускається:

* виконувати обслуговування, ремонт та налагодження ЕОМ безпосередньо на робочому місці оператора;
* зберігати біля ЕОМ папір, дискети, інші носії інформації, запасні блоки, деталі тощо, якщо вони не використовуються для поточної роботи;
* відключати захисні пристрої, самочинно проводити зміни у конструкції та складі ЕОМ або їх технічне налагодження;
* працювати з матричним принтером за відсутності вібраційного килимка та зі знятою (піднятою) верхньою кришкою.

Під пожежною безпекою об'єкта розуміють такий його стан, за якого з Регламентованою імовірністю виключається можливість виникнення і розвитку пожежі та впливу на людей небезпечних чинників пожежі.

Заходи режимного характеру передбачають заборону куріння та застосування відкритого вогню в недозволенних місцях, недопущення появи сторонніх осіб у вибухонебезпечних приміщеннях чи об'єктах, регламентацію пожежної безпеки при проведенні вогневих робіт.

Системою протипожежного захисту є сукупність організаційних заходів, а також технічних засобів, спрямованих на запобігання впливу на людей небезпечних чинників пожежі та обмеження матеріальних збитків від неї яка здійснюється за чотирма напрямками:

* Обмеження розмірів та поширення пожежі;
* Обмеження розвитку пожежі;
* Забезпечення безпечної експлуатації людей та майна;
* Створення умов для успішного гасіння пожежі.

Ефективне гасіння пожеж здійснюється чотирма основними способами:

* охолодженням горючих речовин або зони горіння;
* ізоляцією горючих речовин або окисника від зони горіння;
* розбавленням концентрації повітря чи горючих речовин;
* хімічним гальмуванням реакції горіння.

Для забезпечення цих способів гасінняпожеж використовуються наступні вогнегасильні речовини: вода, піна, інертні та негорючі гази, вогнегасні порошки. Вибір вогнегасної речовини залежить від класу пожежі.

Гасіння пожеж здійснюється пристосуваннями та засобами, які підрозділяються на стаціонарні, пересувніта первинні.Стаціонарні установки пожежогасіння являють собою апарати, трубопроводи та обладнання, які розміщені в постійних місцях і призначені для подачі вогнегасних речовин до місць займання. Такі установки поділяються на автоматичні та напівавтоматичні, які приводяться в дію відповідним сигналізатором (сповіщувачем) або спонукальним пристроєм.

Первинні засоби пожежогасіння призначені для ліквідації невеликих осередків пожеж, а також для гасіння пожеж на початковій стадії. До первинних засобів пожежогасіння належать вогнегасники, пожежний інвентар та пожежні інструменти. Залежно від вогнегасильної речовини вогнегасники поділяються на: водяні, водоімпульсні, пінні, вуглекислотні, порошкові, хладонові. Для швидкого виявлення та оповіщення про пожежу підприємства забезпечуються системами пожежної сигналізації та оповіщення. Пожежні сповіщувачі – це пристрої, що формують сигнал про пожежу. Розрізняють пожежні сповіщувачі ручної та автоматичної дії. Ручний сповіщувач вимикає людина, що виявила пожежу, шляхом натискання на пускову кнопку.

В результаті опису розділу 3 розглянуті основні правила охорони праці та безпеки при роботі з різного роду обчислювальною технікою.

# ВИСНОВОК

Результатом виконання дипломного проекту є розробка програмного продукту, призначеного для проведення інтелектуального аналізу валютних коливань.

Розроблений програмний продукт є актуальним для людей, яким треба швидко та точно дізнатися курси валют на сьогоднішній день, на конкретну дату чи на діапазон дат, також для тих, хто проводить різні дослідження росту або падіння ціни на певну валюту, хоче отримати певні статистичні дані, та для тих хто хоче приблизно спрогнозувати курси.

В результаті був створений веб-сайт, який дає змогу проаналізувати тенденцію валютних курсів. Проект розроблений за допомогою новітніх технологіях і фреймворків:

* Tomcat application server;
* Java EE;
* Maven;
* RESTfull Web сервіси.

Весь зовнішній вигляд побудований за допомогою Bootstrap Framework. Також використані: бібліотека для побудови графіків – ChartJs, шаблонізатор – Underscore.js для спрощення багатьох аспектів виводу даних та технологія AJAX, яка дає змогу відправляти дані без перезавантаженя веб-сторінки.

