

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ОБМІНУ ДИТЯЧИМИ ІГРАШКАМИ
KIDS TOY EXCHANGE SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-129сп
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Голубєв І.К.

(прізвище та ініціали)

Керівник Голуб Т.В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Голубєва Івана Костянтиновича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення обміну дитячими іграшками
Kids Toy Exchange Software

керівник проєкту (роботи) Голуб Тетяна Василівна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми.
3. Розробка програми. 4. Експлуатація та тестування 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
1-4 Основна частина	Голуб Т.В., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання «4» квітня 2022 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	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

(підпис)

Голубєв І.К.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Голуб Т.В.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 82 с., 23 рис., 1 табл., 3 дод., 11 джерел.

ДИТЯЧА ІГРАШКА, ОБМІН РЕЧИМА, ВРАЖЕННЯ, БАЗА ДАНИХ, PYTHON.

Об'єктом роботи є процес обміну дитячими іграшками між людьми. Предметом роботи є програмне забезпечення обміну дитячими іграшками. Мета роботи – розробити програмне забезпечення обміну дитячими іграшками для зменшення витрат людей.

Матеріали, методи та технічні засоби: мова програмування Python, мультипарадигмальне програмування, вебфреймворк Django, шаблон проєктування Model-View-Template, система керування базами даних MySQL.

Результати. Проведено весь процес розробки програми, включаючи аналіз проблеми, визначення і моделювання вимог до програми, проєктування бази даних, розробку програми за допомогою фреймворку Django, що призвело до отримання відповідного практичного інструменту.

Висновки. Виконана розробка представлена програмою, що призначена для підтримки процесу обміну власних дитячих іграшок на інші дитячі іграшки з забезпеченням збереження історії стану кожної дитячої іграшки і вражень користувачів та наданням доступу до такої історії. Це дозволяє батькам уникнути фінансових витрат у випадку їх неможливості або зменшити такі витрати за потреби у процесі розвитку дитини, використовуючи наявну в сім'ї базу іграшок.

Галузь використання. Обмін дитячими іграшками як потреба забезпечення дитині нового досвіду і розваг без необхідності додаткових витрат.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 82 pages, 23 figures, 1 table, 3 appendixes, 11 sources.

The object of the work is the process of exchanging children's toys between people. The subject of the work is the software for exchanging children's toys. The purpose of the work is to develop software for exchanging children's toys to reduce people's costs.

Materials, methods and technical means: Python programming language, multi-paradigm programming, Django web framework, Model-View-Template design template, MySQL database management system.

Results. The whole process of program development was carried out, including problem analysis, definition and modeling of program requirements, database design, program development using Django framework, which led to the acquisition of the appropriate practical tool.

Conclusions. The completed development is represented by a program designed to support the process of exchanging own children's toys for other children's toys while ensuring the preservation of the history of each children's toy and user impressions and providing access to such history. It allows parents to avoid financial costs if they are unable to do so, or reduce such costs as needed during the child's development, using the family's existing toy database.

Field of use. Sharing children's toys as a need to provide children with new experience and entertainment without the need for additional costs.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	7
Вступ	8
1 Аналіз предметної області	9
1.1 Актуальність обміну дитячими іграшками	9
1.2 Аналіз програмних аналогів	11
1.3 Висновки за розділом 1	13
2 Розробка архітектури програми	15
2.1 Вибір мови розробки програми	15
2.2 Функціональні вимоги до програмного забезпечення	15
2.3 Проектування бази даних	19
2.4 Висновки за розділом 2	29
3 Розробка програми	30
3.1 Послідовність роботи	30
3.2 Опис реалізованих класів програми	31
3.3 Висновки за розділом 3	37
4 Експлуатація та тестування програми	39
4.1 Призначення програми	39
4.2 Умови виконання програми	39
4.3 Тестування і опис принципів експлуатації програми	39
4.4 Висновки за розділом 4	45
Висновки	46
Перелік джерел посилання	47
Додаток А Технічне завдання	49
Додаток Б Текст програми	53
Додаток В Слайди презентації	76

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVT – Model-View-Template;

БД – база даних.

ВСТУП

Іграшки відіграють важливу роль у розвитку розуму дітей, покращуючи їхній процес мислення та здатність приймати рішення, при цьому вони складаються з різних категорій (таких як тренування мозку, фізична активність, рекламні товари та предмети колекціонування), однак необхідно розуміти, що іграшки є не просто розвагою, а є цілим ринком, при чому достатньо вивченим на даний момент:

- обсяг світового ринку іграшок у 2020 році становив 129,45 мільярдів доларів США;

- глобальний вплив COVID-19 був безпрецедентним і приголомшливим, і на тлі пандемії цей продукт зазнав позитивного шоку попиту в усіх регіонах: світовий ринок продемонстрував зіркове зростання на 22,30 % у 2020 році, на 141,08 мільярдів доларів США в 2021 році з прогнозованим збільшенням до 230,64 мільярда доларів США до 2028 року при середньому зростанні 7,30% у період 2022-2028 років [1].

