

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
КЕРУВАННЯ ПАРАМЕТРАМИ ПОЗИК
RESEARCH AND SOFTWARE IMPLEMENTATION OF
METHODS FOR MANAGING LOAN PARAMETERS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-213м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

ЧУМАК М.В.

(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(КОД І НАЙМЕНУВАННЯ)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЧУМАКА Микити Вікторовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів керування параметрами позик. Research and Software Implementation of Methods for Managing Loan Parameters

керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “7” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 01 грудня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні рішення щодо реалізації програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 08 ” вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2 тиждень	Розділ 1
3	Дослідження проблеми	3-5 тижні	Розділ 2
4	Проектування та розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис)

Микита ЧУМАК

(Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Валерій ЛЬОВКІН

(Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
106 с., 6 табл., 23 рис., 3 дод., 11 джерел.

ПОЗИКА, ВІДСОТКОВА СТАВКА, ПОЗИЧАЛЬНИК, КРЕДИТОР,
ПРОГНОЗУВАННЯ, АНСАМБЛІ ДЕРЕВ РІШЕНЬ.

Об'єкт роботи – процес підтримки надання позик. Предмет роботи – методи визначення параметрів фінансування позик. Мета роботи – розробити програмне забезпечення підтримки надання позик на основі фінансування їх фізичними особами для зменшення вимог до визначення параметрів фінансування позик.

Матеріали, методи та технічні засоби: мова Python, бібліотека XGBoost, фреймворк Django, система PostgreSQL.

Результати. Програма призначена для підтримки надання позик між фізичними особами, коли фінансові ресурси для позик надаються не банківськими установами, а безпосередньо фізичними особами, які згодні вкласти їх на фінансування певної діяльності інших осіб, отримуючи в підсумку відсотки за надані фінанси. Програма надає можливість формувати заявки на позики, опрацьовувати ці заявки, проводити процес збору коштів, шукати придатні для фінансування позики.

Висновки. Наукова новизна роботи полягає в методі керування параметрами позик, який характеризується використанням дерев рішень на основі створення ансамблів методом XGBoost для прогнозування відсоткової ставки позики за значеннями вхідних ознак, що дозволяє встановлювати варіативні значення параметрів позик таким чином, щоб сприяти результативності процесу збору коштів від фізичних осіб.

Галузь використання. Надання позик користувачам на основі фінансування цих позик окремими фізичними особами.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 106 pages, 6 tables, 23 figures, 3 appendixes, 11 sources.

LOAN, INTEREST RATE, BORROWER, LENDER, FORECASTING, ENSEMBLE OF DECISION TREES.

The object of the work is the process of supporting the provision of loans. The subject of the work is the methods of determining loan financing parameters. The purpose of the work is to develop software to support the granting of loans based on their financing by individuals to reduce the requirements for determining the parameters of loan financing.

Materials, methods and technical means: Python language, XGBoost library, Django framework, PostgreSQL system.

The results. The program is designed to support the provision of loans between individuals, when financial resources for loans are not provided by banking institutions, but directly by individuals who agree to invest them in financing certain activities of other individuals, ultimately receiving interest on the provided finances. The program provides an opportunity to form loan applications, process these applications, carry out the fundraising process, and search for loans suitable for financing.

Conclusions. The scientific novelty of the work consists in the method of managing loan parameters, which is characterized by the use of decision trees based on the creation of ensembles using the XGBoost method for forecasting the interest rate of a loan based on the values of input characteristics, which allows decision-makers to set variable values of loan parameters in such a way to contribute to the effectiveness of the process of collecting funds from individuals.

Field of use. Providing loans to users based on the financing of these loans by individuals.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	10
1.1 Визначення параметрів позик за запитом.....	10
1.2 Програмний застосунок LendingClub.....	14
1.3 Порівняння вебзастосунку підтримки надання позик з прототипом	16
1.4 Висновки за розділом 1	17
2 Матеріали і методи.....	18
2.1 Задача керування параметрами позик.....	18
2.2 Метод керування параметрами позик	21
2.3 Висновки за розділом 2	23
3 Проєктування програмного забезпечення	24
3.1 Вибір мови та засобів розробки.....	24
3.2 Моделювання роботи з програмним забезпеченням за функціональними вимогами.....	25
3.3 Проєктування структури бази даних.....	29
3.4 Висновки за розділом 3	42
4 Основні рішення щодо реалізації програмного забезпечення	43
4.1 Порядок роботи з позиками	43
4.2 Рішення з реалізації вебзастосунку	44
4.3 Висновки за розділом 4	60
5 Експлуатація, тестування та експериментальне дослідження програми	61
5.1 Опис програми.....	61
5.2 Експлуатація програми.....	61
5.3 Експериментальне дослідження	65
5.4 Висновки за розділом 5	69
Висновки	71
Перелік джерел посилань	72

Додаток А Технічне завдання	73
Додаток Б Текст програми	78
Додаток В Слайди презентації.....	99

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

XGBoost – eXtreme Gradient Boosting;

БД – база даних.

ВСТУП

Надання позик зазвичай забезпечується банківськими установами, інвестиційними фондами. Ці установи націлюють свою діяльність на надання фінансових ресурсів за певну сплату. Проте такий варіант є достатньо традиційним, відповідно і відсоткові ставки за таке фінансування є доволі обмеженими, а позичальник може не відповідати певній частині вимог, тоді не маючи можливість отримати кошти. З іншого боку відреагувати на такі ситуації можна, наприклад, збільшенням відсоткової ставки. Проте для цього потрібне інше джерело фінансування, адже банки в такій діяльності є обмеженими. Таким іншим джерелом фінансування і можуть бути фізичні особи. Тоді параметри позик, на основі яких має відбуватися фінансування, а також відсіювати непридатні пропозиції має сервіс, а фінансування надавати фізичні особи.

Для підтримки саме описаних принципів роботи і направлене програмне забезпечення, яке має бути розроблено в цій роботі. З іншого боку для зменшення навантаження на працівників сервісу необхідно надати можливість програмного прогнозування відсоткової ставки за параметрами позики, після чого представник сервісу може або змінити, або підтвердити пропонуване значення, контролюючи отриманий результат. Відповідно надання таких засобів підтримки надання позик є важливим. Підкреслює актуальність цього напрямку наявність потужної платформи LendingClub [1], яка є дуже популярної в Сполучених Штатах для надання позик саме на основі описаних принципів. Однак інтегрування засобів прогнозування в процес керування позиками повинно додатково зменшити витрати на роботу сервісу, що має сприяти активізації надання послуг.

Метою роботи є розробка програмного забезпечення підтримки надання позик на основі фінансування їх фізичними особами для зменшення вимог до визначення параметрів фінансування позик.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Визначення параметрів позик за запитом

Зважаючи на те, що у даній сфері існує достатньо потужне рішення, яким є програмна платформа і застосунок LendingClub, при вивченні параметрів позик за запитом було вирішено вивчити ті параметри, які визначаються на даній платформі. Відповідно було використана вибірка даних [2], яка включає ці дані.

Позики характеризуються рядом наступних показників:

- показник `emp_title` визначає назву посади на роботі, яку займає автор запиту щодо отримання позики (наприклад, менеджер з продажів, розробник програмного забезпечення, інженер, директор);

- показник `emp_length` – кількість років, протягом якої працівник виконує цю роботу, обмежуючи задане значення максимально 10 (тривалість більше 10 років позначається значенням 10);

- показник `state` – код, що позначає адресу проживання позичальника (зокрема позначення штату на основі 2 літер);

- показник `homeownership` – наявне право власності на житло у позичальника (орендоване житло, власне житло або використання іпотечної програми);

- показник `annual_income` – сумарний річний дохід позичальника;

- показник `verified_income` – варіант верифікації отриманих даних про дохід від позичальника (перевірено, не перевірено, перевірено джерело);

- показник `debt_to_income` – співвідношення, яке визначено між сумарними боргами позичальника та його сумарними доходами;

- показник `annual_income_joint` – сумарний річний дохід усіх позичальників, які задіяні в позиції: наявність цього показника додатково до `annual_income` обумовлена наявністю випадків, коли позику беруть на виконання певної справи одразу декілька позичальників разом;

– показник `verification_income_joint` – варіант верифікації отриманих даних про сумарний дохід від позичальників (перевірено, не перевірено, перевірено джерело);

– показник `debt_to_income_joint` – співвідношення, яке визначено між сумарними боргами всіх позичальників, задіяних у позиці, та сумарними доходами всіх позичальників, задіяних у позиці;

– показник `delinq_2y` – кількість випадків прострочених платежів за відкритими позиками (кредитами), що трапилась з позичальником за останні 2 роки;

– показник `months_since_last_delinq` – кількість місяців, яка минула з останнього випадку прострочення платежів за відкритими позиками (кредитами), що трапилась з позичальником;

– показник `earliest_credit_line` – рік, у який було отримано першу позику або відкрито кредитну лінію цим позичальником;

– показник `inquiries_last_12m` – кількість сплачень, які мав виконати позичальник за останній рік;

– показник `total_credit_lines` – загальна кількість позик або кредитних ліній, якими був обтяжений позичальник за весь час (виходячи з наданої ним кредитної історії);

– показник `open_credit_lines` – кількість позик або кредитних ліній, якими сумарно обтяжений позичальник на поточний момент (на момент запиту позики);

– показник `total_credit_limit` – сумарний обсяг кредитних, позичених коштів, які доступні позичальнику на поточний момент (за особливостями обчислення іпотека не входить в це значення, проте кредитні ліміти за банківськими картками входять у максимальному можливому значенні);

– показник `total_credit_utilized` – сумарний обсяг кредитних, позичених коштів, які використані позичальником на поточний момент (за особливостями обчислення іпотека не входить в це значення);

- показник `num_collections_last_12m` – кількість зборів, що припадали на останній рік;
- показник `num_historical_failed_to_pay` – кількість випадків, які сталися з позичальником та полягали в тому, що він не зміг сплатити за призначений платіж за позиною або кредитом;
- показник `months_since_90d_late` – останній випадок, коли позичальник прострочив призначений платіж мінімум на 90 днів, визначений кількістю місяців, яка з того моменту минула;
- показник `current_accounts_delinq` – кількість випадків, за якими позичальник на момент запиту має прострочені платежі, виражена кількістю таких рахунків (кредитів або позик);
- показник `total_collection_amount_ever` – сумарний обсяг коштів, який необхідно було позичальнику сплатити;
- показник `current_installment_accounts` – кількість рахунків у позичальника, які передбачають фіксовані сплати за ними;
- показник `accounts_opened_24m` – кількість нових позик або кредитів, які були отримані (відкриті) позичальником протягом 2 останніх років;
- показник `months_since_last_credit_inquiry` – кількість місяців, яка минула з моменту, коли позичальник востаннє оформив нову позику або кредит;
- показник `num_satisfactory_accounts` – кількість закритих позик або кредитів, які мав позичальник;
- показник `num_accounts_120d_past_due` – кількість рахунків (позик або кредитів), за якими позичальник прострочив призначений платіж мінімум на 120 днів;
- показник `num_accounts_30d_past_due` – кількість рахунків (позик або кредитів), за якими позичальник прострочив призначений платіж мінімум на 30 днів;
- показник `num_active_debit_accounts` – кількість наявних у позичальника активних на момент запиту банківських карт;

- показник `total_debit_limit` – обсяг коштів, що визначає сумарний ліміт, встановлений за всіма банківськими картками;
- показник `num_total_cc_accounts` – кількість кредитних карток, які за весь час мав позичальник;
- показник `num_open_cc_accounts` – кількість кредитних карток, які на момент запиту має позичальник загалом;
- показник `num_cc_carrying_balance` – кількість кредитних карток позичальника, які на момент запиту мають на своєму рахунку певний баланс;
- показник `num_mort_accounts` – кількість відкритих випадків іпотеки у позичальника;
- показник `account_never_delinq_percent` – відсоток випадків кредитів або позик, за якими позичальник жодного разу не стикався з простроченням платежів, з загальної кількості випадків кредит або позик, які були у цього позичальника за весь час;
- показник `tax_liens` – податкова застава;
- показник `public_record_bankrupt` – кількість випадків, коли щодо позичальника було оголошено банкрутство;
- показник `loan_purpose` – призначення запитуваної позичальником позики, визначене відповідною категорією (наприклад, ведення бізнесу, переїзд, лікування, автомобіль, відпочинок);
- показник `application_type` – тип запиту на позику, який може бути або сумісним за декількома позичальниками, або індивідуальним;
- показник `loan_amount` – сума, яку намагається позичити позичальник;
- показник `term` – строк, на який намагається позичити позичальник цю позику, виражений у місяцях;
- показник `interest_rate` – відсоткова ставка за позикою, яка була присвоєна;
- показник `installment` – розмір платежу, який кожного місяця мав отримувати надавач позики (користувач, який профінансував позику);
- показник `grade` – оцінка, яку було присвоєно сервісом позиці;

- показник `sub_grade` – деталізована оцінка, яка була присвоєна сервісом позиці;
- показник `issue_month` – кількість місяців, яка на поточний момент пройшла з моменту розміщення позики;
- показник `loan_status` – поточний стан позики;
- показник `initial_listing_status` – початковий стан позики;
- показник `disbursement_method` – метод виплат за позикою (платіжною картою або напряму готівкою);
- показник `balance` – поточний баланс, який відповідає позиці;
- показник `paid_total` – сумарний обсяг коштів, який на поточний момент сплатив за позикою позичальник;
- показник `paid_principal` – різниця, яка є між початковою сумою запиту позики та поточним залишком за кредитом;
- показник `paid_interest` – обсяг відсотків, які на даний момент вже було сплачено позичальником за позикою;
- показник `paid_late_fees` – сплачений обсяг коштів за комісіями, які були висунуті за простроченнями позичальнику [2]-[3].

1.2 Програмний застосунок LendingClub

LendingClub (рис. 1.1) є платформою, яка започаткувала отримання персональних позик від людей до людей [1]. Тобто це платформа, яка дозволяє отримувати позики не від банку чи від іншої фінансової установи, а від інших людей, які є користувачами платформи та готові надати такі ресурси. Платформа досягла величезного розвитку [4], проте на території України використати її можливості неможливо. Тобто фактично навіть звернутися до вебсайту даного сервісу не є можливим. Відповідно для формування запитів на позики, для надання власних коштів потрібно також знаходитися в Сполучених Штатах Америки.

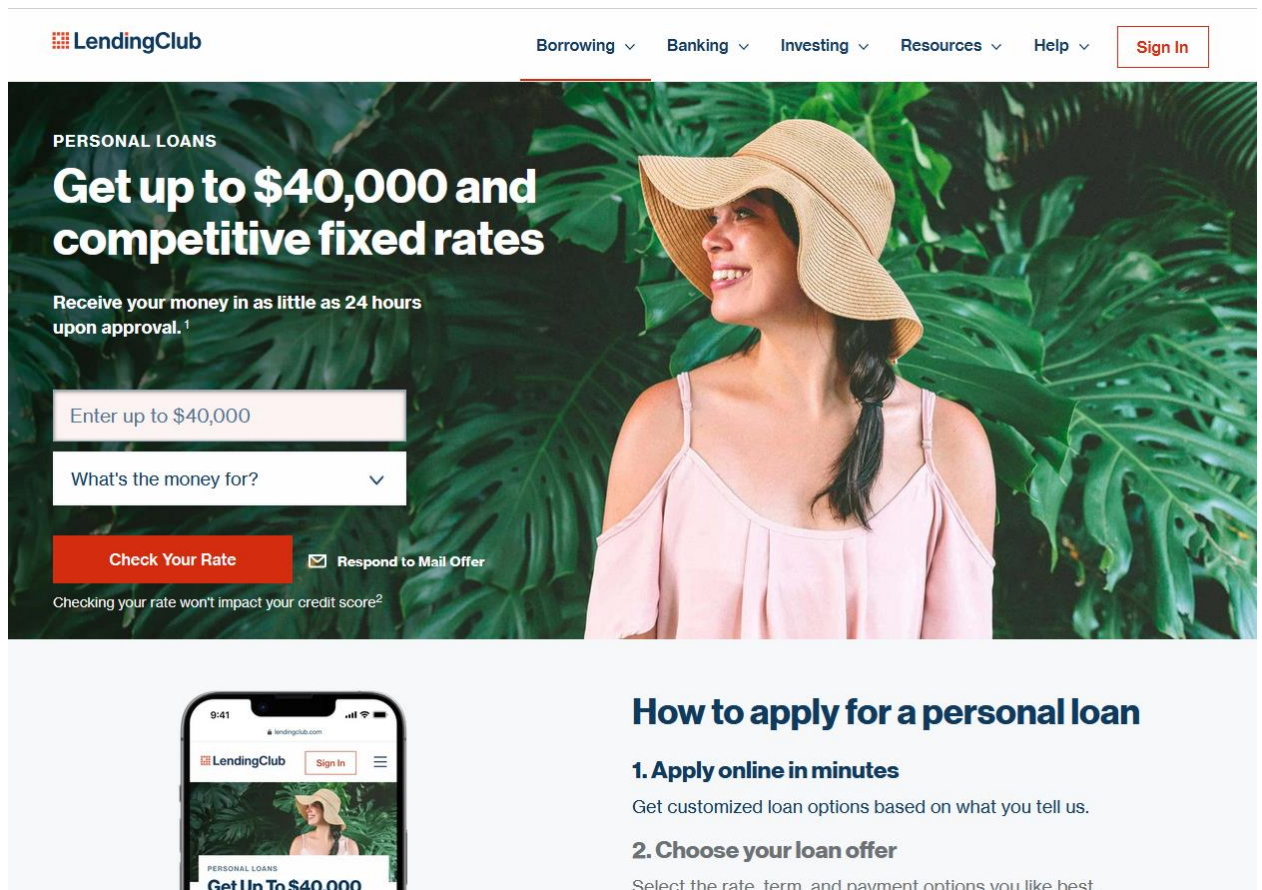


Рисунок 1.1 – Вебсайт LendingClub [1]

LendingClub дозволяє отримувати позики шляхом фінансування їх іншими людьми як за індивідуальними запитами, так і за запитами групи позичальників, направляючи отримані кошти на один з наступних варіантів призначення:

- перебудова, ремонт житла, будинку;
- консолідація боргів;
- надзвичайні ситуації;
- оплата медичних рахунків за лікування;
- відпочинок;
- проведення весілля;
- витрати на ведення бізнесу;
- фінансування придбання автомобілів, мотоциклів, фургонів, човнів [5].

Кожна позика може в підсумку отримати відсоткову ставку в проміжку від 9,06 % до 35,99 %, платіж сервісу за його послуги складає від 3 % до 8 %, отримати позику можна на суму, що знаходиться в проміжку від 1 до 40 тисяч американських доларів, відповідно платежі за погашення позики мають бути здійснені за час від 2 до 6 років [5].

1.3 Порівняння вебзастосунку підтримки надання позик з прототипом

Зважаючи на особливості предметної області, було виконано огляд можливостей платформи LendingClub, які далі на основі відповідного вебзастосунку було порівняно з запланованими можливостями для реалізації у цій роботі (табл. 1.1).

Таблиця 1.1 – Порівняння вебзастосунку підтримки надання позик з прототипом

Показник	LendingClub	ChumakLending
Джерело фінансування позик	Власні кошти користувачів	Власні кошти користувачів
Типи позик	Індивідуальні, сумісні	Індивідуальні, сумісні
Можливості соціальної мережі для взаємодії з позичальниками	—	+
Можливість роботи користувачів з України	—	+
Програмне визначення відсоткової ставки за позиками під час формування запиту	—	+

Концепцію платформи LendingClub вирішено взяти за основу в розроблюваному вебзастосунку підтримки надання позик. Відмінностями вебзастосунку є надання можливостей соціальної мережі для взаємодії з позичальниками, щоб отримати більше інформації про них, можливості програмного визначення відсоткової ставки за позиками під час формування запиту, що дозволяє отримати додаткову інформацію для прийняття рішень ще до подання остаточної заявки. Окрім того надається можливість роботи користувачів з України, а LendingClub – засіб для Сполучених Штатів Америки.

1.4 Висновки за розділом 1

Завдання роботи:

- проаналізувати програми підтримки надання позик;
- розробити метод керування параметрами позик;
- виконати проєктування програми підтримки надання позик;
- реалізувати програму підтримки надання позик;
- виконати експериментальне дослідження прогнозування відсоткової ставки за позиками.

Аналіз програм підтримки надання позик виконано у розділі 1, результатом чого є визначення параметрів, які враховуються та визначаються під час надання позик, особливостей платформи LendingClub, встановлення особливостей розробки.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Задача керування параметрами позик

Задача керування параметрами позик полягає у встановленні тих параметрів позик, які безпосередньо впливають на фінансові показники, за якими позичальник отримує фінансові ресурси для реалізації своєї діяльності. Одним з таких параметрів є відсоткова ставка за позикою. При опрацюванні позики саме відсоткова ставка є одним з критичних параметрів, які мають бути встановлені за позикою. Окрім того це може бути і оцінка позики. Однак оцінка безпосередньо впливає на відсоткову ставку, тому в роботі вирішено зробити акцент на визначенні саме відсоткової ставки як вихідного параметра позики після вирішення задачі керування параметрами позик.

Відповідно в такому варіанті задача керування параметрами позик пов'язана з тим, щоб визначити за відомими значеннями набору ознак, якими характеризується позика, відсоткову ставку, яка буде цій позиці присвоєна. Для вирішення цього необхідно створити модель прогнозування, яка за відомими значеннями параметрів позик, які є вхідними ознаками, буде прогнозувати як вихідну ознаку відсоткову ставку.

Для ряду позик відомими є значення вхідних параметрів позик та значення відсоткової ставки, які були у відповідь встановлені менеджерами сервісу підтримки позик. Використовуючи ці дані для навчання моделі, необхідно отримати модель, яка зможе виконувати прогнозування відсоткової ставки для позик, які представлені тільки значеннями вхідних ознак.