В результаті виконання дипломного проекту:

1. розроблено веб-сайт «ExchangeRates» (<https://exchange-rates7.herokuapp.com/>), який надає можливість проводити статистичні дослідження даних;
2. в програмному продукті передбачено можливість отримання візуального представлення опрацьованих даних за допомогою побудови таблиць та графіків.
3. розглянута розробка володіє можливістю надання даних за введеними датами;
4. створена функція прогнозування курсів валют за допомогою різних видів регресії;
5. простота в користуванні, яка забезпечується зручним мінімалістичним інтерфейсом користувача;
6. швидкодія отримання результатів є досить високою, що не можливо не відмітити, як позитивну сторону розробленого додатку.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Артус М. Гроші та кредит: Навчальний посібник/ Мирослав Артус,; Європейський ун-т. - К.: Вид-во Європейського ун-ту, 2004. - 161 с.
2. Гроші та кредит: Підручник/ За заг. ред. М.І. Савлука,; М-во освіти України. Київський нац. економ. ун-т. - К.: КНЕУ, 2001. - 602 с.
3. Гроші та кредит: Підручник для вузів за спец. «Фінанси та кредит»/ М. Савлук, К. Зуллас, А. Коряк. - К.: Либідь, 2002. – 330 с.
4. Гроші та кредит: Підручник/ За ред. Богдана Івасіва,; М-во освіти і науки України, НБУ, Терноп. акад. нар. госп.. - Тернопіль: Карт-бланш, 2000. - 510 с.
5. Oracle. Java EE. [Електронний ресурс] – Режим доступу: http://www.oracle.com/technetwork/java/javaee/overview/index.html. Дата звернення: 28.05.2016.
6. Hibernate ORM. [Електронний ресурс] – Режим доступу: http://hibernate.org/orm/. Дата звернення: 28.05.2016.
7. JSON. [Електронний ресурс] – Режим доступу: http://www.json.org/. Дата звернення: 29.05.2016.
8. Шевчук А. jQuery для начинающих. [Електронний ресурс] – Режим доступу: http://anton.shevchuk.name/jquery-book/. Дата звернення: 29.05.2016.
9. Chris Fehily. SQL Database Programming./ Chris Fehily. – М.: ДМК Прес, 2012. – 323 c.
10. Madhusudhan Konda. Just Hibernate (Covers 4.0)./ Madhusudhan Konda – М.: ДМК Прес, 2014. – 289 c.
11. James Rumbaugh, Ivar Jacobson, Grady Booch (1999). The unified modeling language reference manual (англ.). [Текст] – 284 с.
12. JavaScript diagramming library [Електронний ресурс] – Режим доступу: http://www.jointjs.com
13. Predictable state container for JavaScript apps [Електронний ресурс] Режим доступу: http://redux.js.org.
14. A JavaScript library for building user interfaces [Електронний ресурс] – Режим доступу: https://facebook.github.io/react
15. Predictable state container for JavaScript apps [Електронний ресурс] – Режим доступу: http://redux.js.org.
16. A JavaScript library for building user interfaces [Електронний ресурс] – Режим доступу: https://facebook.github.io/react.

# ДОДАТОК А

**Лістинг програмного коду**

Файл ExponentialRegression.java

package org.flush.rates.logic;

import org.flush.rates.model.Currency;

import org.flush.rates.model.Period;

import org.flush.rates.util.Util;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.List;

public class ExponentialRegression {

private List<Period> periods = new ArrayList<>();

private String currency;

private List<Integer> xList = new ArrayList<>();

private List<Double> inYList = new ArrayList<>();

private List<Integer> x2List = new ArrayList<>();

private List<Double> inY2List = new ArrayList<>();

private List<Double> xInYList = new ArrayList<>();

private List<Double> yMinusAvgYList = new ArrayList<>();

private List<Double> expList = new ArrayList<>();

private List<Double> yMinusExpList = new ArrayList<>();

private List<LocalDate> nextPeriods = new ArrayList<>();

public ExponentialRegression(String currency, List<Currency> currencyList) {

this.currency = currency;

toPeriod(currencyList, nextPeriods);

}

//calculate avg for each month

private void toPeriod(List<Currency> currencyList, List<LocalDate> nextPeriods) {

LocalDate date = null;

double sum = 0;

for (Currency currency : currencyList) {

date = Util.getLocalDate(currency.getDate());

if (date.getDayOfMonth() == date.lengthOfMonth()) {

sum += Double.parseDouble(currency.getPrice());

periods.add(new Period(date.getMonth() + "-" + date.getYear(), (sum / date.lengthOfMonth())));

sum = 0;

} else {

sum += Double.parseDouble(currency.getPrice());

}

}

if (date != null) {

if (date.getDayOfMonth() != date.lengthOfMonth())

periods.add(new Period(date.getMonth() + "-" + date.getYear(), sum / date.getDayOfMonth()));

nextPeriods.add(date.plusMonths(1));

nextPeriods.add(date.plusMonths(2));

nextPeriods.add(date.plusMonths(3));

}

}

public void calculateX() {

for (int i = 1; i <= 12; i++)

xList.add(i);

xList.add(calculateLastInteger(xList));

}

public Double calculateLastDouble(List<Double> list) {

double sum = 0;

for (double number : list)

sum += number;

return sum;

}

public Integer calculateLastInteger(List<Integer> list) {

int sum = 0;

for (int number : list)

sum += number;

return sum;

}

public void calculateInY() {

periods.forEach(item-> inYList.add(Math.log(item.getValue())));

inYList.add(calculateLastDouble(inYList));