Одночасно потрібно розуміти і інший бік даного процесу. Зі збільшенням продажу іграшок збільшується кількість їх у вжитку. З часом діти виростають, і частина іграшок в процесі стає непотрібною, адже призначена для меншої вікової категорії, інша частина стає непотрібною на наступних етапах. Але з часом переважна кількість іграшок виявляється непотрібною. Весь цей ринок збільшується, а відповідно і кількість таких іграшок збільшується. Потрібно розуміти, що цей ринок можна використати, якщо дивитися на нього як на ринок вживаних дитячих іграшок. Відповідно даний ринок потребує підтримки, яку може здійснити розробка програмного забезпечення обміну дитячими іграшками.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність обміну дитячими іграшками

У 2020 році п'ятьма європейськими ринками з найвищим доходом від іграшок були:

- Великобританія з доходом близько 3,8 мільярдів євро;
- Німеччина з доходом близько 3,6 млрд євро;
- Франція з доходом близько 3,3 млрд євро;
- Італія з доходом близько 1,2 млрд євро;
- Іспанія з доходом близько 0,9 млрд євро [2].

У Сполучених Штатах діти отримують іграшки на суму понад 6500 доларів США у віці від двох до 12 років, а наприклад, в Австралії індустрія іграшок коштує понад 3,7 мільярда доларів на рік, а зростання протягом останніх років, яке було розглянуто раніше, відбувається швидше, ніж загальний сектор онлайн-роздрібної торгівлі, при цьому є також сезонні коливання на ринку: кількість іграшок збільшується, коли починається різдвяний сезон [3].

Крім екологічних проблем, наявність великої кількості іграшок може негативно вплинути на дітей, а також на батьків: приміщення з великою кількістю іграшок надто стимулює і впливає на здатність немовлят, малюків і дітей молодшого віку вчитися та творчо грати, тому подібно до захащених комор або офісних приміщень, які заважають дорослим зосередитися, занадто велика кількість іграшок у домі може ускладнити дітям концентрацію уваги, навчання та розвиток важливих навичок під час гри, дослідження показують, що менша кількість іграшок за раз призводить до кращої якості гри для малюків, що дозволяє їм зосередитися на одній іграшці за раз, розвивати навички концентрації та грати більш творчо, інша проблема з великою кількістю іграшок полягає в тому, що ми схильні приділяти їм менше значення, а тому зменшивши кількість іграшок, дорослі можуть допомогти дітям розвинути вдячність [3].

Отож це дозволяє в підсумку говорити не тільки про величезний ринок дитячих іграшок, але і про те, що цей ринок пов'язаний з великою кількістю сміття або іграшок, які потребують повторного використання. І перший, і другий варіант пов'язані з наступними проблемами:

- запобігання екологічній катастрофі у світі;
- зменшення власних витрат на покупку нових іграшок.

Ці два ракурси пов'язані безпосередньо з тим, що робити з наявними іграшками, і як отримати нові іграшки для дітей.

Одним зі шляхів вирішення обох цих проблем є обмін дитячими іграшками. При чому найбільш цікавим є обмін без участі грошей, тобто коли люди знаходять іграшки, які їм цікаві, і віддають замість них власні іграшки. Це є підходом, який найбільше сприяє екології, підтримці ринку в зваженому стані.

Загалом ринок іграшок можна розділити на такі сегменти:

- ігри та головоломки;
- іграшки для відпочинку та спорту;
- іграшки для немовлят, малюків та дошкільнят;
- ляльки;
- конструктори;
- фігурки та аксесуари;
- молодіжна електроніка;
- транспортні засоби;
- дослідницькі та інші іграшки;
- плюшеві іграшки [2].

Однак, це тільки один з підходів до категоризації іграшок. Наприклад, можлива наступна категоризація:

- маніпулятивні або функціональні іграшки – до них належать іграшки-конструкції та будівельні іграшки, головоломки, іграшки для перевдягання, намистини, кубики, іграшки для ванни, іграшки з піском та водою: маніпулятивні іграшки важливі для розвитку дрібної та великої

моторики, спритності та координації, які є життєво важливими для малювання, письма, одягання тощо;

– активні іграшки – в тому числі різні іграшки на вулиці, спорядження для скелелазіння, спортивне спорядження та іграшки для катання: активні іграшки чудово підходять для загальної фізичної активності та розвитку моторики;

– навчальні іграшки – до них належать настільні та карткові ігри, книги та іграшки для спеціальних навичок, такі як ідентифікація літер, сортувальники форми та кольору;

– творчі іграшки – наприклад, матеріали для мистецтва та рукоділля, музичні іграшки та інструменти, включаючи цифрову музику та програми для малювання;

– іграшки типу «зроби і повір», включаючи наряди та рольові ігри (костюми, одяг, капелюхи, маски та аксесуари), м'які іграшки, ляльки, ляльки, транспортні іграшки [3].

1.2 Аналіз програмних аналогів

Варто відзначити, що прямого популярного аналога програма немає, проте є рішення, з якими її можна порівняти.

Etsy [4] є відомою платформою загалом для продажу виробів ручної роботи. Проте вона має підрозділ і для продажу повторно використовуваних виробів. Таким чином можна знайти дитячі іграшки, які вже використовувались раніше (рис. 1.1). Зокрема вони можуть бути певним чином оброблені і продаватися в новому вигляді. Це є особливістю даного вебсайту. При цьому варто відзначити, що на вебсайті оперують магазини, зокрема і відгуки залишаються магазинам, формуючи їх загальний рейтинг. До того ж варто зазначити, що у випадку даного вебсайту мова йде саме про продаж іграшок, а не про їх обмін, тобто дане рішення не є прямим аналогом програми, яка розробляється.