Набір вхідних ознак, якими має описуватися позика в задачі керування параметрами позик, включає наступні:

- назва посади на роботі, яку займає автор запиту щодо отримання позики;
- кількість років, протягом якої працівник виконує цю роботу (максимально 10);
- код, що позначає адресу проживання позичальника;

- наявне право власності на житло у позичальника;
- сумарний річний дохід позичальника;
- варіант верифікації отриманих даних про дохід від позичальника;
- співвідношення, яке визначено між сумарними боргами позичальника та його сумарними доходами;
- сумарний річний дохід усіх позичальників, які задіяні в позиці;
- варіант верифікації отриманих даних про сумарний дохід від позичальників;
- співвідношення, яке визначено між сумарними боргами всіх позичальників, задіяних у позиці, та сумарними доходами всіх позичальників, задіяних у позиці;
- кількість випадків прострочених платежів за відкритими позиками (кредитами), що трапилась з позичальником за останні 2 роки;
- кількість місяців, яка минула з останнього випадку прострочення платежів за відкритими позиками (кредитами), що трапилась з позичальником;
- рік, у який було отримано першу позику або відкрито кредитну лінію цим позичальником;
- кількість сплатень, які мав виконати позичальник за останній рік;
- загальна кількість позик або кредитних ліній, якими був обтяжений позичальник за весь час;
- кількість позик або кредитних ліній, якими сумарно обтяжений позичальник на поточний момент;
- сумарний обсяг кредитних, позичених коштів, які доступні позичальнику на поточний момент;
- сумарний обсяг кредитних, позичених коштів, які використані позичальником на поточний момент;
- кількість зборів, що припадали на останній рік;
- кількість випадків, які сталися з позичальником та полягали в тому, що він не зміг сплатити за призначений платіж за позикою або кредитом;

- останній випадок, коли позичальник прострочив призначений платіж мінімум на 90 днів, визначений кількістю місяців, яка з того моменту минула;
- кількість випадків, за якими позичальник на момент запиту має прострочені платежі, виражена кількістю таких рахунків (кредитів або позик);
- сумарний обсяг коштів, який необхідно було позичальнику сплатити;
- кількість рахунків у позичальника, які передбачають фіксовані сплати за ними;
- кількість нових позик або кредитів, які були отримані (відкриті) позичальником протягом 2 останніх років;
- кількість місяців, яка минула з моменту, коли позичальник востаннє оформив нову позику або кредит;
- кількість закритих позик або кредитів, які мав позичальник;
- кількість наявних у позичальника активних на момент запиту банківських карт;
- обсяг коштів, що визначає сумарний ліміт, встановлений за всіма банківськими картками;
- кількість кредитних карток, які за весь час мав позичальник;
- кількість кредитних карток, які на момент запиту має позичальник загалом;
- кількість кредитних карток позичальника, які на момент запиту мають на своєму рахунку певний баланс;
- кількість відкритих випадків іпотеки у позичальника;
- відсоток випадків кредитів або позик, за якими позичальник жодного разу не стикався з простроченням платежів, з загальної кількості випадків кредит або позик, які були у цього позичальника за весь час;
- початковий стан позики;
- податкова застава;
- кількість випадків, коли щодо позичальника було оголошено банкрутство;

- призначення запитуваної позичальником позики, визначене відповідною категорією;
- тип запиту на позику, який може бути або сумісним за декількома позичальниками, або індивідуальним;
- сума, яку намагається позичити позичальник;
- строк, на який намагається позичити позичальник цю позику, виражений у місяцях.

Відповідно сума, яку намагається позичити позичальник, та строк, на який намагається позичити позичальник цю позику, можуть змінюватися, при цьому впливаючи на відсоткову ставку позики.

Тоді вихідною ознакою, значення якої для набору позик має бути встановлено, є відсоткова ставка за позиною.

Відповідно задача керування параметрами позики полягає в тому, щоб за наявними значеннями незмінних вхідних ознак позики виконати варіювання значень змінних вхідних ознак в допустимих для цього інтервалах та встановити остаточні значення для них таким чином, щоб відсоткова ставка за позиною була прийнятною для подальшого фінансування її фізичними особами.

2.2 Метод керування параметрами позик

Метод керування параметрами позик дозволяє розв'язати відповідну задачу на основі набору етапів.

На першому етапі має бути підготовлено вибірку даних для прогнозування відсоткової ставки за позиною, включаючи до неї набір ознак, який було представлено у підрозділі 2.1 при формулюванні задачі керування параметрами позик, для кожного екземпляра вибірки, що представляє відповідну позику, встановлюючи відсоткову ставку як вихідну ознаку.

При цьому за назвами посад, які можуть займати позичальники, має бути сформовано набір з 25 посад, що зустрічаються найактивніше. Всі вони

мають бути представлені відповідними значеннями (цілими числами), які задаються відповідну посаду (як категорію). Усім іншим посадам має бути встановлено інше значення (без деталізації таким чином, яка саме це посада).

Окрім того на цьому етапі мають бути прибрані всі екземпляри вибірки, які мають пропущені значення відсоткової ставки. Для пропущених значень за вхідними ознаками мають бути встановлені стандартні значення.

Також значення відсоткової ставки мають бути пронормовані, тобто перетворені на інтервал від 0 до 1.

На наступному етапі має бути створено модель прогнозування відсоткової ставки за позику на основі використання ансамблів дерев рішень, застосовуючи для цього метод eXtreme Gradient Boosting (XGBoost) [6]. Такий варіант ансамблю дерев рішень сприяє створенню спеціалізованих дерев на тих екземплярах вибірки даних для прогнозування, для яких результати прогнозування були гіршими. Це має сприяти в підсумку охопленню різних екземплярів та як результат отриманню більш точних результатів на всій широті позик, які присутні в навчальній вибірці.

Після цього реалізується етап, на якому відбувається навчання створеної моделі на основі опрацьованої попередньо навчальної вибірки даних. Це призводить до створення готової моделі, яка далі може використовуватися для роботи з тими позиками, які потребують встановлення відсоткової ставки.

На наступному етапі, що передбачає повторення для кожної нової позики, має бути сформовано екземпляр даних, для чого має бути визначено значення кожної вхідної ознаки, яка є відомою за позичальником, з перетворенням займаної посади позичальником на роботі у відповідну категорію на основі 25 найбільш популярних. Для ознак, для яких можливе варіювання, а це зокрема сума позики та термін, на який вона надається, має бути встановлено допустимий інтервал значень та крок зміни значень.

На останньому етапі роботи методу керування параметрами позик на основі моделі прогнозування відсоткової ставки реалізується прогнозування

відсоткової ставки для заданих фіксованих значень ознак позики та всіх варіантів змінних вхідних ознак в тих інтервалах, які були встановлені. Тоді на виході має бути отримано значення відсоткової ставки для кожного такого варіанту. Зі всіх отриманих значень має бути обрано той, який є найбільш прийнятним. Тобто набір з суми позики (якщо варіювання допускається), терміну позики та відсоткової ставки. Вибір має здійснюватися, виходячи з комбінації, яка є найкращою з точки зору забезпечення результативності збору коштів для позики та повернення цих коштів кредиторам.

2.3 Висновки за розділом 2

У розділі представлено задачу керування параметрами позик та відповідний метод її розв'язання. Наукова новизна роботи полягає в методі керування параметрами позик, який характеризується використанням дерев рішень на основі створення ансамблів методом XGBoost для прогнозування відсоткової ставки позики за значеннями вхідних ознак, що дозволяє встановлювати варіативні значення параметрів позик таким чином, щоб сприяти результативності процесу збору коштів від фізичних осіб.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір мови та засобів розробки

Розробка вебзастосунку може бути виконана зокрема за рахунок можливостей Python та Django [7] або Ruby та Ruby on Rails [8]. Проведене порівняння (табл. 3.1) вказує, що перший варіант за рахунок Python характеризується кращою підтримкою машинного навчання загалом, хоча для Ruby і присутній зокрема інтерфейс для XGBoost. З іншого боку Django як фреймворк веброзробки має більшу простоту встановлення початкового рішення, кращу читабельність коду, що додатково сприяє створенню ефективного рішення, хоча і Ruby on Rails є достатньо якісним рішенням. Тож за сукупністю факторів було обрано для розробки Python та Django.

Таблиця 3.1 – Вибір засобів розробки програми підтримки надання позик

Критерій	Django	Ruby on Rails
Мова програмування для розробки	Python	Ruby
Фреймворк веброзробки	Django	Ruby on Rails
Складність навчання та підтримки	Менша за рахунок високої читабельності коду	Вища за рахунок великої кількості концепцій
Час на встановлення рішення	Менший	Вищий
Бібліотеки машинного навчання	Більше засобів, краща підтримка, оригінальні версії	Засоби наявні, але нерідко представлені інтерфейсами до Python-рішень, гірша підтримка

3.2 Моделювання роботи з програмним забезпеченням за функціональними вимогами

Користувачі-кредитори є базовими користувачами програми підтримки надання позик, оскільки будь-який початковий користувач без додаткових можливостей може фінансувати позики інших користувачів.

Користувачі-кредитори мають основний потік сценаріїв:

- пошук серед позик;
- перегляд сторінки користувача-позичальника;
- перегляд публікації користувача-позичальника;
- підписування на публікації користувача-позичальника;
- перегляд підписок;
- перегляд сторінки користувача-кредитора;
- внесення даних про користувача-кредитора;
- перегляд матеріалів заявки на позику;
- перегляд заявок на позики користувача;
- фінансування позики;
- перегляд профінансованих користувачем позик;
- перегляд фінансувань позики;
- перегляд отриманих користувачем платежів;
- перегляд платежів за позикою;
- коментування позики;
- перегляд персональних повідомлень;
- коментування публікації;
- створення закладки за позикою;
- рекомендування позики;
- перегляд рекомендацій позик користувача;
- перегляд рекомендацій за позикою;
- перегляд позик у закладках;
- перегляд власних коментарів;

– авторизація (рис. 3.1).

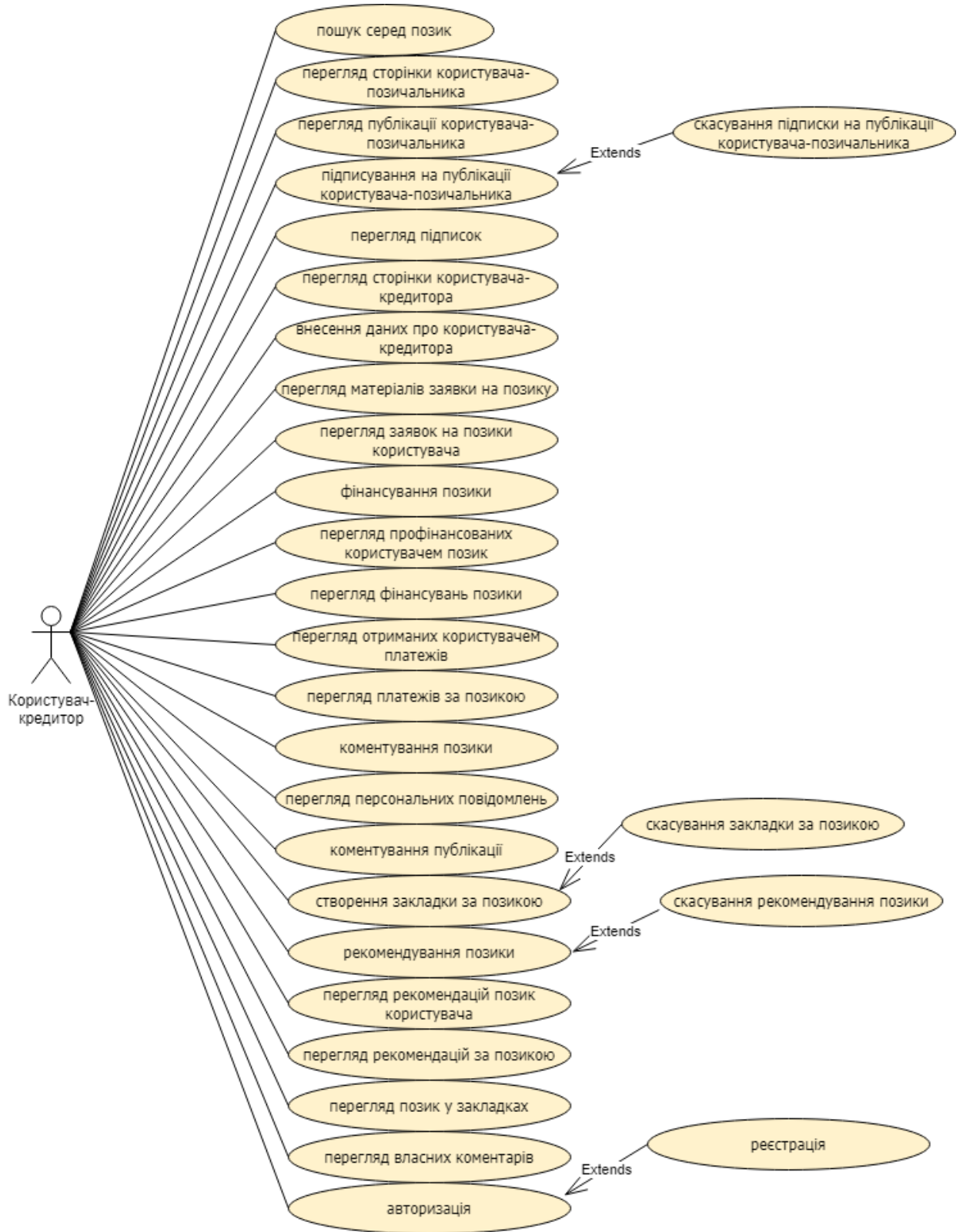


Рисунок 3.1 – Діаграма варіантів використання програми підтримки надання позик користувачами-кредиторами

Альтернативними для цих сценаріїв є такі:

- скасування підписки на публікації користувача-позичальника, що є альтернативним для підписування на публікації користувача-позичальника у випадку, коли підписка на публікації користувача-позичальника вже була забезпечена і не скасована;
- скасування закладки за позикою, що є альтернативним для створення закладки за позикою, коли закладка за позикою вже була створена і не скасована;
- скасування рекомендування позики, що є альтернативним для рекомендування позики;
- реєстрація, що є альтернативною для авторизації, коли акаунта ще у користувач немає жодного.

Користувачі-позичальники є користувачами, які окрім базових функцій висловили бажання використовувати додаткові функції та працювати над власними позиками. Користувачі-позичальники мають основний потік сценаріїв:

- публікація матеріалу;
- внесення загальних даних про користувача-позичальника;
- розміщення заявки на позику;
- внесення власних даних до спільної заявки на позику;
- додавання позичальника для сумісної позики.

Альтернативними для цих сценаріїв є такі:

- скасування заявки на позику, що є альтернативним для розміщення заявки на позику, коли заявка на позику наявна, проте позичальник вирішив відмовитись від неї;
- редагування публікації, що є альтернативним для публікації матеріалу, коли публікація виконана, але потрібні уточнення;
- редагування заявки на позику, що є альтернативним для розміщення заявки на позику, коли заявка на позику наявна, проте потрібні уточнення (рис. 3.2).

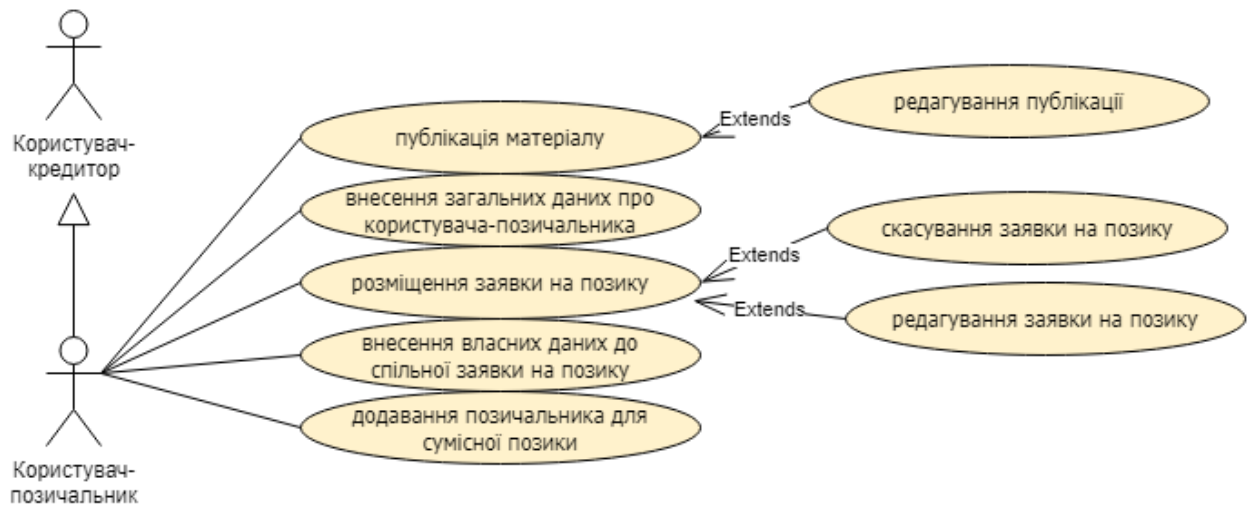


Рисунок 3.2 – Діаграма варіантів використання програми підтримки надання позик користувачами-позичальниками

Представники сервісу надання позик мають основний потік сценаріїв:

- верифікація заявки на позику;
- верифікація документів за позикою;
- авторизація;
- перегляд позик за станом;
- пошук серед позик;
- перегляд сторінки користувача-позичальника;
- перегляд публікації користувача-позичальника;
- перегляд сторінки користувача-кредитора;
- перегляд матеріалів заявки на позику;
- перегляд заявок на позики користувача;
- перегляд профінансованих користувачем позик;
- перегляд фінансувань позики;
- перегляд отриманих користувачем платежів;
- перегляд платежів за позикою;
- перегляд всіх виконаних платежів;
- перегляд всіх виконаних фінансувань (рис. 3.3).

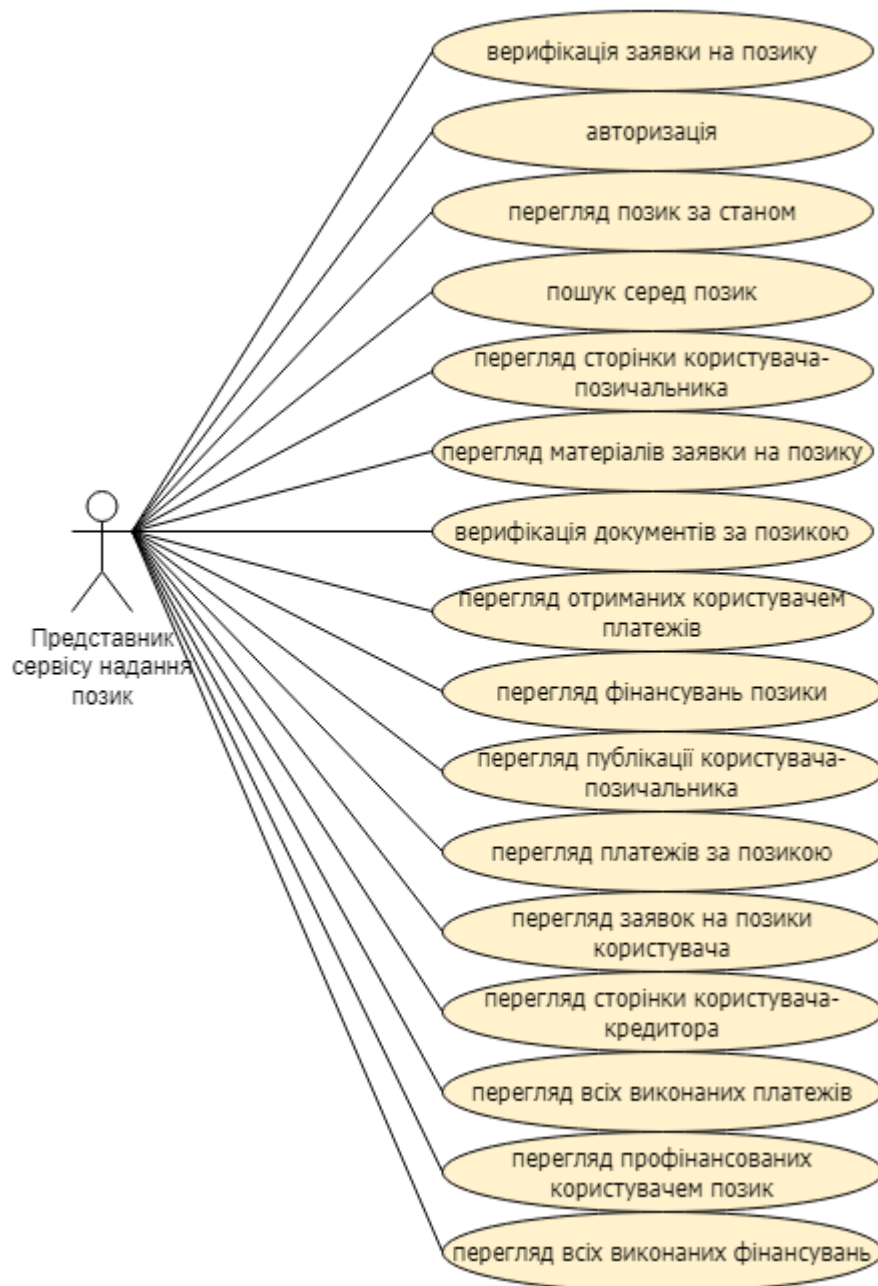


Рисунок 3.3 – Діаграма варіантів використання програми підтримки надання позик представниками сервісу надання позик

3.3 Проєктування структури бази даних

База даних (БД) була конкретизована для подальшого використання своєю структурою, представленою нижче.

Таблиця Region представляє у БД регіони, для яких підтримується надання позик (якщо програма працює в різних країнах, то ця таблиця може

бути організована і фактично використовуватись для зберігання країн) з такими полями:

- id – ідентифікатор регіону, в якому може надаватись позика;
- regname – назва регіону, в якому може надаватись позика.

Таблиця Obligee представляє у БД базові дані акаунтів користувачів, які фактично є кредиторами, з такими полями:

- id – ідентифікатор акаунта користувача-кредитора;
- useraccount – базовий акаунт користувача, який використовується цим користувачем-кредитором;
- personname – ім'я користувача-кредитора (або позичальника, якщо він сформував заявку на позику);
- photofile – файл з фотографією користувача-кредитора;
- information – загальна інформація про себе від користувача-кредитора;
- phone – номер телефону користувача-кредитора.