}

public void calculateX2() {

for (int i = 1; i <= 12; i++)

x2List.add(i \* i);

x2List.add(calculateLastInteger(x2List));

}

public void calculateInY2() {

for (Double d : inYList)

inY2List.add(d \* d);

inY2List.remove(inYList.size() - 1);

inY2List.add(calculateLastDouble(inY2List));

}

public void calculateXInYList() {

for (int i = 1; i <= x2List.size(); i++)

xInYList.add(i \* inYList.get(i - 1));

xInYList.remove(xInYList.size() - 1);

xInYList.add(calculateLastDouble(xInYList));

}

public double calculateEquations() {

double n = xList.size() - 1;

double xSum = xList.get(xList.size() - 1);

double ySum = inYList.get(inYList.size() - 1);

double secondXSum = xSum;

double x2Sum = x2List.get(x2List.size() - 1);

double xInYSum = xInYList.get(xInYList.size() - 1);

double divider = (xSum / n) \* -1; // value for multiply first equation

n \*= divider;

xSum \*= divider;

ySum \*= divider;

secondXSum += n;

x2Sum += xSum;

xInYSum += ySum;

return xInYSum / x2Sum;

}

public double calculateA(double b) {

b \*= xList.get(xList.size() - 1);

double sum = inYList.get(inYList.size() - 1) - b;

return sum / (xList.size() - 1);

}

public double calculateExpRegression(double a) {

return Math.exp(a);

}

public void calculateYMinusAvgY() {

double avgY = (periods.stream().mapToDouble(Period::getValue).sum()) / (periods.size());

periods.forEach(item -> yMinusAvgYList.add(Math.pow(item.getValue() - avgY, 2)));

}

public double calculateYMinusYAvgSum() {

double sum = 0;

for (double d : yMinusAvgYList)

sum += d;

return sum;

}

public void calculateExp(double expRegressionValue, double b) {

expList.forEach(System.out::println);

}

public void calculateYMinusExp2() {

for (int i = 0; i < periods.size(); i++) {

yMinusExpList.add(Math.pow(periods.get(i).getValue() - expList.get(i), 2));

}

}

public double calculateYMinusExp2Sum() {

double sum = 0;

for (double d : yMinusExpList)

sum += d;

return sum;

}

public double determinationCoefficient() {

return 1 - (calculateYMinusExp2Sum() / calculateYMinusYAvgSum());

}

public List<Period> predictForNextMonths(double a, double b) {

double expRegressionValue = calculateExpRegression(a);

periods.add(new Period(nextPeriods.get(0).getMonth() + "-" + nextPeriods.get(0).getYear(), expRegressionValue \* Math.exp(b \* 13)));

periods.add(new Period(nextPeriods.get(1).getMonth() + "-" + nextPeriods.get(1).getYear(), expRegressionValue \* Math.exp(b \* 14)));

periods.add(new Period(nextPeriods.get(2).getMonth() + "-" + nextPeriods.get(2).getYear(), expRegressionValue \* Math.exp(b \* 15)));

return periods;

}

}

Файл PowerRegression.java

package org.flush.rates.logic;

import org.flush.rates.model.Currency;

import org.flush.rates.model.Period;

import org.flush.rates.util.Util;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.List;

public class PowerRegression {

private List<Period> periods = new ArrayList<>();

private List<Integer> xList = new ArrayList<>();

private List<Double> inXList = new ArrayList<>();

private List<Double> inX2List = new ArrayList<>();

private List<Double> inYList = new ArrayList<>();

private List<Double> inY2List = new ArrayList<>();

private List<Double> inXinYList = new ArrayList<>();

private List<Double> yMinusAvgYList = new ArrayList<>();

private List<Double> expList = new ArrayList<>();

private List<Double> yMinusExp2List = new ArrayList<>();

private List<LocalDate> nextPeriods = new ArrayList<>();

private String currency;

public PowerRegression(String currency, List<Currency> currencyList) {

this.currency = currency;

toPeriod(currencyList, nextPeriods);

//periods.forEach(System.out::println);

}

//calculate avg for each month

private void toPeriod(List<Currency> currencyList, List<LocalDate> nextPeriods) {

LocalDate date = null;

double sum = 0;

for (Currency currency : currencyList) {

date = Util.getLocalDate(currency.getDate());

if (date.getDayOfMonth() == date.lengthOfMonth()) {

sum += Double.parseDouble(currency.getPrice());

periods.add(new Period(date.getMonth() + "-" + date.getYear(), (sum / date.lengthOfMonth())));

sum = 0;

} else {

sum += Double.parseDouble(currency.getPrice());

}

}

if (date != null) {

if (date.getDayOfMonth() != date.lengthOfMonth())

periods.add(new Period(date.getMonth() + "-" + date.getYear(), sum / date.getDayOfMonth()));

nextPeriods.add(date.plusMonths(1));

nextPeriods.add(date.plusMonths(2));

nextPeriods.add(date.plusMonths(3));