Etsy Search for anything  Sign in  

Jewelry & Accessories Clothing & Shoes Home & Living Wedding & Party Toys & Entertainment Art & Collectibles Craft Supplies & Tools Vintage

← Back to search results




MistyMorningDolls
37 sales | ★★★★★

♡ OOAK Upcycled, Repainted, Restyled Second Hand Bratz Doll, by UK Artist Jana Forsyth, #0041

➤ **USD 104.22** Only 1 available
Local taxes included (where applicable)

Add to cart

 Rare find — this item is hard to come by.

Highlights ^

- Handmade
- Materials: secondhand doll, ...

Description ^

OOAK (one-of-a-kind) second hand doll that has been restyled and given new personality and character by artist Jana Forsyth. Dolls don't have names intentionally as I think that is a privilege of a new owner to give them one :)

19 shop reviews ★★★★★ Sort by: Recommended ▼

 [Mike and Larissa](#) Apr 6, 2022

★★★★★

Misty Morning Dolls are in a league of their own! Best repaint I've ever seen! Jana is super friendly and helpful :) My...

Purchased item:

 [OOAK Upcycled, Repainted, Restyled S...](#)

 Helpful?

Рисунок 1.1 – Програма Etsy

Ще одним достатньо цікавим рішенням у даній сфері є вебсайт ToyExchangeClub [5] (рис. 1.2). На даному вебсайті виконується як раз обмін дитячими іграшками. Проте це не прямий обмін між власниками. Вебсайт пропонує оформити одну з підписок на один з пакетів іграшок, за яким буде отримано в результаті конкретні іграшки. Це дозволяє диференціювати іграшки в підсумку, проте безпосередньо сім'я не може сама визначити, які

власні іграшки на які саме інші іграшки вона хотіла б поміняти. Що знижує персоналізацію і знижує потенційну зацікавленість у даному процесі.

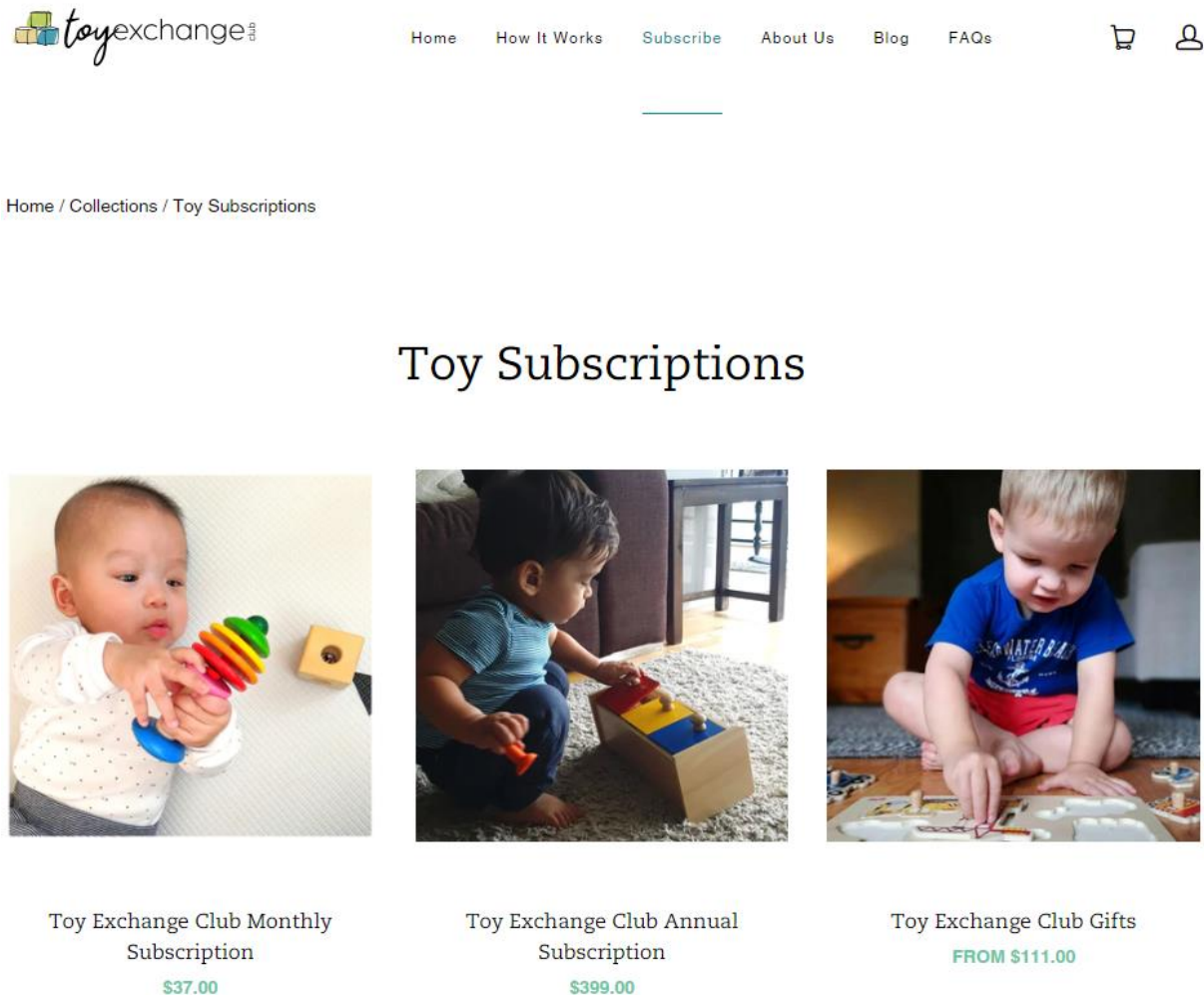


Рисунок 1.2 – Програма ToyExchangeClub

Тому в результаті потрібно зазначити, що безпосередньо програмне забезпечення обміну дитячими іграшками не має прямих аналогів, які б дозволяли сім'ям міняти власні іграшки між собою без використання фінансової складової. Саме на це і має бути направлена розробка.