Таблиця Borrower представляє у БД додаткові дані до акаунта користувача-кредитора, які необхідні для того випадку, коли користувач сформував заявку на позику та почав виконувати в програмі додаткову роль позичальника, з такими полями:

- id – ідентифікатор користувача-позичальника;
- obligeeaccount – базовий акаунт користувача-кредитора, який використовується цим користувачем-позичальником;
- region – регіон, де знаходиться позичальник, визначений зовнішнім ключем до таблиці Region;
- address – адреса проживання позичальника;
- homeownership – наявне право власності на житло у позичальника (орендоване житло, власне житло або використання іпотечної програми);
- earliestcreditline – рік, у який було отримано першу позику або відкрито кредитну лінію цим позичальником.

Таблиця BorrowerEmployee представляє у БД записи про посади, які займають конкретні позичальники, з такими полями:

- id – ідентифікатор посади позичальника;
- borrower – позичальник, який займає цю посаду, визначений зовнішнім ключем до таблиці Borrower;
- employee – посада, яку на своїй роботі займає позичальник, визначена зовнішнім ключем до таблиці Employee;
- fromdate – дата, з якої позичальник займає цю посаду на роботі;
- fixeddate – дата внесення інформації про цю посаду на роботу позичальником.

Таблиця Employee представляє у БД можливі посади, які можуть займати в своїй роботі позичальники, з такими полями:

- id – ідентифікатор можливої посади позичальника;
- emptitle – назва можливої посади позичальника.

Таблиця LoanBorrower представляє у БД відношення позичальників до позик з такими полями:

- id – ідентифікатор відношення позичальника до позики;
- loan – позика, щодо якої визначено ці параметри позичальника, визначена зовнішнім ключем до таблиці Loan;
- borrower – позичальник, який просить про відповідну позику, визначений зовнішнім ключем до таблиці Borrower;
- annualincome – визначений у заявці на позику сумарний річний дохід позичальника;
- verifiedincome – варіант верифікації отриманих даних про дохід від позичальника (перевірено, не перевірено);
- debttoincome – визначене у заявці на позику співвідношення, яке визначено між сумарними боргами позичальника та його сумарними доходами;

– delinq2y – визначена у заявці на позику кількість випадків прострочених платежів за відкритими позиками (кредитами), що трапилась з позичальником за останні 2 роки;

– monthssincelastdelinq – визначена у заявці на позику кількість місяців, яка минула з останнього випадку прострочення платежів за відкритими позиками (кредитами), що трапилась з позичальником;

– inquirieslast12m – визначена у заявці на позику кількість сплачень, які мав виконати позичальник за останній рік;

– totalcreditlines – визначена у заявці на позику загальна кількість позик або кредитних ліній, якими був обтяжений позичальник за весь час (виходячи з наданої ним кредитної історії);

– opencreditlines – визначена у заявці на позику кількість позик або кредитних ліній, якими сумарно обтяжений позичальник на поточний момент (на момент запиту позики);

– totalcreditlimit – визначений у заявці на позику сумарний обсяг кредитних, позичених коштів, які доступні позичальнику на поточний момент (за особливостями обчислення іпотека не входить в це значення, проте кредитні ліміти за банківськими картками входять у максимальному можливому значенні);

– totalcreditutilized – визначений у заявці на позику сумарний обсяг кредитних, позичених коштів, які використані позичальником на поточний момент (за особливостями обчислення іпотека не входить в це значення);

– numcollectionslast12m – кількість зборів, що припадали на останній рік;

– numhistoricalfailedtopay – визначена у заявці на позику кількість випадків, які сталися з позичальником та полягали в тому, що він не зміг сплатити за призначений платіж за позикою або кредитом;

– monthssince90dlate – визначений у заявці на позику останній випадок, коли позичальник прострочив призначений платіж мінімум на 90 днів, визначений кількістю місяців, яка з того моменту минула;

- `currentaccountsdelinq` – визначена у заявці на позику кількість випадків, за якими позичальник на момент запиту має прострочені платежі, виражена кількістю таких рахунків (кредитів або позик);
- `totalcollectionamountever` – визначений у заявці на позику сумарний обсяг коштів, який необхідно було позичальнику сплатити;
- `currentinstallmentaccounts` – визначена у заявці на позику кількість рахунків у позичальника, які передбачають фіксовані сплати за ними;
- `accountsopened24m` – визначена у заявці на позику кількість нових позик або кредитів, які були отримані (відкриті) позичальником протягом 2 останніх років;
- `monthssincelastcreditinquiry` – визначена у заявці на позику кількість місяців, яка минула з моменту, коли позичальник востаннє оформив нову позику або кредит;
- `numsatisfactoryaccounts` – визначена у заявці на позику кількість закритих позик або кредитів, які мав позичальник;
- `numactivedebitaccounts` – визначена у заявці на позику кількість наявних у позичальника активних на момент запиту банківських карт;
- `totaldebitlimit` – визначений у заявці на позику обсяг коштів, що визначає сумарний ліміт, встановлений за всіма банківськими картками позичальника;
- `numtotalccaccounts` – визначена у заявці на позику кількість кредитних карток, які за весь час мав позичальник;
- `numorencssaccounts` – визначена у заявці на позику кількість кредитних карток, які на момент запиту має позичальник загалом;
- `numccsccarryingbalance` – визначена у заявці на позику кількість кредитних карток позичальника, які на момент запиту мають на своєму рахунку певний баланс;
- `nummortaccounts` – визначена у заявці на позику кількість відкритих випадків іпотеки у позичальника;

– `accountneverdelinqpercent` – визначений у заявці на позику відсоток випадків кредитів або позик, за якими позичальник жодного разу не стикався з простроченням платежів, з загальної кількості випадків кредит або позик, які були у цього позичальника за весь час;

– `taxliens` – визначена у заявці на позику податкова застава;

– `publicrecordbankrupt` – визначена у заявці на позику кількість випадків, коли щодо позичальника було оголошено банкрутство;

– `verified` – визначення того, чи пройшли ці дані заявки позичальника верифікацію зі сторони сервісу.

Таблиця `ApproveFiletype` представляє у БД категорії типів документів, які можуть бути надані на підтвердження даних у заявці на позику позичальником, з такими полями:

– `id` – ідентифікатор категорії типів документів, які можуть бути надані на підтвердження даних у заявці на позику позичальником;

– `approvecategory` – назва категорії типів документів, які можуть бути надані на підтвердження даних у заявці на позику позичальником.

Таблиця `LoanBorrowerApprove` представляє у БД файли, які були надані для підтвердження наданої інформації в заявці на позику позичальником, з такими полями:

– `id` – ідентифікатор файлу, який був наданий для підтвердження наданої інформації в заявці на позику позичальником;

– `loanborrower` – позика позичальника, за якою він вніс цей файл для підтвердження наданої інформації в заявці, визначена зовнішнім ключем до таблиці `LoanBorrower`;

– `approvefiletype` – категорія типу документа, який було надано на підтвердження даних у заявці на позику позичальником, визначена зовнішнім ключем до таблиці `ApproveFiletype`;

– `filedata` – дані файлу, який був наданий для підтвердження наданої інформації в заявці на позику позичальником;

– `approvename` – іменування інформації, наданої в файлі документу, який був наданий для підтвердження наданої інформації в заявці на позику позичальником;

– `approvedescription` – коментар позичальника до інформації, наданої в файлі документу, який був наданий для підтвердження наданої інформації в заявці на позику позичальником;

– `loadat` – дата і час завантаження позичальником цього файлу документу;

– `verified` – визначення того, чи пройшов конкретно цей документ верифікацію зі сторони сервісу;

– `verifiedby` – користувач, який виконав верифікацію цього файлу документу зі сторони сервісу, визначений зовнішнім ключем до таблиці `User` БД, яка є стандартною таблицею для авторизації в Django, а тому не виносилась для опису структури БД;

– `verifiedat` – дата і час верифікації сервісом цього файлу документу;

– `personal` – позначка того, що ці дані є прихованими від широкого загалу.

Таблиця `Loan` представляє у БД позики з такими полями:

– `id` – ідентифікатор позики;

– `applicationtype` – тип запиту на позику, який може бути або сумісним за декількома позичальниками, або індивідуальним;

– `initialamount` – сума, що з необхідної одразу наявна у позичальника з тих чи інших джерел;

– `amount` – сума, яку намагається позичити позичальник;

– `amountcollected` – поточна сума, яка зібрана на рахунку для цієї позики;

– `disbursement` – метод виплат за позикою (платіжною карткою або напряму готівкою);

– `status` – стан, в якому позика знаходиться на даний момент;

- term – строк, на який намагається позичити позичальник або позичальники цю позику, виражений у місяцях;
- collectedat – дата і час, коли було зібрано потрібну суму за позикою;
- interestrate – відсоткова ставка, встановлена за позикою;
- interestrateapprove – позначка про те, що ця відсоткова ставка, встановлена за позикою, підтверджена або встановлена самим сервісом: якщо ні, то це означає, що значення є прогнозованим, тобто отриманим на основі запропонованого методу;
- installment – розмір платежу, який кожного місяця має отримувати надавач позики (користувач, який профінансував позику);
- grade – оцінка, яку було присвоєно сервісом позиці, у загальному та деталізованому вигляді разом;
- purpose – призначення запитуваної позичальником позики, визначене відповідною категорією;
- caption – сформована для позики назва, яка відображається у всіх позиціях вебзастосунку, де є обмеження на відображення за кількістю символів;
- fulltitle – сформована для позики повна назва;
- shortdescription – короткий опис ідеї, потреб позики;
- fullpresentation – повний опис ідеї, потреб позики;
- addedat – дата і час подання заявки на позику;
- paidtotal – сумарна кількість коштів, яку було сплачено за цією позикою на цей момент;
- paidprincipal – сумарна кількість коштів, яку потрібно сплатити за цією позикою на цей момент;
- paidlatefees – сплачений обсяг коштів за комісіями, які були висунуті за простроченнями позичальнику.

Таблиця LoanStatus представляє у БД стани, в яких перебували позики, з такими полями:

- id – ідентифікатор стану позики;

- loan – позика, стан якої визначено, визначена зовнішнім ключем до таблиці Loan БД;

- status – стан, який було встановлено для позики;

- setby – представник сервісу, який встановив цей стан для позики, визначений зовнішнім ключем до таблиці User БД, яка є стандартною таблицею для авторизації в Django;

- setat – дата і час, коли цей стан для позики було встановлено представником сервісу.

Таблиця LoanFile представляє у БД файли, які були додані до позики для надання додаткових даних про неї, з такими полями:

- id – ідентифікатор файлу, який був доданий до позики для надання додаткових даних про неї;

- loan – позика, якої стосується цей файл, визначена зовнішнім ключем до таблиці Loan БД;

- filedata – дані самого файлу, який був доданий до позики для надання додаткових даних про неї;

- name – іменування інформації, наданої в файлі, який був доданий до позики для надання додаткових даних про неї;

- description – коментар позичальника до інформації, наданої в файлі, який був доданий до позики для надання додаткових даних про неї;

- loadat – дата і час, коли було завантажено цей файл позичальником до позики;

- personal – позначка того, що ці дані є прихованими від широкого загалу.

Таблиця LoanPhoto представляє у БД фотографії для позики з такими полями:

- id – ідентифікатор фотографії для позики;

- loan – позика, до якої відноситься ця фотографія, визначена зовнішнім ключем до таблиці Loan БД;

- photofile – дані фотографії для позики.

Таблиця LoanPayment представляє у БД платежі, сплачені позичальником за позиною кредитором, з такими полями:

- id – ідентифікатор платежу, сплаченого позичальником за позиною кредитору;
- loaninvestment – надання коштів за позиною кредитором, через яке і відбувається цей платіж з метою повернення коштів позичальником, визначене зовнішнім ключем до таблиці LoanInvestment БД;
- amount – сума, яку було сплачено;
- transaction – дані транзакції у платіжній системі за платежем;
- planneddata – дата, на яку заплановано платіж за позиною від позичальника кредитору;
- fixeddata – дата, коли фактично сплачено платіж за позиною від позичальника кредитору.

Таблиця LoanComment представляє у БД коментарі (або адресовані позичальнику повідомлення) до позики з такими полями:

- id – ідентифікатор коментаря або адресованого позичальнику повідомлення до позики (або навпаки);
- loan – позика, за якою адресовано цей коментар або адресоване повідомлення, визначена зовнішнім ключем до таблиці Loan БД;
- obligee – користувач-кредитор, який створив цей коментар або адресоване позичальнику повідомлення (або якому вони адресовані), визначений зовнішнім ключем до таблиці Obligee БД;
- obligeeauthor – позначка про те, чи це був коментар або повідомлення, адресоване від позичальника до користувача-кредитора як відповідь чи від користувача-кредитора до позичальника;
- comment – текст самого коментаря або повідомлення;
- createdat – дата і час створення цього коментаря або повідомлення;
- personal – позначка про те, що це повідомлення, яке направлене персонально, або це публічний коментар.

Таблиця LoanInvestment представляє у БД платежі для фінансування позики від кредиторів до позичальників з такими полями:

- id – ідентифікатор платежу для фінансування позики від кредитора до позичальника;
- loan – позика, за якою виконано цей платіж, визначена зовнішнім ключем до таблиці Loan БД;
- obligee – користувач-кредитор, який виконав цей платіж для фінансування позики, визначений зовнішнім ключем до таблиці Obligee БД;
- amount – сума, яку було сплачено;
- transaction – дані транзакції у платіжній системі за платежем;
- payedat – дата і час, коли було проведено платіж.

Таблиця BorrowerPost представляє у БД публікацію користувача-позичальника з такими полями:

- id – ідентифікатор публікації користувача-позичальника;
- borrower – користувач-позичальник, який виконав цю публікацію, визначений зовнішнім ключем до таблиці Borrower БД;
- posttitle – назва публікації користувача-позичальника;
- postfulltext – повний текст публікації користувача-позичальника;
- createdat – дата і час, коли користувач-позичальник виконав публікацію.

Таблиця BorrowerPostPhoto представляє у БД фотографії до публікації користувача-позичальника з такими полями:

- id – ідентифікатор фотографії до публікації користувача-позичальника;
- borrowerpost – публікація, до якої відноситься ця фотографія, визначена зовнішнім ключем до таблиці BorrowerPost БД;
- photofile – дані фотографії до публікації користувача-позичальника.

Таблиця BorrowerPostComment представляє у БД коментарі до публікацій користувачів-позичальників з такими полями:

– id – ідентифікатор коментаря до публікації користувача-позичальника;

– borrowerpost – публікація, до якої відноситься цей коментар, визначена зовнішнім ключем до таблиці BorrowerPost БД;

– obligee – користувач-кредитор, який створив цей коментар до публікації, визначений зовнішнім ключем до таблиці Obligee БД;

– obligeeauthor – позначка про те, чи це був коментар до публікації, створений користувачем-кредитором, чи це був коментар у відповідь на такий коментар і створений користувачем-позичальником;

– comment – текст коментаря до публікації користувача-позичальника;

– createdat – дата і час, коли було створено коментар до публікації користувача-позичальника.

Таблиця LoanBookmark представляє у БД закладку на позиці з такими полями:

– id – ідентифікатор створеної користувачем-позичальником закладки на позиці;

– loan – позика, на якій було створено закладку, визначена зовнішнім ключем до таблиці Loan БД;

– obligee – користувач-кредитор, який створив закладку на цій позиці, визначений зовнішнім ключем до таблиці Obligee БД.

Таблиця BorrowerFollow представляє у БД підписки користувачів-кредиторів на користувачів-позичальників з такими полями:

– id – ідентифікатор підписки підписки користувача-кредитора на користувача-позичальника;

– borrower – користувач-позичальник, на якого оформлено підписку, визначений зовнішнім ключем до таблиці Borrower БД;

– obligee – користувач-кредитор, який оформив для себе підписку на користувача-позичальника, визначений зовнішнім ключем до таблиці Obligee БД.

Таблиця LoanRecommend представляє у БД рекомендації, надані позикам користувачами-кредиторами, з такими полями:

- id – ідентифікатор рекомендації, наданої позиці користувачем-кредитором;
- loan – позика, якій надано рекомендацію, визначена зовнішнім ключем до таблиці Loan БД;
- obligee – користувач-кредитор, який надав рекомендацію цій позиці, визначений зовнішнім ключем до таблиці Obligee БД (рис. 3.4).

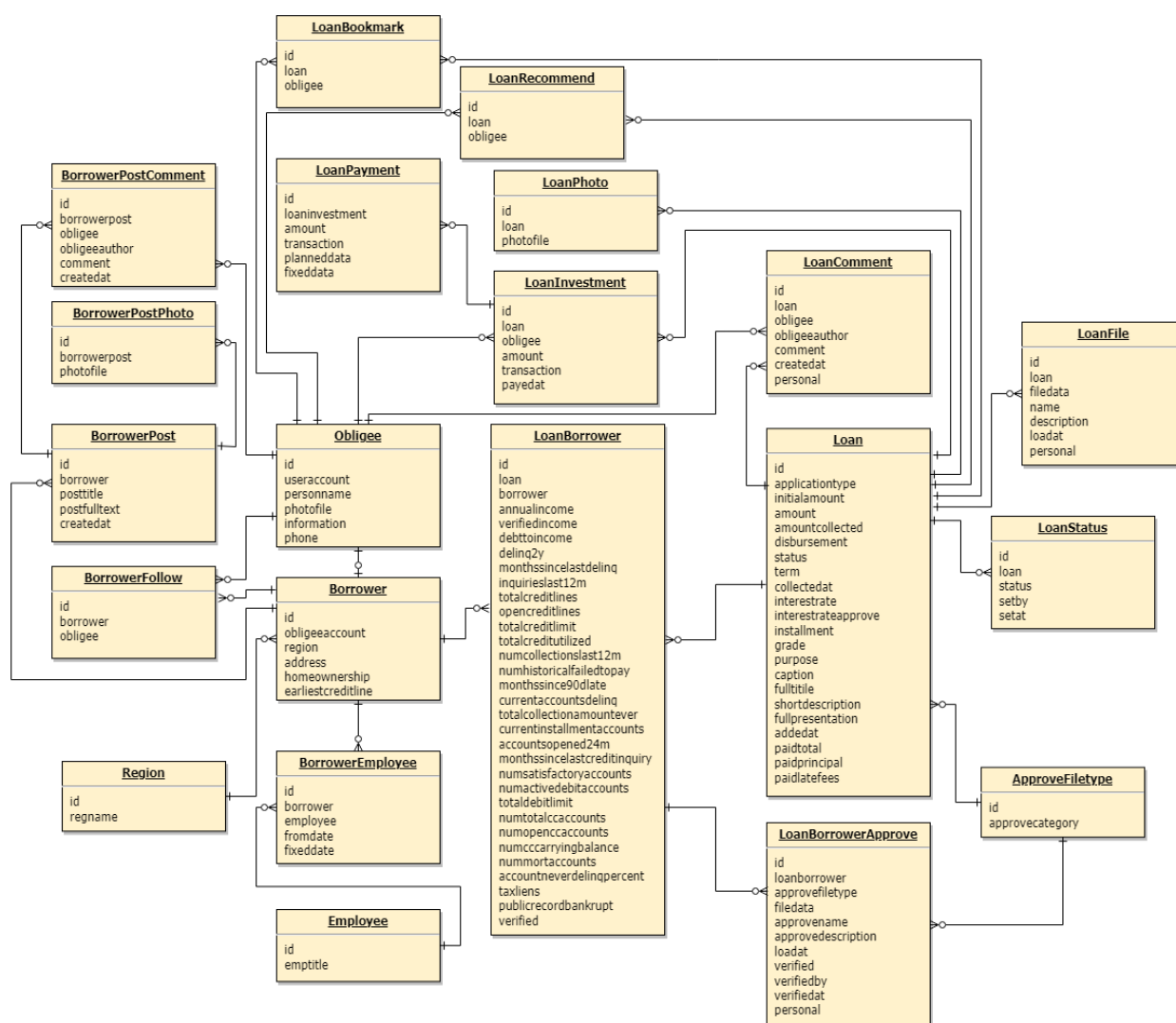


Рисунок 3.4 – Схема БД

3.4 Висновки за розділом 3

Розділ 3 містить вибір зв'язки з мови Python та фреймворку Django для реалізації вебзастосунку, що має підтримувати роботу з даними, машинне навчання, визначення сценаріїв роботи користувачів-орендарів, користувачів-кредиторів, представників сервісу надання позик, проєктування структури БД.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Порядок роботи з позиками

Робота з позиками потребує взаємодії позичальників, сервісу та кредиторів і визначає те, які засоби коли і кому доступні (рис. 4.1).