}

}

public void writeXList() {

for (int i = 1; i <= 12; i++) xList.add(i);

}

public void calculateInXList() {

xList.forEach(item -> inXList.add(Math.log(item)));

}

public void calculateInY() {

periods.forEach(item -> inYList.add(Math.log(item.getValue())));

}

public void calculateInX2List() {

inXList.forEach(item -> inX2List.add(Math.pow(item, 2)));

}

public void calculateInY2List() {

inYList.forEach(item -> inY2List.add(Math.pow(item, 2)));

}

public void calculateInXInYList() {

for (int i = 0; i < inXList.size(); i++)

inXinYList.add(inXList.get(i) \* inYList.get(i));

}

public Integer calculateIntegerSum(List<Integer> list) {

int sum = 0;

for (Integer d : list)

sum += d;

return sum;

}

public Double calculateDoubleSum(List<Double> list) {

double sum = 0;

for (Double d : list)

sum += d;

return sum;

}

public double calculateB() {

double n = xList.size();

double inXSum = calculateDoubleSum(inXList);

double inXSecondSum = inXSum;

double inYSum = calculateDoubleSum(inYList);

double inX2Sum = calculateDoubleSum(inX2List);

double inY2Sum = calculateDoubleSum(inY2List);

double inXinYSum = calculateDoubleSum(inXinYList);

double divider = (inXSum / n) \* -1;

// Multiply

n \*= divider;

inXSum \*= divider;

inYSum \*= divider;

inXSecondSum += n;

inX2Sum += inXSum;

inXinYSum += inYSum;

return inXinYSum / inX2Sum;

}

public double calculateA(double b) {

double n = xList.size();

double inXSum = calculateDoubleSum(inXList);

double inYSum = calculateDoubleSum(inYList);

b \*= inXSum;

return (inYSum - b) / n;

}

public double calculatePowerRegressionValue(double a) {

return Math.exp(a);

}

public double calculateYAvg() {

double avg = 0;

for (Period p : periods)

avg += p.getValue();

return avg / periods.size();

}

public void calculateYMinusAvgY() {

double avg = calculateYAvg();

periods.forEach(item -> yMinusAvgYList.add(Math.pow(item.getValue() - avg, 2)));

}

public double calculateYMinusAvgYSum() {

double sum = 0;

for (Double p : yMinusAvgYList)

sum += p;

return sum;

}

public void calculateExp(double regression, double b) {

for (Integer i: xList)

expList.add(regression \* Math.pow(i, b));

//xList.forEach(item -> expList.add(regression \* Math.pow(item, b)));

}

public void calculateYMinusExp2() {

for (int i = 0; i < periods.size(); i++)

yMinusExp2List.add(Math.pow(periods.get(i).getValue() - expList.get(i), 2));

}

public double calculateExp2Sum() {

double sum = 0;

for (Double d : yMinusExp2List)

sum += d;

return sum;

}

public double calculateDeterminationCoefficient() {

return 1 - (calculateExp2Sum() / calculateYMinusAvgYSum());

}

public List<Period> predictForNextMonths(double a, double b) {

double powerRegressionValue = calculatePowerRegressionValue(a);

periods.add(new Period(nextPeriods.get(0).getMonth() + "-" + nextPeriods.get(0).getYear(), powerRegressionValue \* Math.pow(13, b)));

periods.add(new Period(nextPeriods.get(1).getMonth() + "-" + nextPeriods.get(1).getYear(), powerRegressionValue \* Math.pow(14, b)));

periods.add(new Period(nextPeriods.get(2).getMonth() + "-" + nextPeriods.get(2).getYear(), powerRegressionValue \* Math.pow(15, b)));

return periods;

}

}

Файл Connection.java

package org.flush.rates.network;

import org.flush.rates.util.Storage;

import org.flush.rates.util.Util;

import org.json.JSONObject;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.HttpURLConnection;

import java.net.MalformedURLException;

import java.net.ProtocolException;

import java.net.URL;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.Currency;

import java.util.List;

public class Connection {

private URL urlObj;

private HttpURLConnection connection;

private List<String> dates = new ArrayList<>();

public Connection(){}

public JSONObject sendRequestToFixerIO(String date, String currency, String symbols) {

try {

if (date == null || date.isEmpty())

if (symbols == null || symbols.isEmpty()) urlObj = new URL(String.format(Storage.FIXER\_IO\_LATEST, currency));

else urlObj = new URL(String.format(Storage.FIXER\_IO\_LATEST\_WITH\_SYMBOLS, currency, symbols));

else

if (symbols == null || symbols.isEmpty()) urlObj = new URL(String.format(Storage.FIXER\_IO\_HISTORICAL, date, currency));

else urlObj = new URL(String.format(Storage.FIXER\_IO\_HISTORICAL\_WITH\_SYMBOLS, date, currency, symbols));

connection = (HttpURLConnection) urlObj.openConnection();

connection.setRequestMethod("GET");