1.3 Висновки за розділом 1

Розглянуто актуальність обміну дитячих іграшок, питання категоризації дитячих іграшок та стан ринку дитячих іграшок на даний

момент. Проведено аналіз програмних аналогів, серед яких безпосередньо не виявлено повної відповідності програмі, розробка якої має бути виконана в роботі.

Для виконання розробки програми було встановлено наступні завдання в роботі:

- дослідження проблеми, що було реалізовано в даному розділі;
- визначення вимог до розробки;
- проєктування програмного забезпечення обміну дитячими іграшками;
- програмна реалізація програмного забезпечення обміну дитячими іграшками.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір мови розробки програми

Розробка вебзастосунків може виконуватися різними мовами програмування. С# і Python [6]-[7] є одними з найбільш розповсюджених у даному ракурсі, до того ж активно використовуються для розробки великих систем.

За результатами порівняння мов (табл. 2.1) переважив Python за рахунок власної динамічної типізації та високої придатності для швидкої розробки програми, що дозволяє створювати в результаті якісні програми, що є загалом характеристикою обох мов, з меншими витратами часу.

Таблиця 2.1 – Вибір мови розробки програми

Критерій	С#	Python
Фреймворк розробки вебзастосунків	ASP.NET	Django, Flask
Типізація	Статична	Динамічна
Придатність для швидкої розробки	Середня	Висока
Відкрита ліцензія	Так	Так

2.2 Функціональні вимоги до програмного забезпечення

Функції, які має виконувати програма, охоплюють:

- ознайомлення з дитячою іграшкою;
- додавання дитячої іграшки;
- редагування даних дитячої іграшки;
- відображення виставлених на обмін власних дитячих іграшок;
- відображення оголошень про обмін за дитячою іграшкою;

- відображення вражень про дитячу іграшку;
- відображення пошкоджень дитячої іграшки;
- ознайомлення з оголошенням про обмін дитячої іграшки;
- пошук оголошень про обмін дитячих іграшок;
- додавання оголошення про обмін дитячої іграшки;
- редагування оголошення про обмін дитячої іграшки;
- пошук оголошень, що відповідають наявній у користувача дитячій іграшці;
- пошук оголошень, що відповідають власному оголошенню користувача;
- відображення оголошень користувача про обмін дитячих іграшок;
- додавання моделей дитячих іграшок;
- відображення дитячих іграшок заданої моделі;
- редагування моделі дитячої іграшки;
- відображення оголошень за заданою дитячою іграшкою;
- відображення оголошень про обмін дитячих іграшок заданої категорії;
- відображення списку категорій дитячих іграшок;
- додавання вимоги до оголошення;
- редагування вимоги в оголошенні;
- додавання пошкодження дитячої іграшки;
- редагування даних пошкодження дитячої іграшки;
- відображення моделей дитячих іграшок за брендом;
- надсилання повідомлень користувачу;
- відображення отриманих повідомлень від користувачів;
- висунення пропозиції про обмін дитячої іграшки на оголошення;
- підтвердження пропозиції про обмін дитячої іграшки;
- додавання враження;
- авторизація і реєстрація користувачів.

Усі інші роботи з внесення базових даних, як то категорій дитячих іграшок, передбачається виконувати через адміністративну панель Django адміністратором, а оскільки це стандартний інструмент і дані дії передбачається виконувати можливо раз за весь життєвий цикл програми, то їх окремо не було винесено. Тому моделювання відбувалось для двох типів користувачів: неавторизованих (рис. 2.1) і авторизованих користувачів. Сценарії роботи авторизованого користувача повністю включають сценарії роботи неавторизованого користувача і розширюють їх.