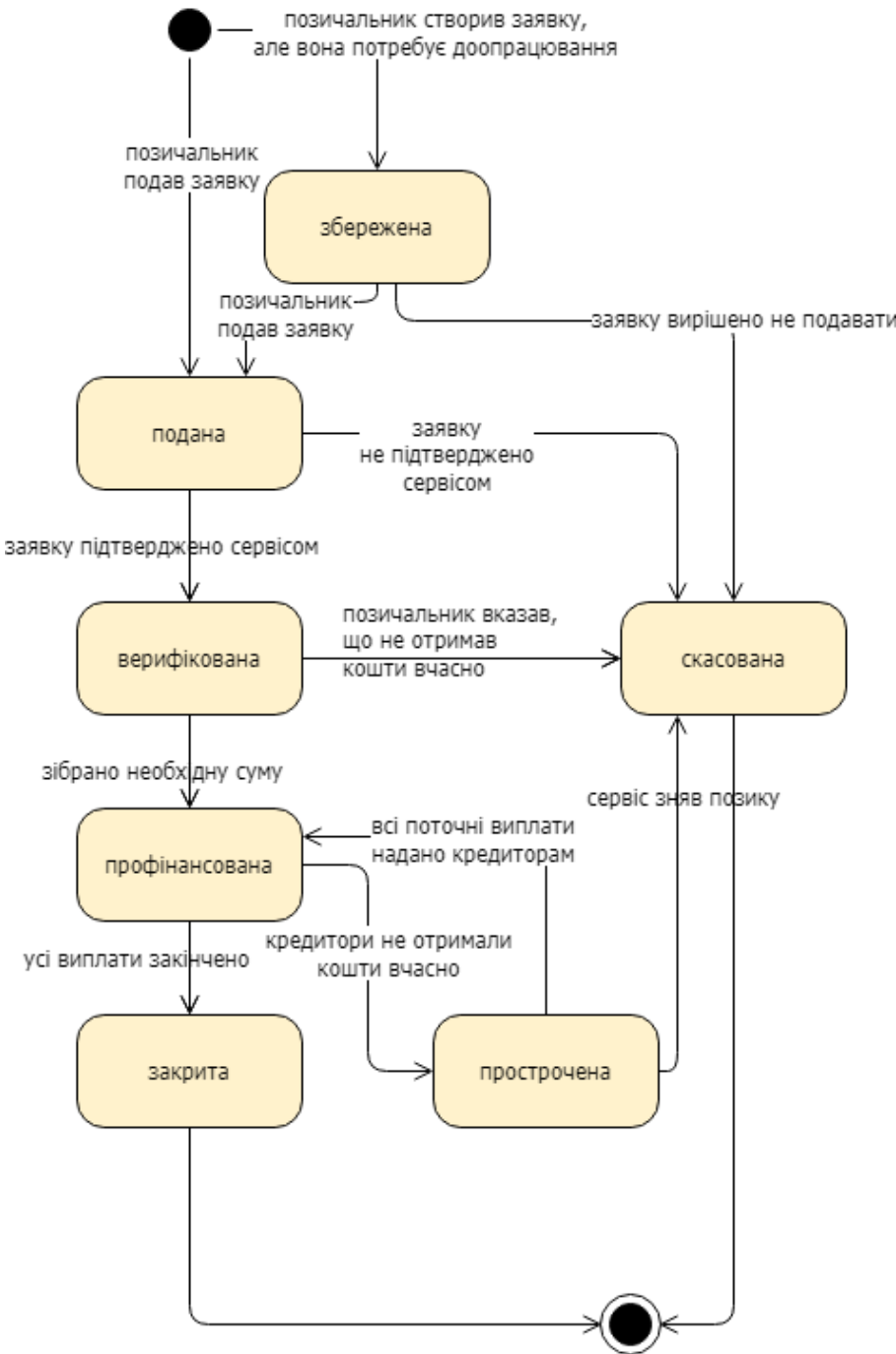


Рисунок 4.1 – Діаграма станів позики

4.2 Рішення з реалізації вебзастосунку

Реалізацію роботи з методом XGBoost виконано за допомогою пакету `xgboost` з імпортування з нього класу `XGBRegressor`. У цьому класі в конструкторі задано кількість дерев у ансамблі (параметр `n_estimators`), далі за допомогою методу `fit` виконано навчання, а за допомогою методу `predict` – прогнозування відсоткової ставки за позикою на основі переданих даних, що цю позику характеризують.

Функція пошуку серед позик має назву `find_loans_view` і реалізує відповідне представлення застосунку. Результати повертаються в шаблон `find_loans.html`, що було створено як і інші шаблони за допомогою шаблону [9]. Самі параметри пошуку задаються у вигляді параметрів GET-запиту, охоплюючи зокрема текст для пошуку (`loancaption`), необхідність виконання пошуку цього тексту тільки в назві або також і в описі (`onlycaption`), конкретну ціль позики (`purpose`) або перелік цілей (`purposes`), мінімальну суму позики (`minamount`), максимальну (`maxamount`), мінімальну початково наявну суму (`mininamount`), максимальну (`maxinamount`), мінімальну зібрану на поточний момент суму позики (`mincamount`), максимальну (`maxcamount`), спільні позики чи тільки індивідуальні (`joint`), метод повернення позик (`dis`), стан позики (`status`) або можливі стани позики (`statuses`), строк фінансування (`term`) або строки фінансування (`terms`), мінімальну відсоткову ставку за позикою (`minirate`), максимальну (`maxirate`), мінімальний відсоток зібраної суми зі всієї потрібної суми (`minrate`), максимальний (`maxrate`), мінімальну оцінку (`mingrade`), максимальну (`maxgrade`), перелік посад, на яких має працювати позичальник (`employees`), напрямок сортування (`ordering`). Результати розподіляються по 20 на сторінку.

Функція перегляду сторінки користувача-позичальника має назву `borrower_page_view` і реалізує відповідне представлення застосунку. Користувач-позичальник задається своїм логіном користувача у параметрі `borrower_user` функції, а якщо це значення не задано, то здійснюється спроба

отримати ці дані з авторизації. За користувачем-позичальником витягаються дані про нього, його заявки на позики, вибираючи з них 10 останніх за часом подання та 10 з найбільшою кількістю рекомендацій, його публікації. Також визначається за позиками, чи були вони поточним користувачем внесені до закладок, чи були за ними надані ним рекомендації. Усі ці дані передаються шаблону `borrower.html`.

Функція перегляду публікації користувача-позичальника має назву `borrower_post_view` і реалізує відповідне представлення застосунку. Номер публікації користувача-позичальника задається параметром `postid`. Окрім публікації додаються також коментарі.

Функція перегляду підписок має назву `followed_borrowers_view` і реалізує відповідне представлення застосунку. Параметр `loanorpost` задає те, що саме має повертатися від користувачів-позичальників, на яких підписався кредитор: заявки на позики чи публікації.

Функція перегляду матеріалів заявки на позику має назву `loan_view` і реалізує відповідне представлення застосунку. Номер заявки на позику задається параметром `loanid`. Якщо запит зроблений власником заявки на позику або представником сервісу, то надаються повністю всі дані, а якщо іншим користувачем, то приховані дані (зокрема додані для підтвердження матеріали) не надаються. Надаються дані позики, фінансування, платежі, позичальники.

Функція перегляду заявок на позики користувача має назву `borrower_loans_view` і реалізує відповідне представлення застосунку. Параметр `ordering` задає сортування позик, `borrower_user` – логін позичальника. Виконується розподіл на сторінки по 20 позик на кожній, що надається в шаблон `borrower_loans.html`.

Функція перегляду фінансувань позики має назву `loan_investments_view` і реалізує відповідне представлення застосунку. Номер заявки на позику задається параметром `loanid`, порядок сортування – параметром `ordering`.

Функція перегляду платежів за позикою має назву `loan_payments_view` і реалізує відповідне представлення застосунку. Номер заявки на позику задається параметром `loanid`, порядок сортування – параметром `ordering`.

Функція перегляду профінансованих користувачем позик має назву `obligee_investments_view` і реалізує відповідне представлення застосунку. Логін користувача-кредитора задається параметром `obligee_user`, напрям сортування платежів – параметром `ordering`.

Функція перегляду отриманих користувачем платежів має назву `obligee_payments_view` і реалізує відповідне представлення застосунку. Логін користувача-кредитора задається параметром `obligee_user`, напрям сортування платежів – параметром `ordering`.

Функція перегляду сторінки користувача-кредитора має назву `obligee_page_view` і реалізує відповідне представлення застосунку. Логін користувача-кредитора задається параметром `obligee_user`.

Функція перегляду позик у закладках має назву `bookmarked_loans_view` і реалізує відповідне представлення застосунку.

Функція перегляду персональних повідомлень має назву `personal_loan_comments_view` і реалізує відповідне представлення застосунку.

Функція перегляду власних коментарів має назву `own_loan_comments_view` і реалізує відповідне представлення застосунку, надаючи коментарі як за позиками, так і за публікаціями.

Функція підписування на публікації користувача-позичальника має назву `borrower_follow_view` і реалізує відповідне представлення застосунку. Якщо підписка вже є цього користувача-кредитора на цього користувача-позичальника, вона скасовується, а якщо немає, то вноситься в БД.

Функція внесення даних про користувача-кредитора має назву `obligee_edit_view` і реалізує відповідне представлення застосунку на основі класу форми `ObligeeForm` з параметром `obligee_user`, що задає логін користувача-кредитора.

Функція створення закладки за позикою має назву `loan_bookmark_add_view` і реалізує відповідне представлення застосунку. Номер позики задається параметром `loadid`. Якщо закладка для цього користувача на цій позиції створена, то вона вилучається.

Функція рекомендування позики має назву `loan_recommend_add_view` і реалізує відповідне представлення застосунку. Номер позики задається параметром `loadid`. Якщо рекомендація для цього користувача на цій позиції створена, то вона вилучається.

Функція перегляду рекомендацій позик користувача має назву `obligee_recommended_loans_view` і реалізує відповідне представлення застосунку.

Функція перегляду рекомендацій за позикою має назву `loan_recommends_view` і реалізує відповідне представлення застосунку. Номер заявки на позику задається параметром `loanid`.

Функція публікації матеріалу має назву `borrower_post_add_view` і реалізує відповідне представлення застосунку на основі класу форми `BorrowerPostForm`.

Функція внесення загальних даних про користувача-позичальника має назву `borrower_add_view` і реалізує відповідне представлення застосунку на основі класу форми `BorrowerForm`.

Функція розміщення заявки на позику має назву `loan_add_view` і реалізує відповідне представлення застосунку на основі класу форми `LoanForm`.

Функція внесення власних даних до спільної заявки на позику має назву `loan_borrower_add_view` і реалізує відповідне представлення застосунку на основі класу форми `LoanBorrowerForm` за позикою з номером `loanid`, визначеним параметром функції, при цьому користувач раніше мав бути доданий до цієї позики її автором.

Функція додавання позичальника для сумісної позики має назву `loan_borrower_create_view` і реалізує відповідне представлення застосунку. Номер позики задається параметром `loanid`.

Функція редагування заявки на позику має назву `borrower_post_edit_view` і реалізує відповідне представлення застосунку на основі класу форми `BorrowerPostForm` для позики з номером `loanid`, визначеним параметром функції.

Функція скасування заявки на позику має назву і реалізує відповідне представлення застосунку `loan_cancel_view`. Номер заявки на позику задається параметром `loanid`. Скасування відбувається тільки тоді, коли заявка на позику належить користувачу або коли користувач представляє сервіс надання позик.

Функція верифікації заявки на позику має назву `loan_verify_view` і реалізує відповідне представлення застосунку для позики з номером `loanid` тільки для представника сервісу надання позик.

Функція перегляду позик за станом має назву `state_loans_view` і реалізує відповідне представлення застосунку. Параметр `loanstate` встановлює конкретний потрібний стан позик, `ordering` – конкретний напрямок сортування, `term` – строк позики, `purpose` – ціль надання позики, `amount_min` – мінімальну суму, запитану в позиці, а `amount_max` – максимальну. Виконується перевірка, що користувач авторизувався як представник сервісу, інакше дані не передаються в шаблон `state_loans.html`. Інші функції створені подібно.

Клас `Region` створений для реалізації моделі регіонів, для яких підтримується надання позик. Поле класу `regname` визначає назву регіону, в якому може надаватись позика, використовуючи тип `CharField`.

Клас `Obligee` створений для реалізації моделі акаунтів користувачів, які фактично є кредиторами.

Поле `useraccount` визначає базовий акаунт користувача, який використовується цим користувачем-кредитором, використовуючи тип `OneToOneField` та зв'язок з моделлю `User`.

Поле класу `personname` визначає ім'я користувача-кредитора (або позичальника, якщо він сформував заявку на позику), використовуючи тип `CharField`.

Поле `photofile` визначає файл з фотографією користувача-кредитора, використовуючи тип `ImageField`.

Поле `information` визначає загальну інформацію про себе від користувача-кредитора, використовуючи тип `TextField`.

Поле `phone` визначає номер телефону користувача-кредитора, використовуючи тип `CharField`.

Поле `loanbookmarks` визначає закладки на позики, створені користувачем-кредитором, використовуючи тип `ManyToManyField` та зв'язок з моделлю `Loan`.

Клас `Borrower` створений для реалізації моделі акаунта користувача-кредитора.

Поле `obligeeaccount` визначає базовий акаунт користувача-кредитора, який використовується цим користувачем-позичальником, використовуючи тип `OneToOneField` та зв'язок з моделлю `Obligee`.

Поле `region` визначає регіон, де знаходиться позичальник, використовуючи тип `ForeignKey` та зв'язок з моделлю `Region`.

Поле `address` визначає адресу проживання позичальника, використовуючи тип `CharField`.

Поле `homeownership` визначає наявне право власності на житло у позичальника (орендоване житло, власне житло або використання іпотечної програми), використовуючи тип `CharField` для вибору з варіантів.

Поле `earliestcreditline` визначає рік, у який було отримано першу позику або відкрито кредитну лінію цим позичальником, використовуючи тип `PositiveIntegerField`.

Поле `followers` визначає підписки користувачів-кредиторів на цього користувача-позичальника, використовуючи тип `ManyToManyField` та зв'язок з моделлю `Obligee`.

Клас `BorrowerEmployee` створений для реалізації моделі записів про посади, які займають конкретні позичальники.

Поле `borrower` визначає позичальника, який займає цю посаду, використовуючи тип `ForeignKey` та зв'язок з моделлю `Borrower`.

Поле `employee` визначає посаду, яку на своїй роботі займає позичальник, використовуючи тип `ForeignKey` та зв'язок з моделлю `Employee`.

Поле `fromdate` визначає дату, з якої позичальник займає цю посаду на роботі, використовуючи тип `DateField`.

Поле `fixeddate` визначає дату внесення інформації про цю посаду на роботі позичальником, використовуючи тип `DateField`.

Клас `Employee` створений для реалізації моделі можливих посад, які можуть займати в своїй роботі позичальники. Поле `emptitle` визначає назву можливої посади позичальника, використовуючи тип `CharField`.

Клас `LoanBorrower` створений для реалізації моделі відношення позичальників до позик.

Поле `loanborrower` визначає позику, щодо якої визначено ці параметри позичальника, використовуючи тип `ForeignKey` та зв'язок з моделлю `Loan`.

Поле `borrower` реалізує позичальника, який просить про відповідну позику, використовуючи тип `ForeignKey` та зв'язок з моделлю `Borrower`.

Поле `annualincome` реалізує визначений у заявці на позику сумарний річний дохід позичальника, використовуючи тип `DecimalField`.

Поле `verifiedincome` реалізує варіант верифікації отриманих даних про дохід від позичальника, використовуючи тип `CharField` для вибору з варіантів.

Поле `debttoincome` реалізує визначене у заявці на позику співвідношення, яке визначено між сумарними боргами позичальника та його сумарними доходами, використовуючи тип `DecimalField`.

Поле `delinq2y` реалізує визначену у заявці на позику кількість випадків прострочених платежів за відкритими позиками (кредитами), що трапилась з позичальником за останні 2 роки, використовуючи тип `PositiveSmallIntegerField`.

Поле `monthssincelastdelinq` реалізує визначену у заявці на позику кількість місяців, яка минула з останнього випадку прострочення платежів за відкритими позиками (кредитами), що трапилась з позичальником, використовуючи тип `PositiveSmallIntegerField`.

Поле `inquirieslast12m` реалізує визначену у заявці на позику кількість сплатень, які мав виконати позичальник за останній рік, використовуючи тип `PositiveSmallIntegerField`.

Поле `totalcreditlines` реалізує визначену у заявці на позику загальну кількість позик або кредитних ліній, якими був обтяжений позичальник за весь час (виходячи з наданої ним кредитної історії), використовуючи тип `PositiveSmallIntegerField`.

Поле `opencreditlines` реалізує визначену у заявці на позику кількість позик або кредитних ліній, якими сумарно обтяжений позичальник на поточний момент (на момент запиту позики), використовуючи тип `PositiveSmallIntegerField`.

Поле `totalcreditlimit` реалізує визначений у заявці на позику сумарний обсяг кредитних, позичених коштів, які доступні позичальнику на поточний момент, використовуючи тип `DecimalField`.

Поле `totalcreditutilized` реалізує визначений у заявці на позику сумарний обсяг кредитних, позичених коштів, які використані позичальником на поточний момент, використовуючи тип `DecimalField`.

Поле `numcollectionslast12m` реалізує кількість зборів, що припадали на останній рік, використовуючи тип `PositiveSmallIntegerField`.

Поле `numhistoricalfailedtopay` реалізує визначену у заявці на позику кількість випадків, які сталися з позичальником та полягали в тому, що він не зміг сплатити за призначений платіж за позикою або кредитом, використовуючи тип `PositiveSmallIntegerField`.

Поле `monthssince90dlate` реалізує визначений у заявці на позику останній випадок, коли позичальник прострочив призначений платіж мінімум

на 90 днів, визначений кількістю місяців, яка з того моменту минула, використовуючи тип `PositiveSmallIntegerField`.

Поле `currentaccountsdelinq` реалізує визначену у заявці на позику кількість випадків, за якими позичальник на момент запиту має прострочені платежі, виражена кількістю таких рахунків (кредитів або позик), використовуючи тип `PositiveSmallIntegerField`.

Поле `totalcollectionamountever` реалізує визначений у заявці на позику сумарний обсяг коштів, який необхідно було позичальнику сплатити, використовуючи тип `DecimalField`.

Поле `currentinstallmentaccounts` реалізує визначену у заявці на позику кількість рахунків у позичальника, які передбачають фіксовані сплати за ними, використовуючи тип `PositiveSmallIntegerField`.

Поле `accountsopened24m` реалізує визначену у заявці на позику кількість нових позик або кредитів, які були отримані (відкриті) позичальником протягом 2 останніх років, використовуючи тип `PositiveSmallIntegerField`.

Поле `monthssincelastcreditinquiry` реалізує визначену у заявці на позику кількість місяців, яка минула з моменту, коли позичальник востаннє оформив нову позику або кредит, використовуючи тип `PositiveSmallIntegerField`.

Поле `numsatisfactoryaccounts` реалізує визначену у заявці на позику кількість закритих позик або кредитів, які мав позичальник, використовуючи тип `PositiveSmallIntegerField`.

Поле `numactivedebitaccounts` реалізує визначену у заявці на позику кількість наявних у позичальника активних на момент запиту банківських карт, використовуючи тип `PositiveSmallIntegerField`.

Поле `totaldebitlimit` реалізує визначений у заявці на позику обсяг коштів, що визначає сумарний ліміт, встановлений за всіма банківськими картками позичальника, використовуючи тип `DecimalField`.

Поле numtotalccaccounts реалізує визначену у заявці на позику кількість кредитних карток, які за весь час мав позичальник, використовуючи тип PositiveSmallIntegerField.

Поле numopencsaccounts реалізує визначену у заявці на позику кількість кредитних карток, які на момент запиту має позичальник загалом, використовуючи тип PositiveSmallIntegerField.

Поле numccsarryingbalance реалізує визначену у заявці на позику кількість кредитних карток позичальника, які на момент запиту мають на своєму рахунку певний баланс, використовуючи тип PositiveSmallIntegerField.

Поле nummortaccounts реалізує визначену у заявці на позику кількість відкритих випадків іпотеки у позичальника, використовуючи тип PositiveSmallIntegerField.

Поле accountneverdelinqpercent реалізує визначений у заявці на позику відсоток випадків кредитів або позик, за якими позичальник жодного разу не стикався з простроченням платежів, з загальної кількості випадків кредит або позик, які були у цього позичальника за весь час, використовуючи тип DecimalField.

Поле taxliens реалізує визначену у заявці на позику податкову заставу, використовуючи тип DecimalField.

Поле publicrecordbankrupt реалізує визначену у заявці на позику кількість випадків, коли щодо позичальника було оголошено банкрутство, використовуючи тип PositiveSmallIntegerField.

Поле verified реалізує визначення того, чи пройшли ці дані заявки позичальника верифікацію зі сторони сервісу, використовуючи тип BooleanField.

Клас ApproveFiletype створений для реалізації моделі категорії типів документів, які можуть бути надані на підтвердження даних у заявці на позику позичальником. Поле approvecategory визначає назву категорії типів документів, які можуть бути надані на підтвердження даних у заявці на позику позичальником, використовуючи тип CharField.

Клас `LoanBorrowerApprove` створений для реалізації моделі файлів, які були надані для підтвердження наданої інформації в заявці на позику позичальником.

Поле `loanborrower` визначає позику позичальника, за якою він вніс цей файл для підтвердження наданої інформації в заявці, використовуючи тип `ForeignKey` та зв'язок з моделлю `LoanBorrower`.

Поле `approvefiletype` визначає категорію типу документа, який було надано на підтвердження даних у заявці на позику позичальником, використовуючи тип `ForeignKey` та зв'язок з моделлю `ApproveFiletype`.

Поле `filedata` визначає дані файлу, який був наданий для підтвердження наданої інформації в заявці на позику позичальником, використовуючи тип `FileField`.

Поле `approvename` визначає іменування інформації, наданої в файлі документу, який був наданий для підтвердження наданої інформації в заявці на позику позичальником, використовуючи тип `CharField`.

Поле `approvedescription` визначає коментар позичальника до інформації, наданої в файлі документу, який був наданий для підтвердження наданої інформації в заявці на позику позичальником, використовуючи тип `TextField`.

Поле `loadat` визначає дату і час завантаження позичальником цього файлу документу, використовуючи тип `DateTimeField`.

Поле `verified` реалізує визначення того, чи пройшов конкретно цей документ верифікацію зі сторони сервісу, використовуючи тип `BooleanField`.

Поле `verifiedby` визначає користувача, який виконав верифікацію цього файлу документу зі сторони сервісу, використовуючи тип `ForeignKey` та зв'язок з моделлю `User`.

Поле `verifiedat` визначає дату і час верифікації сервісом цього файлу документу, використовуючи тип `DateTimeField`.

Поле `personal` визначає позначку того, що ці дані є прихованими від широкого загалу, використовуючи тип `BooleanField`.

Клас Loan створений для реалізації моделі позики.

Поле `applicationtype` визначає тип запиту на позику, використовуючи тип `BooleanField`.

Поле `initialamount` визначає суму, що з необхідної одразу наявна у позичальника з тих чи інших джерел, використовуючи тип `DecimalField`.

Поле `amount` визначає суму, яку намагається позичити позичальник, використовуючи тип `DecimalField`.

Поле `amountcollected` визначає поточну суму, яка зібрана на рахунок для цієї позики, використовуючи тип `DecimalField`.

Поле `disbursement` визначає метод виплат за позикою, використовуючи тип `BooleanField`.

Поле `status` визначає стан, в якому позика знаходиться на даний момент, використовуючи тип `CharField` для вибору з варіантів.