} catch (Exception e) {

e.printStackTrace();

}

return new JSONObject(getJSONString(connection));

}

public JSONObject sendRequestToFreeCurrencyConverter(String startDate, String endDate, String baseCurrency, String toCurrency) {

try {

urlObj = new URL("https://free.currencyconverterapi.com/api/v5/convert?q=" + baseCurrency + "\_" + toCurrency +

"," + toCurrency + "\_" + baseCurrency + "&compact=ultra&date=" + startDate + "&endDate=" + endDate);

connection = (HttpURLConnection) urlObj.openConnection();

connection.setRequestMethod("GET");

} catch (IOException e) {

e.printStackTrace();

}

return new JSONObject(getJSONString(connection));

}

public JSONObject oneRequestToFreeCurrencyConverter(String date, String baseCurrency, String toCurrency) {

try {

urlObj = new URL("https://free.currencyconverterapi.com/api/v5/convert?q=" + baseCurrency + "\_" + toCurrency +

"," + toCurrency + "\_" + baseCurrency + "&compact=ultra&date=" + date);

connection = (HttpURLConnection) urlObj.openConnection();

connection.setRequestMethod("GET");

} catch (IOException e) {

e.printStackTrace();

}

return new JSONObject(getJSONString(connection));

}

public JSONObject sendRequestToMinFin(String date) {

try {

if (date == null) urlObj = new URL(String.format(Storage.NBU\_LATEST, Storage.NBU\_TOKEN));

else urlObj = new URL(String.format(Storage.NBU\_HISTORICAL, Storage.NBU\_TOKEN, date));

connection = (HttpURLConnection) urlObj.openConnection();

connection.setRequestMethod("POST");

} catch (IOException e) {

e.printStackTrace();

}

return new JSONObject(getJSONString(connection));

}

public JSONObject sendRequestToConverter(String baseCurrency, String toCurrency) {

try {

if (toCurrency == null) urlObj = new URL(String.format(Storage.CONVERT\_ALL\_RATES, baseCurrency));

else urlObj = new URL(String.format(Storage.CONVERT\_PAIR\_RATES, baseCurrency, toCurrency));

connection = (HttpURLConnection) urlObj.openConnection();

connection.connect();

} catch (IOException e) {

e.printStackTrace();

}

return new JSONObject(getJSONString(connection));

}

public List<JSONObject> openDiapasonConnection(String startDate, String endDate, String baseCurrency, String symbols) {

LocalDate startDateObj = Util.getLocalDate(startDate);

LocalDate endDateObj = Util.getLocalDate(endDate);

List<JSONObject> jsonObjects = new ArrayList<>();

for (LocalDate date = startDateObj; date.isBefore(endDateObj.plusDays(1)); date = date.plusDays(1)) {

if (baseCurrency.equals("UAH")) jsonObjects.add(sendRequestToMinFin(date.toString()));

else jsonObjects.add(sendRequestToFixerIO(date.toString(), baseCurrency, symbols));

dates.add(date.toString());

}

return jsonObjects;

}

public List<JSONObject> predictionsRequests(LocalDate startDate, LocalDate endDate, String baseCurrency, String toCurrency) {

LocalDate day = endDate;

LocalDate start = startDate;

int difference = day.getDayOfMonth() / 8;

System.out.println(difference);

System.out.println(endDate);

List<URL> urls = new ArrayList<>();

for (int i = 0; i < difference; i++) {

try {

start = startDate.plusDays(8);

urls.add(new URL("https://free.currencyconverterapi.com/api/v5/convert?q=" + baseCurrency + "\_" + toCurrency +

"," + toCurrency + "\_" + baseCurrency + "&compact=ultra&date=" + startDate.toString() + "&endDate=" + start.toString()));

} catch (IOException e) {

e.printStackTrace();

}

startDate = startDate.plusDays(9);

}

int secondDifference = endDate.getDayOfMonth() - start.getDayOfMonth() - 1;

if (secondDifference > 0) {

try {

urls.add(new URL("https://free.currencyconverterapi.com/api/v5/convert?q=" + baseCurrency + "\_" + toCurrency +

"," + toCurrency + "\_" + baseCurrency + "&compact=ultra&date=" + endDate.minusDays(secondDifference) + "&endDate=" + endDate.toString()));

} catch (MalformedURLException e) {

e.printStackTrace();

}

}

List<JSONObject> jsonObjects = new ArrayList<>();

for (URL url : urls) {

try {

connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET");

} catch (IOException e) {

e.printStackTrace();

}

jsonObjects.add(new JSONObject(connection));

}

return jsonObjects;

}

private String getJSONString(HttpURLConnection connection) {

StringBuffer responseBuffer = new StringBuffer();

try (BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));) {

String inputLine;

while ((inputLine = in.readLine()) != null)

responseBuffer.append(inputLine);

} catch (IOException e) {

e.printStackTrace();

}

return responseBuffer.toString();

}

public List<String> getDates() {

return dates;