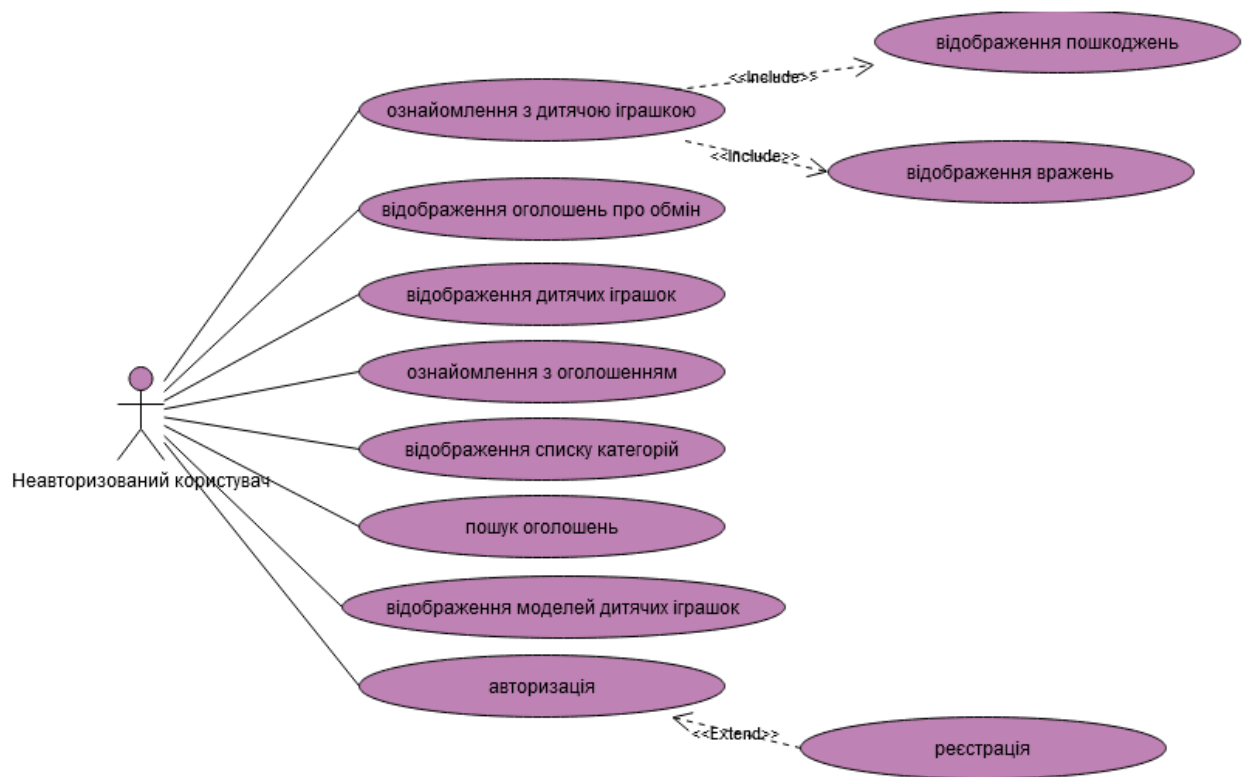


Рисунок 2.1 – Діаграма варіантів використання системи неавторизованим користувачем

Відповідно сценарії роботи неавторизованого користувача:

- ознайомлення з дитячою іграшкою;
- відображення оголошень про обмін за дитячою іграшкою;
- відображення вражень про дитячу іграшку;
- відображення пошкоджень дитячої іграшки;
- ознайомлення з оголошенням про обмін дитячої іграшки;

- пошук оголошень про обмін дитячих іграшок;
- відображення дитячих іграшок заданої моделі;
- відображення оголошень про обмін дитячих іграшок заданої категорії;

- відображення списку категорій дитячих іграшок;
- відображення моделей дитячих іграшок за брендом;
- авторизація користувачів.

У випадку відсутності у користувача облікового запису реалізується альтернативний сценарій до авторизації – реєстрація.

Відповідно сценарії роботи авторизованого користувача (рис. 2.2):

- додавання дитячої іграшки;
- відображення виставлених на обмін власних дитячих іграшок;
- додавання оголошення про обмін дитячої іграшки;
- відображення власних оголошень користувача про обмін дитячих іграшок, що включає керування пропозиціями на оголошення;
- надсилання повідомлень користувачу;
- відображення отриманих повідомлень від користувачів;
- пошук оголошень, що відповідають наявній у користувача дитячій іграшці;

- пошук оголошень, що відповідають власному оголошенню користувача;

- керування моделями дитячих іграшок;
- керування пошкодженнями дитячої іграшки;
- додавання враження;
- висунення пропозиції про обмін дитячих іграшок.

Альтернативними сценаріями є:

- редагування даних дитячої іграшки є альтернативним сценарієм під час реалізації сценарію відображення виставлених на обмін власних дитячих іграшок, якщо користувач хоче внести зміни до даних однієї з іграшок, які стосувались його оголошень;

– редагування оголошення про обмін дитячої іграшки є альтернативним сценарієм під час реалізації відображення власних оголошень, якщо користувач хоче внести зміни в одне з оголошень.



Рисунок 2.2 – Діаграма варіантів використання системи авторизованим користувачем

2.3 Проєктування бази даних

У даному застосунку база даних (БД) є одним з важливих структурних елементів, адже виконання практично всіх функцій програми пов'язано зі взаємодією з даними, а відповідно вимагає правильної їх структуризації. В якості основи використовувався сервер MySQL [8], що дозволило продовжити логіку розробки програми без додаткових витрат, при цьому застосовуючи абсолютно актуальний інструмент.

Сутності, виділені в БД:

- категорія іграшок;
- бренд дитячих іграшок;
- дитяча іграшка;
- модель дитячої іграшки;
- матеріал, з якого виготовлена іграшка;
- пошкодження у конкретної іграшки;
- оголошення про обмін;
- пропозиція про обмін, яку висуває інший користувач на оголошення;
- обмін – підтверджена пропозиція про обмін автором оголошення;
- повідомлення;
- враження користувача;
- користувач.

Таблиця `toyexchanger_category` представляє записи про категорії іграшок, включаючи поля:

- `id` – номер категорії дитячих іграшок;
- `ctitle` – назва категорії дитячих іграшок;
- `about` – детальний опис категорії дитячих іграшок;
- `catpic` – шлях до файлу фотографії, яка відповідає категорії дитячої іграшки.