Поле `term` визначає строк, на який намагається позичити позичальник або позичальники цю позику, виражений у місяцях, використовуючи тип `PositiveSmallIntegerField`.

Поле `collectedat` визначає дату і час, коли було зібрано потрібну суму за позикою, використовуючи тип `DateTimeField`.

Поле `interestrate` визначає відсоткову ставку, встановлена за позикою, використовуючи тип `DecimalField`.

Поле `interestrateapprove` визначає позначку про те, що ця відсоткова ставка, встановлена за позикою, підтверджена або встановлена самим сервісом, використовуючи тип `BooleanField`.

Поле `installment` визначає розмір платежу, який кожного місяця має отримувати надавач позики, використовуючи тип `DecimalField`.

Поле `grade` визначає оцінку, яку було присвоєно сервісом позиці, у загальному та деталізованому вигляді разом, використовуючи тип `CharField`.

Поле `purpose` визначає призначення запитуваної позичальником позики, визначене відповідною категорією, використовуючи тип `CharField` для вибору з варіантів.

Поле `caption` визначає сформовану для позики назву, яка відображається у всіх позиціях вебзастосунку, де є обмеження на відображення за кількістю символів, використовуючи тип `CharField`.

Поле `fulltitle` визначає сформовану для позики повну назву, використовуючи тип `CharField`.

Поле `shortdescription` визначає короткий опис ідеї, потреб позики, використовуючи тип `CharField`.

Поле `fullpresentation` визначає повний опис ідеї, потреб позики, використовуючи тип `TextField`.

Поле `addedat` визначає дату і час подання заявки на позику, використовуючи тип `DateTimeField`.

Поле `paidtotal` визначає сумарну кількість коштів, яку було сплачено за цією позикою на цей момент, використовуючи тип `DecimalField`.

Поле `paidprincipal` визначає сумарну кількість коштів, яку потрібно сплатити за цією позикою на цей момент, використовуючи тип `DecimalField`.

Поле `paidlatefees` визначає сплачений обсяг коштів за комісіями, які були висунуті за простроченнями позичальнику, використовуючи тип `DecimalField`.

Поле `recommends` визначає рекомендації, надані цій позиції користувачами-кредиторами, використовуючи тип `ManyToManyField` та зв'язок з моделлю `Obligee`.

Клас `LoanStatus` створений для реалізації моделі станів, в яких перебували позики.

Поле `loan` визначає позику, стан якої визначено, використовуючи тип `ForeignKey` та зв'язок з моделлю `Loan`.

Поле `status` визначає стан, який було встановлено для позики, використовуючи тип `CharField` для вибору з варіантів.

Поле `setby` визначає представника сервісу, який встановив цей стан для позики, використовуючи тип `ForeignKey` та зв'язок з моделлю `User`.

Поле `setat` визначає дату і час, коли цей стан для позики було встановлено представником сервісу, використовуючи тип `DateTimeField`.

Клас `LoanFile` створений для реалізації моделі файлів, які були додані до позики для надання додаткових даних про неї.

Поле `loan` визначає позику, якої стосується цей файл, використовуючи тип `ForeignKey` та зв'язок з моделлю `Loan`.

Поле `filedata` визначає дані самого файлу, який був доданий до позики для надання додаткових даних про неї, використовуючи тип `FileField`.

Поле `name` визначає іменування інформації, наданої в файлі, який був доданий до позики для надання додаткових даних про неї, використовуючи тип `CharField`.

Поле `description` визначає коментар позичальника до інформації, наданої в файлі, який був доданий до позики для надання додаткових даних про неї, використовуючи тип `CharField`.

Поле `loadat` визначає дату і час, коли було завантажено цей файл позичальником до позики, використовуючи тип `DateTimeField`.

Поле `personal` визначає позначку того, що ці дані є прихованими від широкого загалу, використовуючи тип `BooleanField`.

Клас `LoanPhoto` створений для реалізації моделі фотографії для позики.

Поле `loan` визначає позику, до якої відноситься ця фотографія, використовуючи тип `ForeignKey` та зв'язок з моделлю `Loan`.

Поле `photofile` визначає дані фотографії для позики, використовуючи тип `ImageField`.

Клас `LoanPayment` створений для реалізації моделі платежів, сплачених позичальником за позикою кредиторам.

Поле `loaninvestment` визначає надання коштів за позикою кредитором, через яке і відбувається цей платіж з метою повернення коштів позичальником, використовуючи тип `ForeignKey` та зв'язок з моделлю `LoanInvestment`.

Поле `amount` визначає суму, яку було сплачено, використовуючи тип `DecimalField`.

Поле `transaction` визначає дані транзакції у платіжній системі за платежем, використовуючи тип `CharField`.

Поле `planneddata` визначає дату, на яку заплановано платіж за позикою від позичальника кредитору, використовуючи тип `DateField`.

Поле `fixeddata` визначає дату, коли фактично сплачено платіж за позикою від позичальника кредитору, використовуючи тип `DateField`.

Клас `LoanComment` створений для реалізації моделі коментарів (або адресованих позичальнику повідомлень) до позики.

Поле `loan` визначає позику, за якою адресовано цей коментар або адресоване повідомлення, використовуючи тип `ForeignKey` та зв'язок з моделлю `Loan`.

Поле `obligee` визначає користувача-кредитора, який створив цей коментар або адресоване позичальнику повідомлення (або якому вони адресовані), використовуючи тип `ForeignKey` та зв'язок з моделлю `Obligee`.

Поле `obligeeauthor` визначає позначку про те, чи це був коментар або повідомлення, адресоване від позичальника до користувача-кредитора як відповідь чи від користувача-кредитора до позичальника, використовуючи тип `BooleanField`.

Поле `comment` визначає текст самого коментаря або повідомлення, використовуючи тип `CharField`.

Поле `createdat` визначає дату і час створення цього коментаря або повідомлення, використовуючи тип `DateTimeField`.

Поле `personal` визначає позначку про те, що це повідомлення, яке направлене персонально, або це публічний коментар, використовуючи тип `BooleanField`.

Клас `LoanInvestment` створений для реалізації моделі платежів для фінансування позики від кредиторів до позичальників.

Поле `loan` визначає позику, за якою виконано цей платіж, використовуючи тип `ForeignKey` та зв'язок з моделлю `Loan`.

Поле `obligee` визначає користувача-кредитор, який виконав цей платіж для фінансування позики, використовуючи тип `ForeignKey` та зв'язок з моделлю `Obligee`.

Поле `amount` визначає суму, яку було сплачено, використовуючи тип `DecimalField`.

Поле `transaction` визначає дані транзакції у платіжній системі за платежем, використовуючи тип `CharField`.

Поле `payedat` визначає дату і час, коли було проведено платіж, використовуючи тип `DateTimeField`.

Клас `BorrowerPost` створений для реалізації моделі публікації користувача-позичальника.

Поле `borrower` визначає користувача-позичальника, використовуючи тип `ForeignKey` та зв'язок з моделлю `Borrower`.

Поле `posttitle` визначає назву публікації користувача-позичальника, використовуючи тип `CharField`.

Поле `postfulltext` визначає повний текст публікації користувача-позичальника, використовуючи тип `TextField`.

Поле `createdat` визначає дату і час, коли користувач-позичальник виконав публікацію, використовуючи тип `DateTimeField`.

Клас `BorrowerPostPhoto` створений для реалізації моделі фотографії до публікації користувача-позичальника.

Поле `borrowerpost` визначає публікацію, до якої відноситься ця фотографія, використовуючи тип `ForeignKey` та зв'язок з моделлю `BorrowerPost`.

Поле `photofile` визначає дані фотографії до публікації користувача-позичальника, використовуючи тип `ImageField`.

Клас `BorrowerPostComment` створений для реалізації моделі коментарів до публікацій користувачів-позичальників.

Поле `borrowerpost` визначає публікацію, до якої відноситься цей коментар, використовуючи тип `ForeignKey` та зв'язок з моделлю `BorrowerPost`.

Поле `obligee` визначає користувача-кредитора, який створив цей коментар до публікації, використовуючи тип `ForeignKey` та зв'язок з моделлю `Obligee`.

Поле `obligeeauthor` визначає позначку про те, чи це був коментар до публікації, створений користувачем-кредитором, чи це був коментар у відповідь на такий коментар і створений користувачем-позичальником, використовуючи тип `BooleanField`.

Поле `comment` визначає текст коментаря до публікації користувача-позичальника, використовуючи тип `CharField`.

Поле `createdat` визначає дату і час, коли було створено коментар до публікації користувача-позичальника, використовуючи тип `DateTimeField`.

4.3 Висновки за розділом 4

Програмне забезпечення підтримки надання позик повноцінно реалізовано як вебзастосунок на основі фреймворку Django, що і визначило необхідність створення представлень, шаблонів вебсторінок для кожної функціональної можливості, моделей для роботи з даними, які й були реалізовані та описані в цьому розділі.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис програми

Програма призначена для підтримки надання позик між фізичними особами, коли фінансові ресурси для позик надаються не банківськими установами, а безпосередньо фізичними особами, які згодні вкласти їх на фінансування певної діяльності інших осіб, отримуючи в підсумку відсотки за надані фінанси.

Програма має використовуватися позичальниками, кредиторами та представниками сервісу, які виконують керування позиками, визначаючи ті позики, які мають бути заблоковані від доступу до позичальників, а для всіх інших позик встановлюючи відсоткові ставки та надаючи їх таким чином до уваги потенційним позичальникам. Для взаємодії та сприяння розумінню позичальників кредиторами першим надається можливість створювати власні публікації, таким чином знайомлячи кредиторів з інтересами, мисленням, позиціонуванням позичальників.

5.2 Експлуатація програми

При формуванні запиту до вебзастосунку спочатку будь-який користувач розглядається як користувач-кредитор, відповідно незалежно від того, чи він авторизувався, йому доступне меню з розділами Пошук для виконання пошуку серед позик, підписки для перегляду заявок на позики та публікацій тих користувачів-позичальників, на яких підписався авторизований користувач, Платежі для перегляду тих платежів, які користувач-кредитор отримав (за допомогою окремої кнопки можна відобразити надані користувачем фінансування), Повідомлення для перегляду адресованих персональних коментарів (за допомогою окремої кнопки можна відобразити написані користувачем коментарі), Закладки для перегляду позик,

які були додані до закладок користувачем (за допомогою окремої кнопки можна відобразити надані користувачем рекомендації), кнопка для авторизації або кнопки для виходу та переходу до кабінету з власними даними (рис. 5.1).

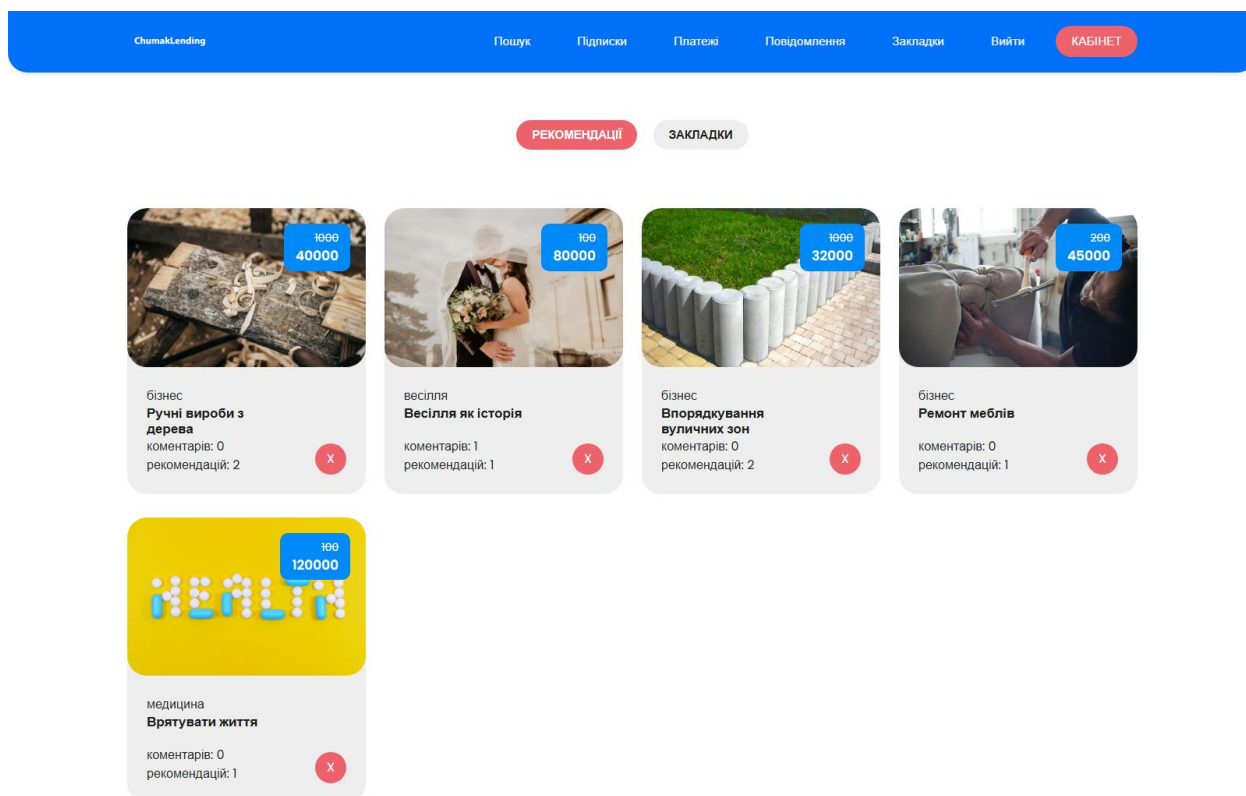


Рисунок 5.1 – Рекомендовані користувачем-кредитором позики

Зі сторінки кабінету доступна також кнопка для розміщення нової заявки на позику. Після того, як це виконано вперше, користувач отримує запис про користувача-позичальника. Відповідно після авторизації йому доступні відповідні засоби в меню, які використовуються для роботи з позиками (рис. 5.2). До таких засобів належить розділ Позик, в якому знаходиться перелік власних заявок на позики та кнопка для розміщення нової позики, та розділ Публікації для перегляду створених самостійно публікацій та створення нової публікації за допомогою відповідної кнопки.

Під час розміщення нової заявки на позику необхідно заповнити велику форму, яка передбачає внесення всіх необхідних даних, додавання фотографій, додавання файлів документів для підтвердження внесених

позичальником даних. Згори форма передбачає визначення інтервалів строків позики та сум позики. При цьому задається мінімальне, максимальне значення та крок. Це задає параметри для роботи методу керування параметрами позик, який був запропонований. Внизу форми знаходяться кнопки для збереження заявки або відправлення її для розгляду сервісом. Але згори наявна кнопка оцінювання, яка за заданими значеннями виконує на основі запропонованого методу визначення варіантів відсоткових ставок.

ChumakLending Пошук Підписки Позики Публікації Платежі Повідомлення Закладки Вийти **КАБІНЕТ**

РОЗМІЩЕННЯ НОВОЇ ЗАЯВКИ НА ПОЗИКУ

ЗМІНИТИ ФОТОГРАФІЮ

ДОДАТИ ФОТОГРАФІЮ

Коротка назва
Врятувати життя

Повна назва
Врятувати життя

Тип запиту
індивідуальний

Призначення позики
медицина

Мінімальна необхідна сума позики
207150

Базова необхідна сума позики
207150

Допустимий крок суми позики
0

Мінімальний допустимий строк позики, місяців
36

Максимальний допустимий строк позики, місяців
36

Допустимий крок строку, місяців
0

ОЦІНИТИ*

*використовуйте оцінювання тільки тоді, коли всі дані нижче задано
Термін: 36 місяців, сума: 207150, відсоткова ставка: 6.72 %

Опис

Рисунок 5.2 – Створення нової позики користувачем-позичальником

Представник сервісу надання позик у свою чергу має меню з 3 основних розділів та засобів авторизації. Основні розділи включають відображення позик, де доступне переключення між відображенням всіх позик за станом та пошуком, перегляду всіх платежів, виконаних для повернення коштів за позиками, та всіх фінансувань позик кредиторами (рис. 5.3). При цьому представник сервісу не можна створювати рекомендації, додавати до закладок позики, тому йому такі кнопки за позиками не відображаються.

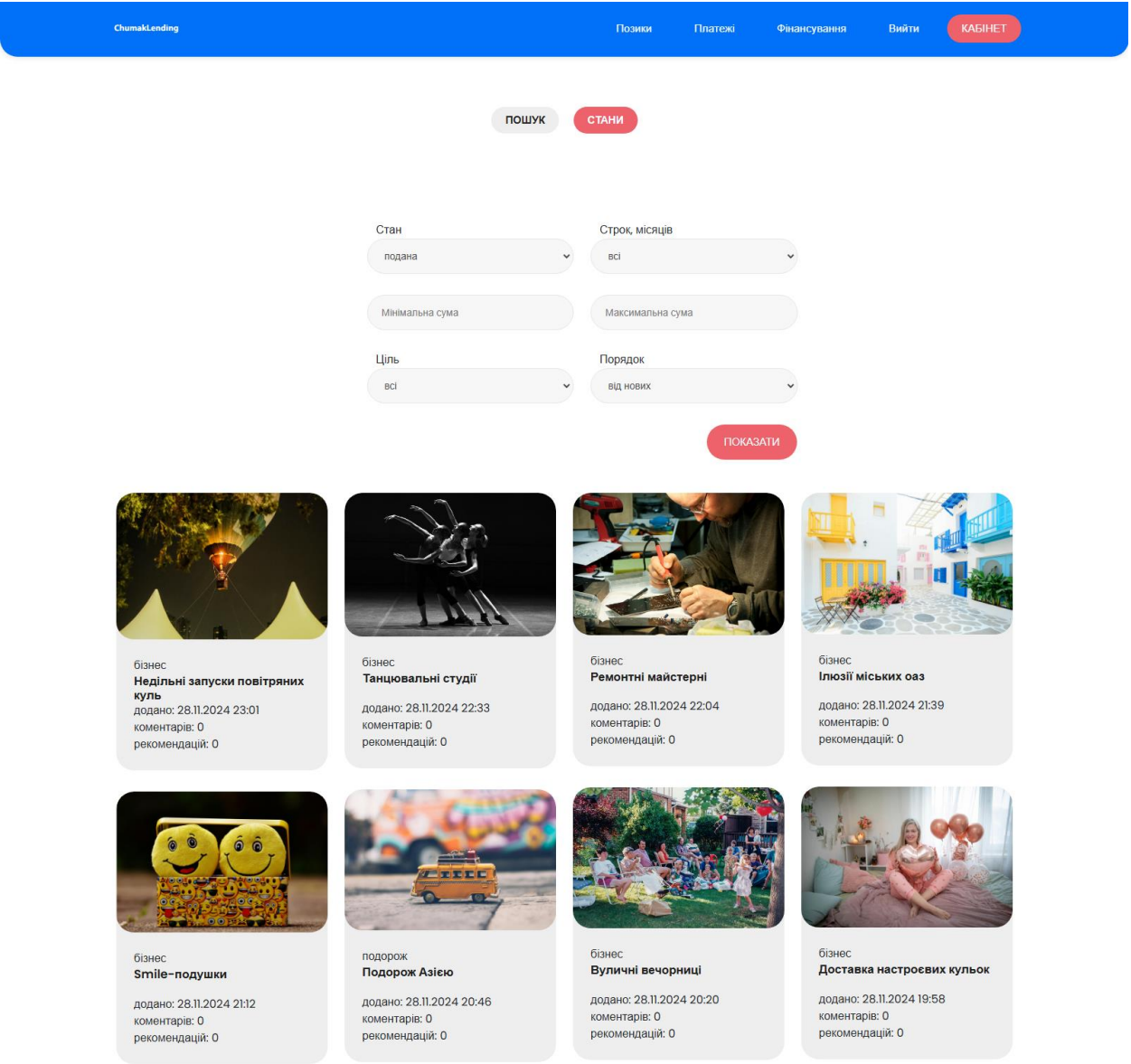


Рисунок 5.3 – Перегляд позик за станом представником сервісу надання
ПОЗИК

5.3 Експериментальне дослідження

Експериментальне дослідження прогнозування відсоткової ставки позики виконувалось за допомогою вибірки [2], до складу якої входить 10000 спостережень з платформи LendingClub [1].

Для того щоб проаналізувати те, на скільки ефективним є використання XGBoost [6] в якості базової моделі, було створено додаткові моделі на основі індивідуальних дерев рішень [10] та на основі методу k-найближчих сусідів [11]. При цьому порівняння всіх цих 3 методів було виконано для різної кількості вхідних ознак. Розглядався варіант, коли присутні всі ознаки, а також ряд варіантів зі зменшеною кількістю ознак. Для цього було використано дерево рішень [10], за допомогою якого було оцінено інформативність всіх ознак, а тоді відсортовано їх за зменшенням відповідної оцінки. Обмежуючи кількість вхідних ознак, наприклад, до 35, обирались перші 35 з ознак. У підсумку було розглянуто варіанти зі всіма ознаками, з 40, 39, 35, 20, 10 ознаками (табл. 5.1).

Отримані результати чітко вказують на те, що варіант з 39 вхідними ознаками дозволив отримати найкращий результат прогнозування відсоткової ставки позики як за середньою абсолютною похибкою, так і за кореневою середньоквадратичною похибкою. Фактично помітно, що і у випадку дерев рішень, і у випадку XGBoost відбувається збільшення похибки за обома показниками як при зменшенні кількості ознак, так і при збільшенні. Відповідно саме набір з 39 вхідних ознак слід вважати найбільш інформативним.

При цьому порівняння відбувалось у варіанті, коли ансамбль дерев рішень складався зі 100 дерев. Такий ансамбль характеризувався середньою абсолютною похибкою, що на 28,67 % менша за дерево рішень та на 21,05 % за метод k-найближчих сусідів, та кореневою середньоквадратичною похибкою, яка на 29,44 % менша за дерево рішень та на 18,41 % менша за метод k-найближчих сусідів.