}

}

Файл Currency.java

package org.flush.rates.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

public class Currency {

private String baseCurrencyName;

private String date;

private String price;

private String toCurrency;

private String priceSold;

public Currency() {

}

public Currency(String baseCurrencyName, String date, String toCurrency, String price, String priceSold) {

this.baseCurrencyName = baseCurrencyName;

this.date = date;

this.toCurrency = toCurrency;

this.price = price;

this.priceSold = priceSold;

}

public String getBaseCurrencyName() {

return baseCurrencyName;

}

public String getDate() {

return date;

}

public String getPrice() {

return price;

}

public String getToCurrency() {

return toCurrency;

}

public String getPriceSold() {

return priceSold;

}

public void setBaseCurrencyName(String baseCurrencyName) {

this.baseCurrencyName = baseCurrencyName;

}

public void setDate(String date) {

this.date = date;

}

public void setPrice(String price) {

this.price = price;

}

public void setToCurrency(String toCurrency) {

this.toCurrency = toCurrency;

}

public void setPriceSold(String priceSold) {

this.priceSold = priceSold;

}

@Override

public String toString() {

return "Currency{" +

"baseCurrencyName='" + baseCurrencyName + '\'' +

", date='" + date + '\'' +

", price='" + price + '\'' +

", toCurrency='" + toCurrency + '\'' +

", priceSold='" + priceSold + '\'' +

'}';

}

}

Файл Period.java

package org.flush.rates.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement

public class Period {

private String period;

private Double value;

public Period() {

}

public Period(String period, Double value) {

this.period = period;

this.value = value;

}

public String getPeriod() {

return period;

}

public void setPeriod(String period) {

this.period = period;

}

public Double getValue() {

return value;

}

public void setValue(Double value) {

this.value = value;

}

@Override

public String toString() {

return "Period{" +

"period='" + period + '\'' +

", value=" + value +

'}';

}

}

Файл FixerResource.java

package org.flush.rates.resources;

import org.flush.rates.errors.ErrorMessages;

import org.flush.rates.logic.Converter;

import org.flush.rates.logic.DynamicsIndicatorsService;

import org.flush.rates.logic.ExponentialRegression;

import org.flush.rates.logic.PowerRegression;

import org.flush.rates.model.Currency;

import org.flush.rates.model.Period;

import org.flush.rates.network.Connection;

import org.flush.rates.util.FormParamChecker;

import org.flush.rates.util.JSONParsing;

import org.flush.rates.util.Util;

import org.json.JSONObject;

import javax.ws.rs.\*;

import javax.ws.rs.core.MediaType;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

import java.util.List;

@Path("resource")

public class FixerResource {

private static final JSONParsing jsonParsing = new JSONParsing();

private FormParamChecker checker = new FormParamChecker();

private Connection connection = new Connection();

private Converter converter = new Converter();

private JSONObject temp = null;

@POST

@Path("/uahLive")

@Produces(MediaType.APPLICATION\_JSON)

public List<Currency> getLiveUAH(@FormParam("liveSend") String currency, @FormParam("toCurrency") String toCurrency) {

String baseCurrency = checker.formatCurrency(currency);

if (!checker.isRate(baseCurrency))

throw new WebApplicationException(ErrorMessages.baseCurrencyException());

toCurrency = checker.formatCurrency(toCurrency);

if (!checker.isRate(toCurrency))

throw new WebApplicationException(ErrorMessages.rateException());

List<Currency> toReturn = new ArrayList<>();

DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd");

LocalDateTime now = LocalDateTime.now();

LocalDate localDate = Util.getLocalDate(dtf.format(now));

//localDate = localDate.minusDays(1);

/\*if (baseCurrency.equals("UAH")) {

temp = connection.sendRequestToMinFin(null);

toReturn.addAll(jsonParsing.parseFromMinFin(temp, null));

temp = connection.sendRequestToMinFin(localDate.toString());

toReturn.addAll(jsonParsing.parseFromMinFin(temp, null));

} else if (!baseCurrency.equals("UAH")) {

temp = connection.sendRequestToFixerIO(null, baseCurrency, null);

toReturn.addAll(jsonParsing.parseFromFixerIO(temp));

temp = connection.sendRequestToFixerIO(localDate.minusDays(1).toString(), baseCurrency, null);

toReturn.addAll(jsonParsing.parseFromFixerIO(temp));

}\*/

JSONObject object = connection.oneRequestToFreeCurrencyConverter(localDate.toString(), toCurrency, baseCurrency);

toReturn.addAll(jsonParsing.parseOne(object, toCurrency, baseCurrency));

object = connection.oneRequestToFreeCurrencyConverter(localDate.minusDays(1).toString(), toCurrency, baseCurrency);

toReturn.addAll(jsonParsing.parseOne(object, toCurrency, baseCurrency));

double difference = Double.parseDouble(toReturn.get(0).getPrice()) - Double.parseDouble(toReturn.get(1).getPrice());

toReturn.remove(1);

toReturn.get(0).setPrice(String.valueOf(Util.round(Double.parseDouble(toReturn.get(0).getPrice()), 3)));

toReturn.get(0).setPriceSold(String.valueOf(Util.round(difference, 3)));

toReturn.get(0).setToCurrency(toCurrency);

return toReturn;