Таблиця `toyexchanger_toy` представляє записи про самі конкретні дитячі іграшки, включаючи поля:

- `id` – номер дитячої іграшки;
- `totitle` – назва дитячої іграшки;
- `about` – опис дитячої іграшки;
- `category_id` – категорія, до якої належить дитяча іграшка;
- `startrecomage` – мінімальний рекомендований вік дитини;
- `endrecomage` – максимальний рекомендований вік дитини;
- `startage` – мінімальний вік, починаючи з якого дитина може грати з іграшкою;

– gender – стать дитини, для якої створена дитяча іграшка (може бути не визначено);

– toymodel_id – модель дитячої іграшки;

– is_certified – сертифікована дитяча іграшка або не сертифікована;

– produced – рік виробництва іграшки;

– country – країна виробництва іграшки;

– color – колір іграшки;

– toypic – шлях до папки з файлами фотографій дитячої іграшки.

Таблиці `toyexchanger_toymodel` і `toyexchanger_toy` мають зв'язок типу один до багатьох, бо одна модель може мати багато пов'язаних з нею дитячих іграшок, а одна іграшка може стосуватися однієї моделі.

Таблиці `toyexchanger_category` і `toyexchanger_toy` мають зв'язок типу один до багатьох, бо одна категорія може мати багато дитячих іграшок, а одна іграшка може стосуватися однієї категорії.

Таблиця `toyexchanger_brand` представляє записи про бренди виробників дитячих іграшок, включаючи поля:

– id – номер бренда дитячих іграшок;

– ctitle – назва бренда дитячих іграшок;

– about – детальний опис бренда дитячих іграшок.

Таблиця `toyexchanger_toymodel` представляє записи про моделі дитячих іграшок, включаючи поля:

– id – номер моделі дитячої іграшки;

– modtitle – назва моделі дитячої іграшки;

– about – детальний опис моделі дитячої іграшки;

– brand_id – бренд виробника моделі дитячої іграшки;

– point – середня оцінка моделі дитячої іграшки.

Таблиці `toyexchanger_brand` і `toyexchanger_toymodel` мають зв'язок типу один до багатьох, бо один бренд може мати багато моделей дитячих іграшок, а одна модель дитячої іграшки може вироблятися тільки під одним брендом.

Таблиця `toyexchanger_material` представляє записи про матеріали, з яких виробляються дитячі іграшки, включаючи поля:

- `id` – номер матеріалу;
- `mattitle` – назва матеріалу;
- `natural` – натуральний матеріал або штучний.

Таблиця `toyexchanger_toy_material` представляє записи про матеріали, з яких зроблено дитячу іграшку, включаючи поля:

- `id` – номер запису;
- `toy_id` – дитяча іграшка;
- `material_id` – матеріал дитячої іграшки.

Таблиця `toyexchanger_toy_material` реалізує зв'язок типу багато до багатьох між таблицями `toyexchanger_toy` і `toyexchanger_material`, оскільки кожна дитяча іграшка може складатися з різних матеріалів, а кожен матеріал може бути складовою різних дитячих іграшок.

Таблиця `toyexchanger_opinion` представляє записи про враження користувачів про дитячі іграшки, включаючи поля:

- `id` – номер враження;
- `exchangeruser_id` – користувач-автор враження;
- `created` – момент публікації враження;
- `point` – оцінка, виставлена загалом даним дитячій іграшці (не екземпляру, а іграшці загалом);
- `statepoint` – оцінка стану конкретної дитячої іграшки (конкретному екземпляру);
- `toy_id` – іграшка, про яку створено враження;
- `detailopinion` – повне враження (текст).

Таблиці `toyexchanger_toy` і `toyexchanger_opinion` мають зв'язок типу один до багатьох, бо одна дитяча іграшка може потенційно за час існування програми мати багато вражень, а кожне враження може стосуватися тільки однієї дитячої іграшки.

Таблиці `toyexchanger_exchangeruser` і `toyexchanger_opinion` мають зв'язок типу один до багатьох, бо один користувач може мати багато вражень, а кожне враження може бути створено тільки одним користувачем.

Таблиця `toyexchanger_damage` представляє записи про пошкодження іграшки, включаючи поля:

- `id` – номер пошкодження дитячої іграшки;
- `about` – опис пошкодження;
- `toy_id` – дитяча іграшка, до якої належить пошкодження;
- `author_id` – користувач, який створив запис про пошкодження;
- `created` – момент створення запису про пошкодження;
- `approved` – кількість підтверджень даного пошкодження;
- `unapproved` – кількість незгод з пошкодженням.

Таблиці `toyexchanger_toy` і `toyexchanger_damage` мають зв'язок типу один до багатьох, бо одна дитяча іграшка може мати багато пошкоджень, а кожне пошкодження описує тільки одну дитячу іграшку.

Таблиці `toyexchanger_exchangeruser` і `toyexchanger_damage` мають зв'язок типу один до багатьох, бо один користувач може створити записи про багато пошкоджень, а кожне пошкодження може бути записано тільки одним користувачем.

Таблиця `toyexchanger_exchangeruser` представляє користувача, включаючи поля:

- `id` – номер запису користувача;
- `username` – логін запису користувача;
- `first_name` – ім'я користувача;
- `last_name` – прізвище користувача;
- `email` – електронна пошта користувача;
- `password` – пароль користувача;
- `is_staff` – можливість доступу до адміністративної панелі або відсутність такої можливості;
- `is_active` – активність запису користувача або відсутність активності;

- `is_superuser` – можливість доступу до всіх можливостей або відсутність такої можливості;

- `last_login` – момент останньої авторизації користувача;

- `date_joined` – момент створення запису користувача;

- `about` – опис користувача.