Таблиця 5.1 – Результати прогнозування відсоткової ставки позики з різною кількістю вхідних ознак

Метод	Середня абсолютна похибка	Коренева середньоквадратична похибка
Дерево рішень зі всіма вхідними ознаками	0,00306778	0,00406196
Метод k-найближчих сусідів зі всіма вхідними ознаками	0,00274482	0,00349476
XGBoost зі всіма вхідними ознаками	0,00218533	0,00288257
Дерево рішень з 40 вхідними ознаками	0,00305814	0,00404539
Метод k-найближчих сусідів з 40 вхідними ознаками	0,00274482	0,00349476
XGBoost з 40 вхідними ознаками	0,00218432	0,0028834
Дерево рішень з 39 вхідними ознаками	0,00303764	0,0040409
Метод k-найближчих сусідів з 39 вхідними ознаками	0,00274482	0,00349476
XGBoost з 39 вхідними ознаками	0,0021669	0,00285138
Дерево рішень з 35 вхідними ознаками	0,00304254	0,00404128
Метод k-найближчих сусідів з 35 вхідними ознаками	0,00274482	0,00349476
XGBoost з 35 вхідними ознаками	0,00218698	0,00289246
Дерево рішень з 20 вхідними ознаками	0,0030936	0,00407879
Метод k-найближчих сусідів з 20 вхідними ознаками	0,00272249	0,0034802
XGBoost з 20 вхідними ознаками	0,00221248	0,00289896
Дерево рішень з 10 вхідними ознаками	0,00314124	0,00419205
Метод k-найближчих сусідів з 10 вхідними ознаками	0,00274392	0,0034831
XGBoost з 10 вхідними ознаками	0,00234291	0,00305467

Окрім того було детально розглянуто перелік вхідних ознак та їх інформативність (табл. 5.2). Отримані значення вказують на те, що переважна кількість ознак дійсно є достатньо інформативними для створення моделі.

Таблиця 5.2 – Найбільш інформативна ознака

Ознака	Інформативність
total_debit_limit	0,142439024
term	0,130443568
debt_to_income	0,081710937
total_credit_limit	0,047285566
loan_amount	0,042108036
total_credit_utilized	0,039191262
initial_listing_status	0,034256673
num_active_debit_accounts	0,031599412
earliest_credit_line	0,030700599
emp_title	0,028254491
months_since_last_credit_inquiry	0,028179545
annual_income	0,027936361
verified_income	0,027197975
total_credit_lines	0,025636136
accounts_opened_24m	0,025340577
state	0,025331115
months_since_last_delinq	0,023433871
num_mort_accounts	0,021681495
num_cc_carrying_balance	0,020068028
loan_purpose	0,016734338
account_never_delinq_percent	0,015907259
num_total_cc_accounts	0,014968712
inquiries_last_12m	0,013675264

Деякі з ознак тим не менш мають близько до нульової інформативність (табл. 5.3). Цим і пояснюється збільшення похибки при використанні повного набору ознак. Фактично ці результати підтверджують, що використання 39 вхідних ознак є найкращим варіантом для створення моделі.

Таблиця 5.3 – Найменш інформативні ознаки

Ознака	Інформативність
num_satisfactory_accounts	0,01219083
num_open_cc_accounts	0,012102101
total_collection_amount_ever	0,01122676
current_installment_accounts	0,010655527
emp_length	0,01057346
annual_income_joint	0,009959651
debt_to_income_joint	0,009294906
homeownership	0,007167431
open_credit_lines	0,006865971
months_since_90d_late	0,00653473
delinq_2y	0,002877278
verification_income_joint	0,002514928
num_historical_failed_to_pay	0,001980489
application_type	0,000797966
num_collections_last_12m	0,000589372
public_record_bankrupt	0,000564209
tax_liens	2,41E-05
current_accounts_delinq	0

Окрім того було детально також проаналізовано використання моделі на основі методу XGBoost стосовно кількості дерев рішень у ансамблі (табл. 5.4). Отримані результати вказують, що найкращий результат вдалось отримати при використанні ансамблю з 20 дерев у запропонованому методі.

При більшій та меншій кількості дерев похибка збільшувалась. При створенні тестової вибірки 20 % всіх даних відносилось до неї.

Таблиця 5.4 – Результати прогнозування відсоткової ставки позики на основі запропонованого методу за різної кількості дерев рішень у ансамблі

Метод	Середня абсолютна похибка	Коренева середньоквадратична похибка
5 дерев у ансамблі	0,0022886	0,00292744
10 дерев у ансамблі	0,0021741	0,00283498
20 дерев у ансамблі	0,00212651	0,00280321
30 дерев у ансамблі	0,00214366	0,00281935
40 дерев у ансамблі	0,00215425	0,00283353
50 дерев у ансамблі	0,00216492	0,00284691
100 дерев у ансамблі	0,0021669	0,00285138
200 дерев у ансамблі	0,00219565	0,0028994

За найкращого результату запропонований метод характеризується середньою абсолютною похибкою, що на 29,99 % менша за дерево рішень та на 22,53 % за метод k-найближчих сусідів, та кореневою середньоквадратичною похибкою, яка на 30,63 % менша за дерево рішень та на 19,79 % менша за метод k-найближчих сусідів.

5.4 Висновки за розділом 5

Розділ містить опис роботи з програмою зі сторони різних користувачів, результати експериментального дослідження прогнозування відсоткової ставки позики на основі запропонованого методу та альтернативних варіантів, включаючи дослідження впливу різної кількості ознак, різної кількості дерев рішень у ансамблі. Запропонований метод

характеризується середньою абсолютною похибкою, що на 29,99 % менша за дерево рішень та на 22,53 % за метод k -найближчих сусідів, та кореневою середньоквадратичною похибкою, яка на 30,63 % менша за дерево рішень та на 19,79 % менша за метод k -найближчих сусідів.

ВИСНОВКИ

У результаті виконання дипломної кваліфікаційної роботи магістра розроблено програмне забезпечення підтримки надання позик на основі фінансування їх фізичними особами для зменшення вимог до визначення параметрів фінансування позик. Для роботи над програмним проєктом використано мову Python, бібліотеку XGBoost, фреймворк Django, пакети Numpy, Pandas, Scikit-learn, систему PostgreSQL.

Програма призначена для підтримки надання позик між фізичними особами, коли фінансові ресурси для позик надаються не банківськими установами, а безпосередньо фізичними особами, які згодні вкласти їх на фінансування певної діяльності інших осіб, отримуючи в підсумку відсотки за надані фінанси. Програма має використовуватися позичальниками, кредиторами та представниками сервісу, які виконують керування позиками, визначаючи ті позики, які мають бути заблоковані від доступу до позичальників, а для всіх інших позик встановлюючи відсоткові ставки та надаючи їх таким чином до уваги потенційним позичальникам. Для взаємодії та сприяння розумінню позичальників кредиторами першим надається можливість створювати власні публікації, таким чином знайомлячи кредиторів з інтересами, мисленням, позиціонуванням позичальників.

Наукова новизна роботи полягає в методі керування параметрами позик, який характеризується використанням дерев рішень на основі створення ансамблів методом XGBoost для прогнозування відсоткової ставки позики за значеннями вхідних ознак, що дозволяє встановлювати варіативні значення параметрів позик таким чином, щоб сприяти результативності процесу збору коштів від фізичних осіб. Запропонований метод характеризується середньою абсолютною похибкою, що на 29,99 % менша за дерево рішень та на 22,53 % за метод k-найближчих сусідів, та кореневою середньоквадратичною похибкою, яка на 30,63 % менша за дерево рішень та на 19,79 % менша за метод k-найближчих сусідів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Online Personal Loans. Full-Service Banking. LendingClub [Electronic resource]. – Access mode : <https://www.lendingclub.com>.
2. Arvidsson, J. Lending Club Data [Electronic resource] / J. Arvidsson. – Access mode : <https://www.kaggle.com/datasets/joebeachcapital/lending-club>.
3. Loan data from Lending Club [Electronic resource]. – Access mode : https://www.openintro.org/data/index.php?data=loans_full_schema.
4. LendingClub Reports Third Quarter 2024 Results. LendingClub Corporation [Electronic resource]. – Access mode : <https://ir.lendingclub.com/news/news-details/2024/LendingClub-Reports-Third-Quarter-2024-Results/default.aspx>.
5. Millerbernd, A. LendingClub Personal Loans [Electronic resource] : 2024 Review – NerdWallet / A. Millerbernd, K. Lowe. – Access mode : <https://www.nerdwallet.com/reviews/loans/personal-loans/lendingclub-personal-loans>.
6. XGBoost Tutorials – xgboost 2.1.1 documentation [Electronic resource]. – Access mode : <https://xgboost.readthedocs.io/en/stable/tutorials/index.html>.
7. Django [Electronic resource]. – Access mode : <https://github.com/django/django>.
8. Ruby on Rails Guides [Electronic resource]. – Access mode : <https://guides.rubyonrails.org/>.
9. Free Template 589 Lugx Gaming [Electronic resource]. – Access mode : <https://templatemo.com/tm-589-lugx-gaming>.
10. Decision Tree Regression – scikit-learn 1.5.2 documentation [Electronic resource]. – Access mode : https://scikit-learn.org/1.5/auto_examples/tree/plot_tree_regression.html.
11. K-Nearest Neighbor (KNN) Algorithm [Electronic resource]. – Access mode : <https://www.geeksforgeeks.org/k-nearest-neighbours/>.

ДОДАТОК А
Технічне завдання

Вступ

Надання позик зазвичай забезпечується банківськими установами, інвестиційними фондами. Ці установи націлюють свою діяльність на надання фінансових ресурсів за певну сплату. Проте такий варіант є достатньо традиційним, відповідно і відсоткові ставки за таке фінансування є доволі обмеженими, а позичальник може не відповідати певній частині вимог, тоді не маючи можливість отримати кошти. З іншого боку відреагувати на такі ситуації можна, наприклад, збільшенням відсоткової ставки. Проте для цього потрібне інше джерело фінансування, адже банки в такій діяльності є обмеженими. Таким іншим джерелом фінансування і можуть бути фізичні особи. Тоді параметри позик, на основі яких має відбуватися фінансування, а також відсіювати непридатні пропозиції має сервіс, а фінансування надавати фізичні особи. Для підтримки саме такої роботи і направлене програмне забезпечення, яке має бути розроблено в цій роботі. З іншого боку для зменшення навантаження на працівників сервісу необхідно надати можливість програмного прогнозування відсоткової ставки за параметрами позики, після чого представник сервісу може або змінити, або підтвердити пропоноване значення, контролюючи отриманий результат. Відповідно надання таких засобів підтримки надання позик є важливим.

A.1 Підстави для розробки

На підставі наказу № 398 від 7 жовтня 2024 р., який був виданий у Національному університеті «Запорізька політехніка», де і ведеться розробка в межах даної дипломної кваліфікаційної роботи магістра, визначена тема роботи «Дослідження та програмна реалізація методів керування параметрами позик», що було деталізовано також виданим завданням на роботу.

A.2 Призначення розробки

Програма призначена для підтримки надання позик між фізичними особами, коли фінансові ресурси для позик надаються не банківськими установами, а безпосередньо фізичними особами, які згодні вкласти їх на фінансування певної діяльності інших осіб, отримуючи в підсумку відсотки за надані фінанси. Програма має використовуватися позичальниками, кредиторами та представниками сервісу, які виконують керування позиками, визначаючи ті позики, які мають бути заблоковані від доступу до позичальників, а для всіх інших позик встановлюючи відсоткові ставки та надаючи їх таким чином до уваги потенційним позичальникам. Для взаємодії та сприяння розумінню позичальників кредиторами першим надається можливість створювати власні публікації, таким чином знайомлячи кредиторів з інтересами, мисленням, позиціонуванням позичальників.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Для реалізації комплексної роботи вебзастосунку підтримки надання позик необхідно забезпечити такі функціональні можливості:

- пошук серед позик;
- перегляд сторінки користувача-позичальника;
- перегляд публікації користувача-позичальника;
- підписування на публікації користувача-позичальника;
- скасування підписки на публікації користувача-позичальника;
- перегляд підписок;
- публікація матеріалу;
- перегляд сторінки користувача-кредитора;
- внесення загальних даних про користувача-позичальника;
- внесення даних про користувача-кредитора;

- розміщення заявки на позику;
- верифікація заявки на позику;
- скасування заявки на позику;
- редагування публікації;
- перегляд матеріалів заявки на позику;
- верифікація документів за позикою;
- редагування заявки на позику;
- додавання позичальника для сумісної позики;
- перегляд заявок на позики користувача;
- внесення власних даних до спільної заявки на позику;
- фінансування позики;
- перегляд профінансованих користувачем позик;
- перегляд фінансувань позики;
- перегляд отриманих користувачем платежів;
- перегляд платежів за позикою;
- коментування позики;
- перегляд персональних повідомлень;
- коментування публікації;
- створення закладки за позикою або скасування її;
- рекомендування позики або скасування її;
- перегляд рекомендацій позик користувача;
- перегляд рекомендацій за позикою;
- перегляд позик у закладках;
- перегляд власних коментарів;
- авторизація і реєстрація;
- перегляд всіх виконаних платежів;
- перегляд позик за станом;
- перегляд всіх виконаних фінансувань.

A.3.2 Умови експлуатації

Експлуатація програмного забезпечення підтримки надання позик має забезпечуватися виконанням ряду вимог до серверної частини:

- забезпечення вебсерверу з підтримкою інтерпретатора мови програмування Python 3;
- налаштування Django-застосунку для виконання на сервері з реалізацією міграції для забезпечення необхідної структури бази даних за допомогою підтримки системи керування базами даних PostgreSQL;
- інсталювані Python-пакети Numpy, Pandas, Scikit-learn.

A.3.3 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення підтримки надання позик має забезпечуватися серверною частиною з виконанням таких мінімальних параметрів технічних засобів:

- двоядерний процесор;
- 4 Гб оперативної пам'яті;
- 10 Гб простору на жорсткому диску, що необхідно для збереження даних самого вебсайту та фотографій, файлів, БД: в залежності від якості, кількості фотографій можливе подальше збільшення мінімальних вимог.

A.4 Порядок контролю та приймання

Робота має виконуватися у відповідності з затвердженим завданням і контролюється керівником. Отримані результати роботи представляються в пояснювальній записці.

Завершальним етапом виконання та відповідно контролю і приймання дипломної кваліфікаційної роботи магістра є її захист, який виконується перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

from .models import Region
from .models import Obligee
from .models import Borrower
from .models import BorrowerEmployee
from .models import Employee
from .models import LoanBorrower
from .models import ApproveFiletype
from .models import LoanBorrowerApprove
from .models import Loan
from .models import LoanStatus
from .models import LoanFile
from .models import LoanPhoto
from .models import LoanPayment
from .models import LoanComment
from .models import LoanInvestment
from .models import BorrowerPost
from .models import BorrowerPostPhoto
from .models import BorrowerPostComment
from django.shortcuts import render
from django.core.paginator import Paginator
from django.db.models import Count

def borrower_page_view(request, borrower_user = None):
    personal = False
    if not borrower_user:
        personal = True
        borrower_user =
Borrower.objects.select_related("obligeeaccount",
"obligeeaccount__useraccount").prefetch_related("followers",
"obligeeaccount__loanbookmarks")).filter(obligeeaccount__useracc
ount = request.user)
        borrower_user.followers_count =
borrower_user.followers.count()
    else:
        try:
            borrower_user =
Borrower.objects.select_related("obligeeaccount",
"obligeeaccount__useraccount").prefetch_related("followers",
"obligeeaccount__loanbookmarks")).filter(obligeeaccount__useracc
ount__username = borrower_user)
            borrower_user.followers_count =
borrower_user.followers.count()
        except Borrower.DoesNotExist:
            borrower_user = None
            return render(request,
"chumaklending/borrower.html", {"error": "Такого користувача-
позичальника немає. Помилка заданого значення."})
    if personal:
        borrower_loans =
LoanBorrower.objects.select_related("loan",
"borrower").prefetch_related("borrower__followers",
"loan__recommends").filter(borrower = borrower_user)

```

```

else:
    borrower_loans =
LoanBorrower.objects.select_related("loan",
"borrower").prefetch_related("borrower__followers",
"loan__recommends").filter(borrower = borrower_user,
loan__status__gt = 1)
    borrower_loans_last = borrower_loans.order_by("-
loan__addedat")[:10]
    for borrower_loan in borrower_loans_last:
        borrower_loan.photo = LoanPhoto.objects.filter(loan
= borrower_loan.loan).order_by("id")[0]
        borrower_loan.recommends_count =
borrower_loan.loan.recommends.count()
        borrower_loan.comments_count =
LoanComment.objects.filter(loan = borrower_loan.loan).count()
        borrower_loans_best =
borrower_loans.annotate(recommends_count=models.Count('loan_reco
mmends')).order_by("-recommends_count")[:10]
        for borrower_loan in borrower_loans_best:
            borrower_loan.photo = LoanPhoto.objects.filter(loan
= borrower_loan.loan).order_by("id")[0]
            borrower_loan.recommends_count =
borrower_loan.loan.recommends.count()
            borrower_loan.comments_count =
LoanComment.objects.filter(loan = borrower_loan.loan).count()
            borrower_posts = BorrowerPost.objects.filter(borrower =
borrower_user)
            borrower_posts = borrower_posts.order_by("-createdat")
            for borrower_post in borrower_posts:
                borrower_post.photo =
BorrowerPostPhoto.objects.filter(borrowerpost =
borrower_post).order_by("id")[0]
                auth_borrower_user =
Obligee.objects.prefetch_related("loanbookmarks").filter(useracc
ount = request.user)
                if auth_borrower_user:
                    for borrower_loan in borrower_loans_last:
                        if borrower_loan.loan in
auth_borrower_user.loanbookmarks:
                            borrower_loan.bookmark = True
                        else:
                            borrower_loan.bookmark = False
                    for borrower_loan in borrower_loans_best:
                        if borrower_loan.loan in
auth_borrower_user.loanbookmarks:
                            borrower_loan.bookmark = True
                        else:
                            borrower_loan.bookmark = False
                    for borrower_loan in borrower_loans_last:
                        if auth_borrower_user in
borrower_loan.loan.recommends:
                            borrower_loan.recommend = True
                        else:

```

```

        borrower_loan.recommend = False
    for borrower_loan in borrower_loans_best:
        if auth_borrower_user in
borrower_loan.loan.recommends:
            borrower_loan.recommend = True
        else:
            borrower_loan.recommend = False
    if auth_borrower_user in borrower.followers:
        borrower.follower = True
    else:
        borrower.follower = False
    return render(request, "chumaklending/borrower.html",
{"borrower": borrower, "loans_best": borrower_loans_best,
"loans_last": borrower_loans_last })

def borrower_loans_view(request, ordering, borrower_user =
None):
    personal = False
    if not borrower_user:
        personal = True
        borrower_user =
Borrower.objects.select_related("obligeeaccount",
"obligeeaccount__useraccount").prefetch_related("followers",
"obligeeaccount__loanbookmarks")).filter(obligeeaccount__useracc
ount = request.user)
        borrower_user.followers_count =
borrower_user.followers.count()
    else:
        try:
            borrower_user =
Borrower.objects.select_related("obligeeaccount",
"obligeeaccount__useraccount").prefetch_related("followers").fil
ter(obligeeaccount__useraccount__username = borrower_user)
            borrower_user.followers_count =
borrower_user.followers.count()
        except Borrower.DoesNotExist:
            borrower_user = None
            return render(request,
"chumaklending/borrower_loans.html", {"error": "Такого
користувача-позичальника немає. Помилка заданого значення."})
        if personal:
            borrower_loans =
LoanBorrower.objects.select_related("loan",
"borrower").prefetch_related("borrower__followers",
"loan__recommends").filter(borrower = borrower_user)
        else:
            borrower_loans =
LoanBorrower.objects.select_related("loan",
"borrower").prefetch_related("borrower__followers",
"loan__recommends").filter(borrower = borrower_user,
loan__status__gt = 1)
            if ordering == 0:

```



```

        if borrower_loan.loan in
auth_borrower_user.loanbookmarks:
            borrower_loan.bookmark = True
        else:
            borrower_loan.bookmark = False
    for borrower_loan in borrower_loans:
        if auth_borrower_user in
borrower_loan.loan.recommends:
            borrower_loan.recommend = True
        else:
            borrower_loan.recommend = False
    if auth_borrower_user in borrower.followers:
        borrower.follower = True
    else:
        borrower.follower = False
    return render(request,
"chumaklending/borrower_loans.html", {"borrower": borrower,
"loans": borrower_loans })

def state_loans_view(request, loanstate, ordering = 0, term
= 0, purpose = -1, amount_min = 0, amount_max = 0):
    auth_borrower_user =
Obligee.objects.prefetch_related("loanbookmarks").filter(useracc
ount = request.user)
    if request.user.is_staff:
        state_loans = Loan.objects.filter(state =
loanstate)
        if term != 0:
            state_loans = state_loans.filter(term = term)
        if purpose != 0:
            state_loans = state_loans.filter(purpose =
purpose)
        if amount_min != 0:
            state_loans = state_loans.filter(amount__gte =
amount_min)
        if amount_max != 0:
            state_loans = state_loans.filter(amount__lte =
amount_max)
        if ordering == 0:
            state_loans = state_loans.order_by("-addedat")
        elif ordering == 1:
            state_loans = state_loans.order_by("addedat")
        elif ordering == 2:
            state_loans =
state_loans.annotate(recommends_count=models.Count('recommends')
).order_by("-recommends_count")
        elif ordering == 3:
            state_loans =
state_loans.annotate(recommends_count=models.Count('recommends')
).order_by("recommends_count")
        elif ordering == 4:
            state_loans = state_loans.order_by("caption")
        elif ordering == 5:

```

```

        state_loans = state_loans.order_by("-caption")
    elif ordering == 6:
        state_loans =
state_loans.order_by("collectedat")
    elif ordering == 7:
        state_loans = state_loans.order_by("-
collectedat")

    loanpaginator = Paginator(state_loans, 20)
    loanpage = request.GET.get("page")
    state_loans = loanpaginator.get_page(loanpage)
    for borrower_loan in state_loans:
        borrower_loan.photo =
LoanPhoto.objects.filter(loan =
borrower_loan.loan).order_by("id")[0]
        borrower_loan.recommends_count =
borrower_loan.loan.recommends.count()
        borrower_loan.comments_count =
LoanComment.objects.filter(loan = borrower_loan.loan).count()
        return render(request,
"chumaklending/state_loans.html", {"loans": borrower_loans })
    return render(request,
"chumaklending/state_loans.html", {"error": "Доступно тільки для
використання сервісом" })

```