}

@POST

@Path("/byDate")

@Produces(MediaType.APPLICATION\_JSON)

public List<Currency> getIt(@FormParam("inputDate") String date,

@FormParam("selectBaseCurrency") String currency,

@FormParam("currenciesByDate") String currenciesByDate) {

String baseCurrency = checker.formatCurrency(currency);

currenciesByDate = currenciesByDate.toUpperCase();

if (!checker.isRate(baseCurrency))

throw new WebApplicationException(ErrorMessages.baseCurrencyException());

String symbols;

if (!checker.checkSymbols(currenciesByDate)) symbols = null;

else symbols = checker.formatSymbols(currenciesByDate);

if (baseCurrency.equals("UAH")) {

temp = connection.sendRequestToMinFin(date);

return jsonParsing.parseFromMinFin(temp, date);

} else if (!baseCurrency.equals("UAH")) {

temp = connection.sendRequestToFixerIO(date, baseCurrency, symbols);

return jsonParsing.parseFromFixerIO(temp);

}

throw new WebApplicationException(ErrorMessages.unknownException());

}

@POST

@Path("/converter")

@Produces(MediaType.APPLICATION\_JSON)

public List<Currency> convert(@FormParam("converterCurrency") String currency,

@FormParam("toRates") String toRate,

@FormParam("converterDate") String date,

@FormParam("inputQuantity") String quantity) {

//validate base currency

String baseCurrency = checker.formatCurrency(currency);

if (!checker.isRate(baseCurrency))

throw new WebApplicationException(ErrorMessages.baseCurrencyException());

//validate to currency

String to = checker.formatCurrency(toRate);

if (!checker.isRate(to))

throw new WebApplicationException(ErrorMessages.rateException());

//validate quantity

if (!checker.isNumeric(quantity))

throw new WebApplicationException(ErrorMessages.numericException());

if (date == null && baseCurrency.equals("UAH")) {

temp = connection.sendRequestToConverter(baseCurrency, to);

return converter.convert(Util.filterOne(jsonParsing.parseOneConverterRate(temp), to), quantity);

}

if (date == null && !baseCurrency.equals("UAH"))

throw new WebApplicationException(ErrorMessages.dateException());

if (to.equals("UAH")) {

temp = connection.sendRequestToMinFin(date);

return converter.convert(Util.filterOne(jsonParsing.parseFromMinFin(temp, date), baseCurrency), quantity);

} else {

temp = connection.sendRequestToFixerIO(date, baseCurrency, to);

return converter.convert(jsonParsing.parseFromFixerIO(temp), quantity);

}

/\*if (!date.isEmpty()) {

if (baseCurrency.equals("UAH")) {

temp = connection.sendRequestToMinFin(date);

return converter.convert(Util.filterResultList(jsonParsing.parseFromMinFin(temp, date), symbols), quantity);

} else if (!baseCurrency.equals("UAH")) {

temp = connection.sendRequestToFixerIO(date, baseCurrency, symbols);

return converter.convert(jsonParsing.parseFromFixerIO(temp), quantity);

}

}\*/

/\*if (symbols == null) {

temp = connection.sendRequestToConverter(baseCurrency, null);

return converter.convert(jsonParsing.parseFromConverterAll(temp), quantity);

} else if (checker.checkSymbols(symbols)) {

if (symbols.trim().length() == 3) {

temp = connection.sendRequestToConverter(baseCurrency, symbols);

return converter.convert(jsonParsing.parseOneConverterRate(temp), quantity);

}

else if (symbols.length() >= 7) {

temp = connection.sendRequestToConverter(baseCurrency, null);

return converter.convert(jsonParsing.parseManyConverterRates(temp, symbols), quantity);

}

}\*/

//throw new WebApplicationException(ErrorMessages.unknownException());

}

@POST

@Path("/hotChart")

@Produces(MediaType.APPLICATION\_JSON)

public List<Currency> buildHotChart(@FormParam("baseChartCurrency") String baseChartCurrency,

@FormParam("toChartCurrency") String chartCurrency) {

//validate base currency

String baseCurrency = checker.formatCurrency(baseChartCurrency);

System.out.println("Base: " + baseChartCurrency);

if (!checker.isRate(baseCurrency))

throw new WebApplicationException(ErrorMessages.baseCurrencyException());

chartCurrency = checker.formatCurrency(chartCurrency);

System.out.println(chartCurrency);

if (!checker.isRate(chartCurrency.toUpperCase()))

throw new WebApplicationException(ErrorMessages.symbolsException());

DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd");

LocalDateTime now = LocalDateTime.now();

LocalDate localDate = Util.getLocalDate(dtf.format(now));