Таблиця `toyexchanger_statement` представляє оголошення щодо обміну власної дитячої іграшки, включаючи поля:

- `id` – номер оголошення щодо обміну власної дитячої іграшки;

- `exchangeruser_id` – користувач, який сформував оголошення;

- `toy_id` – дитяча іграшка користувача, яку він хоче обміняти;

- `created` – момент створення оголошення;

- `is_active` – активність оголошення або неактивність;

- `together` – необхідність задоволення всіх вимог оголошення одночасно;

- `city` – місто, в якому знаходиться іграшка для обміну.

Таблиці `toyexchanger_exchangeruser` і `toyexchanger_statement` мають зв'язок типу один до багатьох, бо один користувач може мати багато оголошень, а кожне оголошення може бути створено тільки одним користувачем.

Таблиці `toyexchanger_toy` і `toyexchanger_statement` мають зв'язок типу один до багатьох, бо одна дитяча іграшка потенційно за час свого існування і існування програми може мати багато оголошень про обмін, а кожне оголошення стосується тільки однієї дитячої іграшки.

Таблиця `toyexchanger_statement_content` представляє вміст оголошення стосовно іграшок, на які власник хоче поміняти свою іграшку, включаючи поля:

- `id` – номер запису про вміст оголошення;

- `statement_id` – оголошення про обмін дитячої іграшки, складовою якого є цей запис;

– category_id – категорія дитячих іграшок, на які хоче власник поміняти свою іграшку;

– age – вік дитини, для якої потрібна іграшка;

– gender – стать дитини, для якої потрібна іграшка;

– is_certified – наявність сертифікату іграшки;

– startproduced – мінімальний рік виробництва іграшки;

– endproduced – максимальний рік виробництва іграшки;

– country – країна виробництва іграшки;

– color – колір іграшки;

– startpoint – мінімальна середня оцінка іграшки;

– endpoint – максимальна середня оцінка іграшки;

– amount – кількість іграшок такої категорії (якщо потрібно);

– about – опис особливостей іграшок цієї категорії.

Таблиці toyexchanger_statement і toyexchanger_statement_content мають зв'язок типу один до багатьох, бо одне оголошення може мати багато вимог, а кожна вимога стосується тільки одного оголошення про обмін дитячої іграшки.

Таблиці toyexchanger_category і toyexchanger_statement_content мають зв'язок типу один до багатьох, бо одна категорія дитячих іграшок може бути використана в різних вимогах до оголошень, а кожна вимога може містити тільки одну категорію.

Таблиця toyexchanger_statement_content_brand представляє записи про бренди з вимоги з оголошення, включаючи поля:

– id – номер запису;

– brand_id – бренд дитячої іграшки;

– statement_content_id – вимога оголошення.

Таблиця toyexchanger_statement_content_material представляє запис про матеріал іграшки, на яку готовий обміняти власник оголошення свою іграшку, включаючи поля:

- id – номер запису про матеріал іграшки для обміну іграшки з оголошення;
- statement_content_id – категорія дитячої іграшки, на яку власник хоче поміняти свою іграшку;
- material_id – матеріал дитячої іграшки.

Таблиця `toyexchanger_statement_content_material` реалізує зв'язок типу багато до багатьох між таблицями `toyexchanger_statement_content` і `toyexchanger_material`, оскільки кожна вимога до оголошення може містити багато матеріалів, з яких виготовляються іграшки, а кожен матеріал може бути складовою різних вимог.

Таблиця `toyexchanger_proposition` (рис. 2.3) представляє пропозицію про обмін за оголошенням, включаючи поля:

- id – номер пропозиції;
- statement_id – оголошення, до якого стосується пропозиція;
- exchangeruser_id – користувач, який висунув пропозицію;
- created – момент створення пропозиції;
- is_approved – підтвердження власником оголошення пропозиції або скасування;
- comment – опис деталей пропозиції;
- city – місто, в якому знаходяться іграшки пропоновані для обміну.

Таблиці `toyexchanger_statement` і `toyexchanger_proposition` мають зв'язок типу один до багатьох, бо одне оголошення може мати багато пропозицій про обмін, а кожна пропозиція може стосуватися тільки одного оголошення.

Таблиці `toyexchanger_exchangeruser` і `toyexchanger_proposition` мають зв'язок типу один до багатьох, бо один користувач може мати багато пропозицій про обмін дитячими іграшками, а пропозиція визначається тільки одним користувачем.