```

from django.db.models import ExpressionWrapper, F

def find_loans_view(request):
    if request.user.is_staff:
        found_loans = Loan.objects.all()
    else:
        found_loans = Loan.objects.filter(state__gt = 1)
        loan_caption = request.GET.get("loancaption")
        only_caption = request.GET.get("onlycaption")
        if loan_caption and not only_caption:
            found_loans =
found_loans.filter(Q(caption__icontains = loan_caption) |
Q(fulltitle__icontains = loan_caption) |
Q(shortdescription__icontains = loan_caption) |
Q(fullpresentation__icontains = loan_caption))
            elif loan_caption:
                found_loans =
found_loans.filter(fulltitle__icontains = loan_caption)
                loan_purpose = request.GET.get("purpose")
                if loan_purpose:
                    loan_purpose = int(loan_purpose)
                    found_loans = found_loans.filter(purpose =
loan_purpose)
                loan_purposes = request.GET.get("purposes")
                if loan_purposes:
                    loan_purposes = loan_purposes.split(",")
                    loan_purposes_list = []
                    for purpose in loan_purposes:
                        loan_purposes_list.append(int(purpose))

```

```

        found_loans = found_loans.filter(purpose__in =
loan_purposes_list)
        loan_min_amount = request.GET.get("minamount")
        if loan_min_amount:
            loan_min_amount = int(loan_min_amount)
            found_loans = found_loans.filter(amount__gte =
loan_min_amount)
        loan_max_amount = request.GET.get("maxamount")
        if loan_max_amount:
            loan_max_amount = int(loan_max_amount)
            found_loans = found_loans.filter(amount__lte =
loan_max_amount)
        loan_min_init_amount = request.GET.get("mininamount")
        if loan_min_init_amount:
            loan_min_init_amount = int(loan_min_init_amount)
            found_loans = found_loans.filter(initialamount__gte
= loan_min_init_amount)
        loan_max_init_amount = request.GET.get("maxinamount")
        if loan_max_init_amount:
            loan_max_init_amount = int(loan_max_init_amount)
            found_loans = found_loans.filter(initialamount__lte
= loan_max_init_amount)
        loan_min_coll_amount = request.GET.get("mincamount")
        if loan_min_coll_amount:
            loan_min_coll_amount = int(loan_min_coll_amount)
            found_loans =
found_loans.filter(amountcollected__gte = loan_min_coll_amount)
        loan_max_coll_amount = request.GET.get("maxcamount")
        if loan_max_coll_amount:
            loan_max_coll_amount = int(loan_max_coll_amount)
            found_loans =
found_loans.filter(amountcollected__lte = loan_max_coll_amount)
        joint = request.GET.get("joint")
        joint = int(joint)
        if joint == 0:
            found_loans = found_loans.filter(applicationtype =
True)
        elif joint == 1:
            found_loans = found_loans.filter(applicationtype =
False)
        disbursement = request.GET.get("dis")
        disbursement = int(disbursement)
        if disbursement == 0:
            found_loans = found_loans.filter(disbursement =
True)
        elif disbursement == 1:
            found_loans = found_loans.filter(disbursement =
False)
        loan_status = request.GET.get("status")
        if loan_status:
            loan_status = int(loan_status)
            found_loans = found_loans.filter(status =
loan_status)

```

```

loan_statuses = request.GET.get("statuses")
if loan_statuses:
    loan_statuses = loan_statuses.split(",")
    loan_statuses_list = []
    for status in loan_statuses:
        loan_statuses_list.append(int(status))
    found_loans = found_loans.filter(status__in =
loan_statuses_list)
    loan_term = request.GET.get("term")
    if loan_term:
        loan_term = int(loan_term)
        found_loans = found_loans.filter(term = loan_term)
    loan_terms = request.GET.get("terms")
    if loan_terms:
        loan_terms = loan_terms.split(",")
        loan_terms_list = []
        for term in loan_terms:
            loan_terms_list.append(int(term))
        found_loans = found_loans.filter(term__in =
loan_terms_list)
    loan_min_intrate = request.GET.get("minintrate")
    if loan_min_intrate:
        loan_min_intrate = float(loan_min_intrate)
        found_loans = found_loans.filter(interestrate__gte
= loan_min_intrate)
    loan_max_intrate = request.GET.get("maxintrate")
    if loan_max_intrate:
        loan_max_intrate = float(loan_max_intrate)
        found_loans = found_loans.filter(interestrate__lte
= loan_max_intrate)
    collected_part_min = request.GET.get("minrate")
    if collected_part_min:
        found_loans = found_loans.annotate
(coll=ExpressionWrapper(F('amountcollected') * 100.0 /
F('amount'), output_field=FloatField()))
        collected_part_min = float(collected_part_min)
        found_loans = found_loans.filter(coll__gte =
collected_part_min)
    collected_part_max = request.GET.get("maxrate")
    if collected_part_max:
        if not collected_part_min:
            found_loans = found_loans.annotate
(coll=ExpressionWrapper(F('amountcollected') * 100.0 /
F('amount'), output_field=FloatField()))
            collected_part_max = float(collected_part_max)
            found_loans = found_loans.filter(coll__lte =
collected_part_max)
    loan_min_grade = request.GET.get("mingrade")
    loan_max_grade = request.GET.get("maxgrade")
    if not loan_max_grade:
        loan_max_grade = "G5"
    if not loan_min_grade:
        loan_min_grade = "A1"

```

```

        possible_grades = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
        min_grade_letter_1 = loan_min_grade[0]
        min_grade_letter_1 =
possible_grades.index(min_grade_letter_1)
        min_grade_letter_2 = loan_min_grade[1]
        min_grade_letter_2 = int(min_grade_letter_2)
        max_grade_letter_1 = loan_max_grade[0]
        max_grade_letter_1 =
possible_grades.index(max_grade_letter_1)
        max_grade_letter_2 = loan_max_grade[1]
        max_grade_letter_2 = int(max_grade_letter_2)
        letters = list(range(min_grade_letter_1,
max_grade_letter_1 + 1 ))
        available_grades = []
        for letter in letters:
            if letter == min_grade_letter_1:
                possible_digit_min = min_grade_letter_2
            else:
                possible_digit_min = 1
            if letter == max_grade_letter_2:
                possible_digit_max = max_grade_letter_2
            else:
                possible_digit_max = 1
            digits = list(range(possible_digit_min,
possible_digit_max+1))
            for digit in digits:
                code = possible_grades[letter]+str(digit)
                available_grades.append(code)
        found_loans = found_loans.filter(grade__in =
available_grades)
        loan_employees = request.GET.get("employees")
        if loan_employees:
            loan_employees = loan_employees.split(",")
            loan_employees_list = []
            for employee in loan_employees:
                loan_employees_list.append(int(employee))
            employee_loans = []
            borrower_employees =
BorrowerEmployee.objects.filter(employee__id__in =
loan_employees_list).values_list("borrower__id")
            for loan in found_loans:
                borrowers = LoanBorrower.objects.filter(loan =
loan, borrower__id__in = borrower_employees)
                if borrowers:
                    employee_loans.append(loan.id)
            found_loans = found_loans.filter(id__in =
employee_loans)
        loan_ordering = request.GET.get("ordering")
        if loan_ordering:
            loan_ordering = int(loan_ordering)
        else:
            loan_ordering = 0
        if ordering == 0:

```

```

        found_loans = found_loans.order_by("-addedat")
    elif ordering == 1:
        found_loans = found_loans.order_by("addedat")
    elif ordering == 2:
        found_loans =
found_loans.annotate(recommends_count=models.Count('recommends'))
        found_loans.order_by("-recommends_count")
    elif ordering == 3:
        found_loans =
found_loans.annotate(recommends_count=models.Count('recommends'))
        found_loans.order_by("recommends_count")
    elif ordering == 4:
        found_loans = found_loans.order_by("caption")
    elif ordering == 5:
        found_loans = found_loans.order_by("-caption")
    elif ordering == 6:
        found_loans = found_loans.order_by("collectedat")
    elif ordering == 7:
        found_loans = found_loans.order_by("-
collectedat")
    elif ordering == 8:
        found_loans = found_loans.order_by("amount")
    elif ordering == 9:
        found_loans = found_loans.order_by("-
amount")
    elif ordering == 10:
        found_loans =
found_loans.order_by("amountcollected")
    elif ordering == 11:
        found_loans = found_loans.order_by("-
amountcollected")
    loanpaginator = Paginator(found_loans, 20)
    loanpage = request.GET.get("page")
    found_loans = loanpaginator.get_page(loanpage)
    for borrower_loan in found_loans:
        borrower_loan.photo = LoanPhoto.objects.filter(loan
= borrower_loan.loan).order_by("id")[0]
        borrower_loan.recommends_count =
borrower_loan.loan.recommends.count()
        borrower_loan.comments_count =
LoanComment.objects.filter(loan = borrower_loan.loan).count()
        auth_borrower_user =
Obligee.objects.prefetch_related("loanbookmarks").filter(useracc
ount = request.user)
        if auth_borrower_user:
            for borrower_loan in found_loans:
                if borrower_loan.loan in
auth_borrower_user.loanbookmarks:
                    borrower_loan.bookmark = True
                else:
                    borrower_loan.bookmark = False
            for borrower_loan in found_loans:

```



```

        else:
            loan_part.borrower.follower = False
        if personal:
            loan_part.files =
LoanBorrowerApprove.objects.select_related("verifiedby",
"approvefiletype").filter(loanborrower =
loan_part).order_by("loadat")
        else:
            loan_part.files =
LoanBorrowerApprove.objects.select_related("verifiedby",
"approvefiletype").filter(loanborrower = loan_part, personal =
False).order_by("loadat")
        if not personal:
            comments = LoanComment.objects.filter(loan = loan,
personal = False).order_by("-createdat")
        else:
            comments_onlypersonal =
request.GET.get("onlypersonal")
            if comments_onlypersonal == 0:
                comments = LoanComment.objects.filter(loan =
loan, personal = True).order_by("-createdat")
            elif comments_onlypersonal == 1:
                comments = LoanComment.objects.filter(loan =
loan, personal = False).order_by("-createdat")
            elif comments_onlypersonal == 2:
                comments = LoanComment.objects.filter(loan =
loan).order_by("-createdat")
            loan.comments = comments[:10]
            if not personal:
                loan.files = LoanFile.objects.filter(loan = loan,
personal = False).order_by("loadat")
            else:
                loan.files = LoanFile.objects.filter(loan =
loan).order_by("loadat")
            loan.comments_count = loan.comments.count()
            if borrower_user:
                if loan in borrower_user.loanbookmarks:
                    loan.bookmark = True
                else:
                    loan.bookmark = False
            if borrower_user in loan.recommends:
                loan.recommend = True
            else:
                loan.recommend = False
            investments = LoanInvestment.objects.filter(loan =
loan)
            investments = investments.order_by("-payedat")
            loan.investments = investments[:10]
            payments =
LoanPayment.objects.filter(loaninvestment__loan = loan)
            payments = payments.order_by("-fixeddata")
            loan.payments_count = payments.count()
            loan.payments = payments[:10]

```

```

        return render(request, "chumaklending/loan.html",
{"loan": loan })

    def loan_investments_view(request, loanid, ordering = 0):
        personal = False
        try:
            borrower_user =
Borrower.objects.select_related("obligeeaccount",
"obligeeaccount__useraccount").prefetch_related("followers",
"obligeeaccount__loanbookmarks")).filter(obligeeaccount__useracc
ount = request.user)
        except Borrower.DoesNotExist:
            borrower_user = None
        try:
            loan = Loan.objects.select_related("borrower",
"borrower__obligeeaccount",
"borrower__obligeeaccount__useraccount").prefetch_related("borro
wer__followers", "recommends").filter(id = loanid)
        except Loan.DoesNotExist:
            loan = None
            return render(request, "chumaklending/loan.html",
{"error": "Такої заявки на позику не існує." })
            if loan.borrower == borrower_user or
request.user.is_staff:
                personal = True
            if not personal and loan.status <= 1:
                return render(request, "chumaklending/loan.html",
{"error": "Ви не маєте доступу до цієї заявки на позику на дану
мить." })
            loan.photo = LoanPhoto.objects.filter(loan =
loan).order_by("id")[0]
            loan.recommends_count = loan.recommends.count()
            if not personal:
                comments = LoanComment.objects.filter(loan = loan,
personal = False).order_by("createdat")
            else:
                comments = LoanComment.objects.filter(loan =
loan).order_by("createdat")
            loan.comments_count = loan.comments.count()
            if borrower_user:
                if loan in borrower_user.loanbookmarks:
                    loan.bookmark = True
                else:
                    loan.bookmark = False
            if borrower_user in loan.recommends:
                loan.recommend = True
            else:
                loan.recommend = False
            investments = LoanInvestment.objects.filter(loan =
loan)
            if ordering == 0:
                investments = investments.order_by("payedat")
            elif ordering == 1:

```

```

        investments = investments.order_by("-payedat")
    elif ordering == 2:
        investments = investments.order_by("amount")
    elif ordering == 3:
        investments = investments.order_by("-amount")
    loan.investments_count = investments.count()
    loan.paginator = Paginator(investments, 20)
    loanpage = request.GET.get("invpage")
    loan.investments = loan.paginator.get_page(loanpage)
    return render(request, "chumaklending/loan_invs.html",
{"loan": loan })

def loan_payments_view(request, loanid, ordering = 0):
    personal = False
    try:
        borrower_user =
Borrower.objects.select_related("obligeeaccount",
"obligeeaccount__useraccount").prefetch_related("followers",
"obligeeaccount__loanbookmarks").filter(obligeeaccount__useracco
unt = request.user)
    except Borrower.DoesNotExist:
        borrower_user = None
    try:
        loan = Loan.objects.select_related("borrower",
"borrower__obligeeaccount",
"borrower__obligeeaccount__useraccount").prefetch_related("borro
wer__followers", "recommends").filter(id = loanid)
    except Loan.DoesNotExist:
        loan = None
    return render(request, "chumaklending/loan.html",
{"error": "Такої заявки на позику не існує." })
    if loan.borrower == borrower_user or
request.user.is_staff:
        personal = True
        if not personal and loan.status <= 1:
            return render(request, "chumaklending/loan.html",
{"error": "Ви не маєте доступу до цієї заявки на позику на дану
МИТЬ." })
        loan.photo = LoanPhoto.objects.filter(loan =
loan).order_by("id")[0]
        loan.recommends_count = loan.recommends.count()
        if not personal:
            comments = LoanComment.objects.filter(loan = loan,
personal = False).order_by("createdat")
        else:
            comments = LoanComment.objects.filter(loan =
loan).order_by("createdat")
        loan.comments_count = loan.comments.count()
        if borrower_user:
            if loan in borrower_user.loanbookmarks:
                loan.bookmark = True
            else:
                loan.bookmark = False

```

```

        if borrower_user in loan.recommends:
            loan.recommend = True
        else:
            loan.recommend = False

    payments =
LoanPayment.objects.filter(loaninvestment__loan = loan)
    if ordering == 0:
        payments = payments.order_by("fixeddata")
    elif ordering == 1:
        payments = payments.order_by("-fixeddata")
    elif ordering == 2:
        payments = payments.order_by("amount")
    elif ordering == 3:
        payments = payments.order_by("-amount")
    elif ordering == 4:
        payments = payments.order_by("planneddata")
    elif ordering == 5:
        payments = payments.order_by("-planneddata")
    loan.payments_count = payments.count()
    loanpaginator = Paginator(payments, 20)
    loanpage = request.GET.get("paypage")
    loan.payments = loanpaginator.get_page(loanpage)
    return render(request, "chumaklending/loan_pays.html",
{"loan": loan })

    def borrower_post_view(request, postid):
        borrower_user =
Borrower.objects.select_related("obligeeaccount",
"obligeeaccount__useraccount").filter(obligeeaccount__useraccoun
t = request.user)
        try:
            borrower_post =
BorrowerPost.objects.select_related("borrower",
"borrower__followers").filter(id = postid)
            borrower_post.borrower.followers_count =
borrower_post.borrower.followers.count()
        except BorrowerPost.DoesNotExist:
            borrower_post = None
            return render(request,
"chumaklending/borrowerpost.html", {"error": "Такої публікації
немає. Помилка заданого значення."})
            borrower_post.photos =
BorrowerPostPhoto.objects.filter(borrowerpost = borrower_post)
            comments =
BorrowerPostComment.objects.filter(borrowerpost = borrower_post)
            comments = comments.order_by("createdat")
            borrower_post.comments_count = comments.count()
            loanpaginator = Paginator(comments, 20)
            loanpage = request.GET.get("page")
            comments = loanpaginator.get_page(loanpage)
            borrower_post.comments = comments
            if borrower_user and borrower_user in
borrower_post.borrower.followers:

```



```

        for borrower_loan in followed_loans:
            if borrower_user in
borrower_loan.loan.recommends:
                borrower_loan.recommend = True
            else:
                borrower_loan.recommend = False
        return render(request, "chumaklending/foll.html",
{"loans": followed_loans, "posts": followed_posts })

def obligee_payments_view(request, obligee_user = None,
ordering = 0):
    if not obligee_user:
        obligee_user =
Obligee.objects.select_related("useraccount").prefetch_related("
loanbookmarks").filter(useraccount = request.user)
    else:
        try:
            obligee_user =
Obligee.objects.select_related("useraccount").prefetch_related("
loanbookmarks").filter(useraccount__username = obligee_user)
            render(request,
"chumaklending/obligee_invs.html", {"error": "Такого користувача
немає." })
        except Obligee.DoesNotExist:
            obligee_user = None
    payments =
LoanPayment.objects.select_related("loaninvestment",
"loaninvestment__loan").filter(loaninvestment__obligee =
obligee_user)
    if ordering == 0:
        payments = payments.order_by("fixeddata")
    elif ordering == 1:
        payments = payments.order_by("-fixeddata")
    elif ordering == 2:
        payments = payments.order_by("amount")
    elif ordering == 3:
        payments = payments.order_by("-amount")
    elif ordering == 4:
        payments = payments.order_by("planneddata")
    elif ordering == 5:
        payments = payments.order_by("-planneddata")
    obligee_user.payments_count = payments.count()
    loanpaginator = Paginator(payments, 20)
    loanpage = request.GET.get("paypage")
    payments = loanpaginator.get_page(loanpage)
    render(request, "chumaklending/obligee_pays.html",
{"payments": payments, "obligee": obligee_user })

def obligee_investments_view(request, obligee_user = None,
ordering = 0):
    if not obligee_user:

```

```

        obligee_user =
Obligee.objects.select_related("useraccount").prefetch_related("
loanbookmarks").filter(useraccount = request.user)
    else:
        try:
            obligee_user =
Obligee.objects.select_related("useraccount").prefetch_related("
loanbookmarks").filter(useraccount__username = obligee_user)
            render(request,
"chumaklending/obligee_invs.html", {"error": "Такого користувача
немає." })
        except Obligee.DoesNotExist:
            obligee_user = None
    investments =
LoanInvestment.objects.select_related("loaninvestment",
"loaninvestment__loan").filter(loaninvestment__obligee =
obligee_user)
    if ordering == 0:
        investments = investments.order_by("payedat")
    elif ordering == 1:
        investments = investments.order_by("-payedat")
    elif ordering == 2:
        investments = investments.order_by("amount")
    elif ordering == 3:
        investments = investments.order_by("-amount")
    obligee_user.investments_count = investments.count()
    loanpaginator = Paginator(investments, 20)
    loanpage = request.GET.get("invpage")
    loan.investments = loanpaginator.get_page(loanpage)
    return render(request,
"chumaklending/obligee_invs.html", {"investments": investments,
"obligee": obligee_user })

def obligee_page_view(request, obligee_user = None):
    if not obligee_user:
        obligee_user =
Obligee.objects.select_related("useraccount").prefetch_related("
loanbookmarks").filter(useraccount = request.user)
    else:
        try:
            obligee_user =
Obligee.objects.select_related("useraccount").prefetch_related("
loanbookmarks").filter(useraccount__username = obligee_user)
            render(request,
"chumaklending/obligee_invs.html", {"error": "Такого користувача
немає." })
        except Obligee.DoesNotExist:
            obligee_user = None
    investments =
LoanInvestment.objects.select_related("loaninvestment",
"loaninvestment__loan").filter(loaninvestment__obligee =
obligee_user)
    investments = investments.order_by("-payedat")[:10]

```

```

        payments =
LoanPayment.objects.select_related("loaninvestment",
"loaninvestment__loan").filter(loaninvestment__obligee =
obligee_user)
        payments = payments.order_by("-fixeddata")[:10]
        return render(request, "chumaklending/obligee.html",
{"obligee": obligee_user, "investments": investments,
"payments": payments })

    def bookmarked_loans_view(request):
        obligee_user =
Obligee.objects.select_related("useraccount").prefetch_related("
obligeeaccount__loanbookmarks").filter(obligeeaccount__useraccou
nt = request.user)
        bookmarked_loans =
Loan.objects.prefetch_related("recommends").filter(id__in =
borrower_user.obligeeaccount.loanbookmarks__id, loan__status__gt
= 1)
        bookmarked_loans = bookmarked_loans.order_by("-
addedat")
        loanpaginator = Paginator(bookmarked_loans, 20)
        loanpage = request.GET.get("page")
        bookmarked_loans = loanpaginator.get_page(loanpage)
        for borrower_loan in bookmarked_loans:
            borrower_loan.photo = LoanPhoto.objects.filter(loan
= borrower_loan.loan).order_by("id")[0]
            borrower_loan.recommends_count =
borrower_loan.loan.recommends.count()
            borrower_loan.comments_count =
LoanComment.objects.filter(loan = borrower_loan.loan).count()
            if obligee_user:
                for borrower_loan in bookmarked_loans:
                    if obligee_user in borrower_loan.recommends:
                        borrower_loan.recommend = True
                    else:
                        borrower_loan.recommend = False
        return render(request, "chumaklending/bookmarks.html",
{"loans": bookmarked_loans })

    def obligee_recommended_loans_view(request):
        obligee_user =
Obligee.objects.select_related("useraccount").prefetch_related("
obligeeaccount__loanbookmarks").filter(obligeeaccount__useraccou
nt = request.user)
        obligee_loans =
Loan.objects.prefetch_related("recommends").filter(recommends__i
n = [obligee_user], loan__status__gt = 1)
        obligee_loans = obligee_loans.order_by("-addedat")
        loanpaginator = Paginator(bookmarked_loans, 20)
        loanpage = request.GET.get("page")
        bookmarked_loans = loanpaginator.get_page(loanpage)
        for borrower_loan in obligee_loans:

```