String endDate = localDate.toString();

localDate = localDate.minusDays(6);

String startDate = localDate.toString();

//List<JSONObject> jsonObjects = connection.openDiapasonConnection(startDate, endDate, baseCurrency, chartCurrency.toUpperCase());

JSONObject jsonObjects = connection.sendRequestToFreeCurrencyConverter(startDate, endDate, baseCurrency, chartCurrency);

return jsonParsing.parse(jsonObjects, chartCurrency, baseCurrency);

}

@POST

@Path("/chart")

@Produces(MediaType.APPLICATION\_JSON)

public List<Currency> buildChart(@FormParam("baseChartCurrency") String baseChartCurrency, @FormParam("toChartCurrency") String chartCurrency,

@FormParam("firstChartDate") String startDate,

@FormParam("secondChartDate") String endDate) {

//validate base currency

String baseCurrency = checker.formatCurrency(baseChartCurrency);

if (!checker.isRate(baseCurrency))

throw new WebApplicationException(ErrorMessages.baseCurrencyException());

chartCurrency = checker.formatCurrency(chartCurrency);

if (!checker.isRate(chartCurrency.toUpperCase()))

throw new WebApplicationException(ErrorMessages.symbolsException());

if (startDate.isEmpty() || endDate.isEmpty() || !Util.checkDiapason(startDate, endDate))

throw new WebApplicationException(ErrorMessages.dateException());

List<JSONObject> jsonObjects = connection.openDiapasonConnection(startDate, endDate, baseCurrency, chartCurrency.toUpperCase());

return jsonParsing.parseDiapason(jsonObjects, baseCurrency, connection.getDates(), chartCurrency.toUpperCase());

}

@POST

@Path("/dynamics")

@Produces(MediaType.APPLICATION\_JSON)

public List<Period> predictRatesValue(@FormParam("predictionCurrency") String toDynamicCurrency, @FormParam("typeOfPrediction") String algorithm) {

if (algorithm == null || algorithm.isEmpty())

throw new WebApplicationException(ErrorMessages.unknownException());

DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd");

LocalDateTime now = LocalDateTime.now();

LocalDate localDate = Util.getLocalDate(dtf.format(now));

String endDate = localDate.toString();

localDate = localDate.minusDays(localDate.getDayOfMonth() - 1);

String firstDate = localDate.toString();

//String firstDate = localDate.minusMonths(12).toString();

System.out.println("Start date: " + firstDate);

System.out.println("End date: " + endDate);

List<JSONObject> jsonObjects = connection.openDiapasonConnection(firstDate, endDate, "UAH", toDynamicCurrency.toUpperCase());

if (algorithm.equals("Dynamics Indicators")) {

DynamicsIndicatorsService service = new DynamicsIndicatorsService(toDynamicCurrency.toUpperCase());

service.calculateDynamicsTable(jsonParsing.parseDiapason(jsonObjects, "UAH", connection.getDates(), toDynamicCurrency.toUpperCase()));

service.calculateBasicDynamics();

return service.predict();

} else if (algorithm.equals("Non-Linear Regression")) {

List<Currency> currencies = jsonParsing.parseDiapason(jsonObjects, "UAH", connection.getDates(), toDynamicCurrency.toUpperCase());

PowerRegression powerRegression = new PowerRegression(toDynamicCurrency, currencies);

powerRegression.writeXList();

powerRegression.calculateInXList();

powerRegression.calculateInY();

powerRegression.calculateInX2List();

powerRegression.calculateInY2List();

powerRegression.calculateInXInYList();

double bPr = powerRegression.calculateB();

double aPr = powerRegression.calculateA(bPr);

double regression = powerRegression.calculatePowerRegressionValue(aPr);

powerRegression.calculateYMinusAvgY();

powerRegression.calculateExp(regression, bPr);

powerRegression.calculateYMinusExp2();

//powerRegression.yMinusExp2List.forEach(System.out::println);

//Exponential

ExponentialRegression exponentialRegression = new ExponentialRegression(toDynamicCurrency, currencies);

exponentialRegression.calculateX();

exponentialRegression.calculateInY();

exponentialRegression.calculateX2();

exponentialRegression.calculateInY2();

exponentialRegression.calculateXInYList();

exponentialRegression.calculateYMinusAvgY();

double exB = exponentialRegression.calculateEquations();

double exA = exponentialRegression.calculateA(exB);

exponentialRegression.calculateExp(exponentialRegression.calculateExpRegression(exA),

exponentialRegression.calculateEquations());

exponentialRegression.calculateYMinusExp2();

double expDeterminationCoeff = exponentialRegression.determinationCoefficient();

double prDeterminationCoeff = powerRegression.calculateDeterminationCoefficient();

if (expDeterminationCoeff > prDeterminationCoeff) {

return exponentialRegression.predictForNextMonths(exA, exB);

} else {

return powerRegression.predictForNextMonths(aPr, bPr);

}

}

throw new WebApplicationException(ErrorMessages.unknownException());

}

}