```
        borrower_loan.photo = LoanPhoto.objects.filter(loan
= borrower_loan.loan).order_by("id")[0]
        borrower_loan.recommends_count =
borrower_loan.loan.recommends.count()
        borrower_loan.comments_count =
LoanComment.objects.filter(loan = borrower_loan.loan).count()
        return render(request, "chumaklending/loansoblig.html",
{"loans": obligee_loans})
```

ДОДАТОК В
Слайди презентації

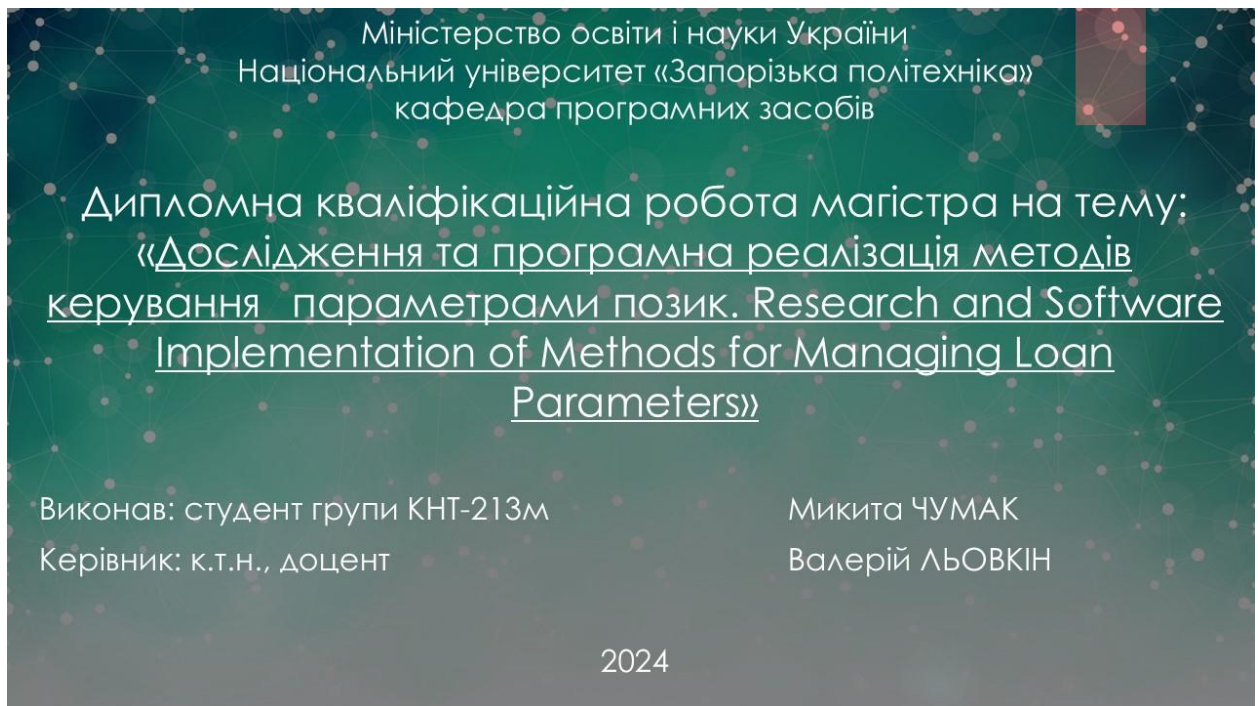


Рисунок В.1 – Початковий слайд

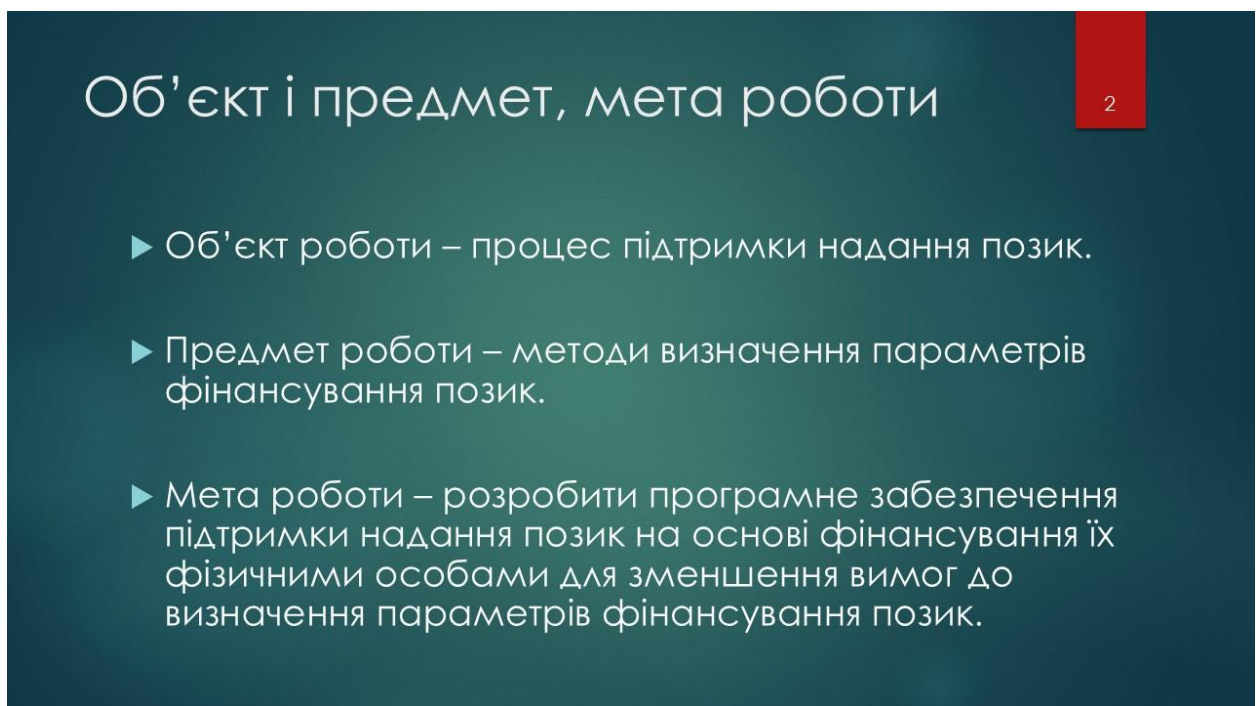


Рисунок В.2 – Об'єкт і предмет, мета роботи

Завдання роботи

3

- ▶ проаналізувати програми підтримки надання позик;
- ▶ розробити метод керування параметрами позик;
- ▶ виконати проєктування програми підтримки надання позик;
- ▶ реалізувати програму підтримки надання позик;
- ▶ виконати експериментальне дослідження прогнозування відсоткової ставки за позиками.

Рисунок В.3 – Завдання роботи

Порівняння вебзастосунку підтримки надання позик з прототипом

4

Показник	LendingClub	ChumakLending
Джерело фінансування позик	Власні кошти користувачів	Власні кошти користувачів
Типи позик	Індивідуальні, сумісні	Індивідуальні, сумісні
Можливості соціальної мережі для взаємодії з позичальниками	–	+
Можливість роботи користувачів з України	–	+
Програмне визначення відсоткової ставки за позиками під час формування запиту	–	+

Рисунок В.4 – Порівняння вебзастосунку підтримки надання позик з прототипом

Метод керування параметрами ПОЗИК

5

- ▶ підготовка вибірки даних для прогнозування відсоткової ставки за позикою, виконуючи перетворення назви посад в категорії з використанням 25 найпопулярніших та виконуючи нормування відсоткової ставки;
- ▶ створено модель прогнозування відсоткової ставки за позикою на основі використання ансамблів дерев рішень, застосовуючи для цього метод XGBoost;
- ▶ навчання створеної моделі на основі опрацьованої попередньо навчальної вибірки даних;
- ▶ формування екземпляра даних, для чого має бути визначено значення кожної вхідної ознаки, яка є відомою за позичальником, з перетворенням займаної посади позичальником на роботу у відповідну категорію на основі 25 найбільш популярних а для ознак, для яких можливе варіювання, має бути встановлено допустимий інтервал значень та крок зміни значень;
- ▶ на основі моделі прогнозування відсоткової ставки прогнозування відсоткової ставки для заданих фіксованих значень ознак позики та всіх варіантів змінних вхідних ознак в тих інтервалах, які були встановлені: тоді на виході має бути отримано значення відсоткової ставки для кожного такого варіанту, а зі всіх отриманих значень має бути обрано той, який є найбільш прийнятним.

Рисунок В.5 – Метод керування параметрами позик

Вибір засобів розробки програми підтримки надання ПОЗИК

6

Критерій	Django	Ruby on Rails
Мова програмування для розробки	Python	Ruby
Фреймворк веброзробки	Django	Ruby on Rails
Складність навчання та підтримки	Менша за рахунок високої читабельності коду	Вища за рахунок великої кількості концепцій
Час на встановлення рішення	Менший	Вищий
Бібліотеки машинного навчання	Більше засобів, краща підтримка, оригінальні версії	Засоби наявні, але нерідко представлені інтерфейсами до Python-рішень, гірша підтримка

Рисунок В.6 – Вибір засобів розробки програми підтримки надання позик

The diagram illustrates the use cases for a 'Credit and Loan' system, organized into three main functional areas: Credit, Loan, and Loan and Credit.

- Credit Section:** The actor 'Користувач-кредитор' (Credit User) is associated with 20 use cases:
 - пошук серед позик (search for loans)
 - перегляд сторінки користувача-позичальника (view borrower's page)
 - перегляд публікації користувача-позичальника (view borrower's publication)
 - підписування на публікації користувача-позичальника (subscribe to borrower's publication) - Extends: скасування підписки на публікації користувача-позичальника (cancel subscription to borrower's publication)
 - перегляд підписок (view subscriptions)
 - перегляд сторінки користувача-кредитора (view lender's page)
 - внесення даних про користувача-кредитора (enter data about lender)
 - перегляд матеріалів заявки на позику (view loan application materials)
 - перегляд заявок на позику користувача (view user's loan applications)
 - фінансування позик (loan financing)
 - перегляд профілювання користувачем позик (view user's loan profiling)
 - перегляд фінансування позик (view loan financing)
 - перегляд сторінок користувачів-платівець (view borrower's payment pages)
 - перегляд платежів за позикою (view loan payments)
 - комітування позик (loan commitment)
 - перегляд персональних повідомлень (view personal messages) - Extends: скасування закладки за позику (cancel bookmark for loan)
 - комітування публікації (publishing commitment) - Extends: скасування рекомендації позик (cancel loan recommendation)
 - створення закладки за позику (bookmark loan)
 - рекомендування позик (loan recommendation)
 - перегляд рекомендацій позик користувача (view user's loan recommendations)
 - перегляд рекомендацій за позикою (view loan recommendations)
 - перегляд позик у закладках (view loans in bookmarks)
 - перегляд власних коментарів (view own comments) - Extends: реєстрація (registration)
 - авторизація (authorization)
- Loan Section:** The actor 'Користувач-позичальник' (Borrower User) is associated with 6 use cases:
 - публікація матеріалу (material publication) - Extends: редагування публікації (editing publication)
 - внесення загальних даних про користувача-позичальника (enter general data about borrower)
 - розміщення заявки на позику (loan application placement) - Extends: скасування заявки на позику (cancel loan application)
 - внесення власних даних до спільної заявки на позику (enter own data into joint loan application) - Extends: редагування заявки на позику (editing loan application)
 - додавання позичальника для сумісної позик (adding borrower for joint loan)
- Loan and Credit Section:** The actor 'Представник сервісу надання позик' (Loan Service Representative) is associated with 14 use cases:
 - верифікація заявки на позику (loan application verification)
 - авторизація (authorization)
 - перегляд позик за станом (view loans by status)
 - пошук серед позик (search for loans)
 - перегляд сторінки користувача-позичальника (view borrower's page)
 - перегляд матеріалів заявки на позику (view loan application materials)
 - верифікація документів за позикою (loan document verification)
 - перегляд отриманих користувачем платежів (view payments received by user)
 - перегляд фінансування позик (view loan financing)
 - перегляд публікації користувача-позичальника (view borrower's publication)
 - перегляд платежів за позикою (view loan payments)
 - перегляд заявок на позики користувача (view user's loan applications)
 - перегляд сторінки користувача-кредитора (view lender's page)
 - перегляд всіх виконаних платежів (view all completed payments)
 - перегляд профілювання користувачем позик (view user's loan profiling)
 - перегляд всіх виконаних фінансування (view all completed financing)

ПОЗИК

8

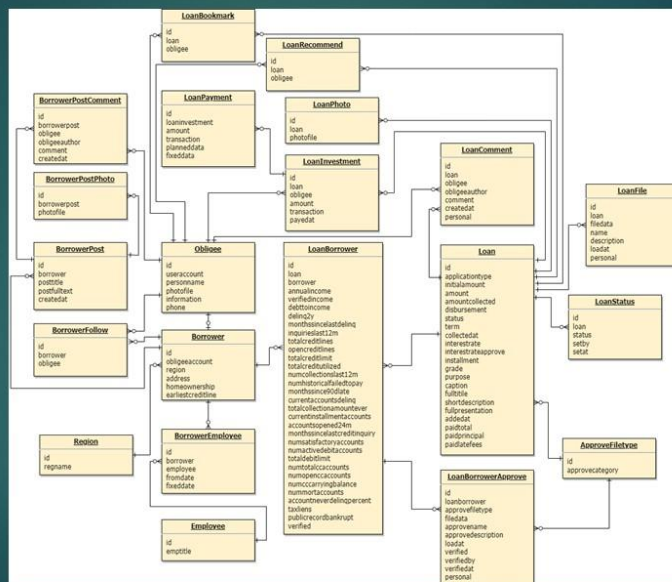


Рисунок В.8 – Схема БД

Діаграма станів позики

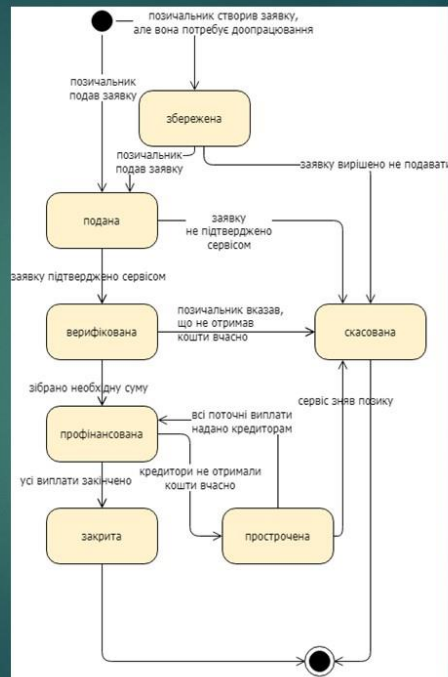


Рисунок В.9 – Діаграма станів позики

Рекомендовані користувачем-кредитором позики

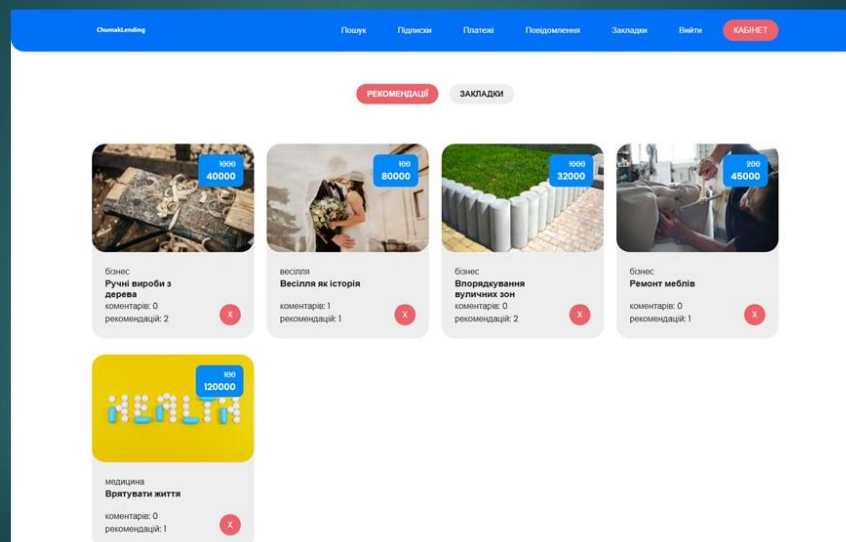


Рисунок В.10 – Рекомендовані користувачем-кредитором позики

Створення нової позики користувачем-позичальником

11

The screenshot shows a web form for creating a new loan. The title is 'РОЗМІЩЕННЯ НОВОЇ ЗАЯВКИ НА ПОЗИКУ'. The form includes a logo placeholder on the left and a series of input fields on the right. The fields are: 'Воротня позички' (Loan type), 'Головна позичка' (Main loan), 'Воротня позички' (Loan type), 'Тип застави' (Collateral type), 'Ідентифікаційний' (Identification), 'Додатковий позичка' (Additional loan), 'Мінімальна необхідна сума позички' (Minimum required loan amount), '207800', 'Базисна необхідна сума позички' (Base required loan amount), '207800', 'Додатковий кредит позички' (Additional credit loan), '0', 'Мінімальний допустимий строк позички, місяць' (Minimum acceptable loan term, month), '36', 'Максимальний допустимий строк позички, місяць' (Maximum acceptable loan term, month), '36', 'Додатковий кредит позички' (Additional credit loan), '0', and a 'Заявити' (Apply) button. Below the form, there is a note: '*Термін позички розраховується тільки тоді, коли всі дані вказані заповнені' (The loan term is calculated only when all data is filled in). Below this, it says 'Термін: 36 місяць, сума: 207800, відсоткова ставка: 8,72 %' (Term: 36 months, sum: 207800, interest rate: 8.72 %).

Рисунок В.11 – Створення нової позики користувачем-позичальником

Перегляд позик за станом представником сервісу надання позик

12

The screenshot shows a web page for reviewing loan status. The title is 'Перегляд позик за станом представником сервісу надання позик'. The page has a navigation bar with 'Позичальник', 'Позика', 'Позичальник', 'Фінансування', 'Відомості', and 'Відомості'. Below the navigation bar, there are filters for 'Стан' (Status) with a dropdown menu showing 'позика' (loan), 'Строк, місяць' (Term, month) with a dropdown menu showing '36', 'Мінімальна сума' (Minimum sum), 'Максимальна сума' (Maximum sum), 'Ціль' (Goal) with a dropdown menu showing '36', and 'Параметр' (Parameter) with a dropdown menu showing '36'. There is a 'Заявити' (Apply) button. Below the filters, there is a grid of loan cards. Each card shows a thumbnail image, a title, a description, a date, a count, and a recommendation button. The cards are: 'Позика: Надійшли запити позички' (Loan: Received loan requests), 'Позика: Тимчасовий ступінь' (Loan: Temporary degree), 'Позика: Рішення майстерів' (Loan: Master's decision), 'Позика: Швидкі рішення' (Loan: Fast decisions), 'Позика: Висхідні позички' (Loan: Rising loans), 'Позика: Підприємство Аліса' (Loan: Company Alice), 'Позика: Висхідні позички' (Loan: Rising loans), and 'Позика: Дітями позички' (Loan: Children's loans).

Рисунок В.12 – Перегляд позик за станом представником сервісу надання
ПОЗИК

Експериментальне дослідження

13

Метод	Середня абсолютна похибка	Коренева середньоквадратична похибка
Дерево рішень зі всіма вхідними ознаками	0,00306778	0,00406196
Метод k-найближчих сусідів зі всіма вхідними ознаками	0,00274482	0,00349476
XGBoost зі всіма вхідними ознаками	0,00218533	0,00288257
Дерево рішень з 40 вхідними ознаками	0,00305814	0,00404539
Метод k-найближчих сусідів з 40 вхідними ознаками	0,00274482	0,00349476
XGBoost з 40 вхідними ознаками	0,00218432	0,0028834
Дерево рішень з 39 вхідними ознаками	0,00303764	0,0040409
Метод k-найближчих сусідів з 39 вхідними ознаками	0,00212651	0,00280321
XGBoost з 39 вхідними ознаками	0,0021669	0,00285138
Дерево рішень з 35 вхідними ознаками	0,00304254	0,00404128
Метод k-найближчих сусідів з 35 вхідними ознаками	0,00274482	0,00349476
XGBoost з 35 вхідними ознаками	0,00218698	0,00289246
Дерево рішень з 20 вхідними ознаками	0,0030936	0,00407879
Метод k-найближчих сусідів з 20 вхідними ознаками	0,00272249	0,0034802
XGBoost з 20 вхідними ознаками	0,00221248	0,00289896
Дерево рішень з 10 вхідними ознаками	0,00314124	0,00419205
Метод k-найближчих сусідів з 10 вхідними ознаками	0,00274392	0,0034831
XGBoost з 10 вхідними ознаками	0,00234291	0,00305467

Рисунок В.13 – Експериментальне дослідження

Висновки

14

- У результаті виконання дипломної кваліфікаційної роботи магістра розроблено програмне забезпечення підтримки надання позик на основі фінансування їх фізичними особами для зменшення вимог до визначення параметрів фінансування позик. Для роботи над програмним проектом використано мову Python, бібліотеку XGBoost, фреймворк Django, пакети Numpy, Pandas, Scikit-learn, систему PostgreSQL.
- Наукова новизна роботи полягає в методі керування параметрами позик, який характеризується використанням дерев рішень на основі створення ансамблів методом XGBoost для прогнозування відсоткової ставки позики за значеннями вхідних ознак, що дозволяє встановлювати варіативні значення параметрів позик таким чином, щоб сприяти результативності процесу збору коштів від фізичних осіб.

Рисунок В.14 – Висновки