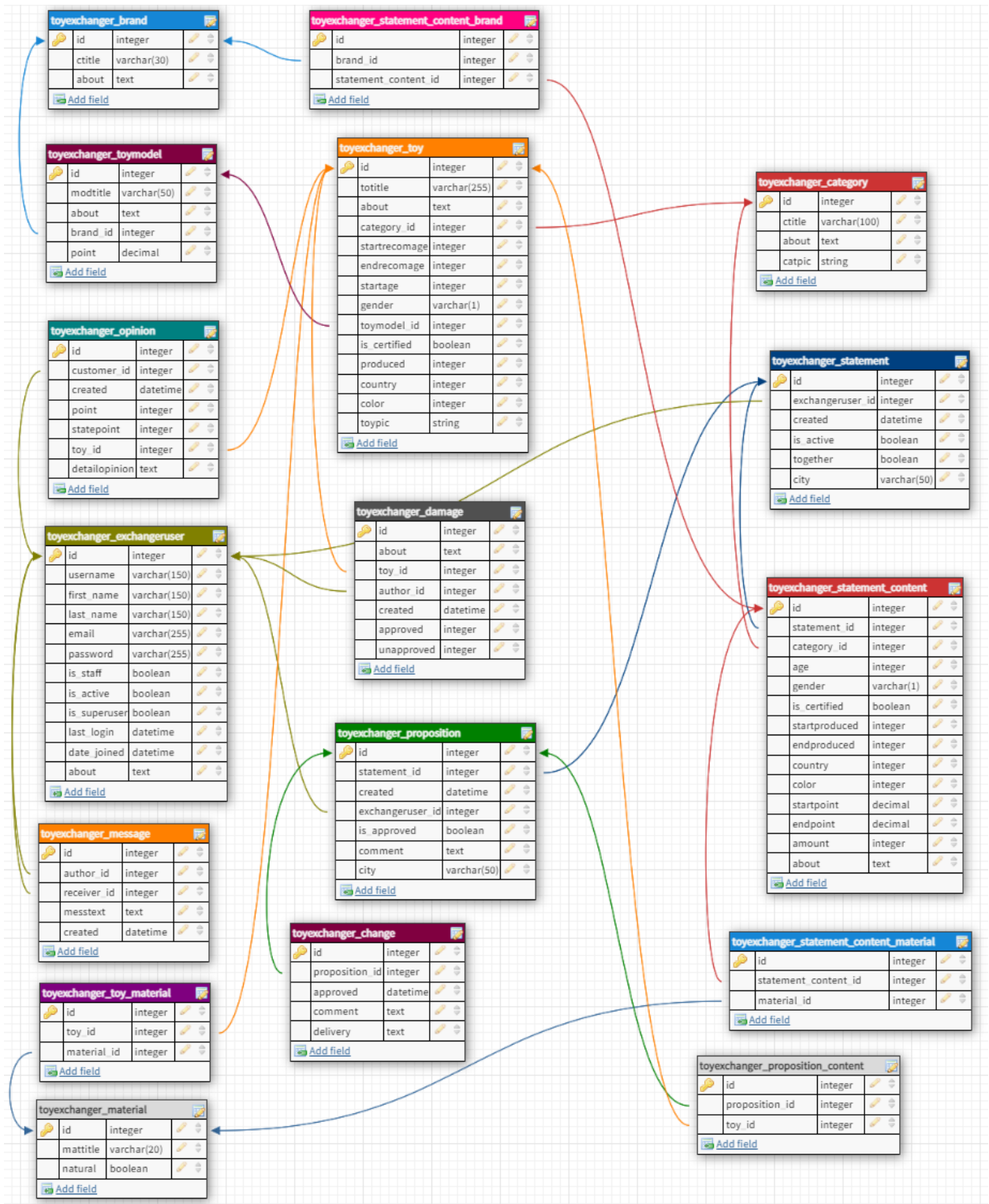


Рисунок 2.3 – Схема БД

Таблиця `toyexchanger_proposition_content` представляє вміст пропозиції, включаючи поля:

- `id` – номер запису про вміст пропозиції про обмін;
- `proposition_id` – пропозиція, вміст якої визначається;

– toy_id – іграшка, яка внесена в пропозицію.

Таблиця toyexchanger_proposition_content реалізує зв'язок типу багато до багатьох між таблицями toyexchanger_proposition і toyexchanger_toy, оскільки кожна пропозиція може включати різні дитячі іграшки, а кожна дитяча іграшка може входити в різні пропозиції.

Таблиця toyexchanger_change представляє обмін дитячими іграшками, включаючи поля:

- id – номер запису про обмін дитячих іграшок;
- proposition_id – пропозиція, за якою виконано обмін, тобто фактично підтверджена пропозиція;
- approved – момент фіксування обміну;
- comment – особливості обміну.

Таблиці toyexchanger_change і toyexchanger_proposition мають зв'язок типу один до одного, бо один обмін може бути виконаний тільки на основі однієї висунутої пропозиції, а одна пропозиція може бути закріплена тільки одним обміном.

Таблиця toyexchanger_message представляє повідомлення користувача, включаючи поля:

- id – номер повідомлення;
- author_id – автор повідомлення;
- receiver_id – отримувач повідомлення;
- messtext – текст повідомлення;
- created – момент створення повідомлення.

Таблиці toyexchanger_exchangeruser і toyexchanger_message мають зв'язок типу один до багатьох через поля author_id і receiver_id, бо один користувач може мати багато повідомлень, надісланих йому або в яких він є автором, але кожне повідомлення може мати тільки одного автора і одного отримувача.

2.4 Висновки за розділом 2

У другому розділі забезпечено реалізацію прийняття рішення щодо вибору мови програмування Python, моделювання сценаріїв роботи неавторизованого і авторизованого користувачів з програмою, проєктування БД з визначенням всіх таблиць для забезпечення функціональних вимог до програми і зв'язків між ними.

3 РОЗРОБКА ПРОГРАМИ

3.1 Послідовність роботи

Центральним процесом, комунікація через який відбувається в програмі, є процес роботи над оголошеннями. Оголошення по суті представляє наявну дитячу іграшку і відображає її у вигляді, прийнятному для пошуку іншими користувачами. Окрім того, як було визначено під час проєктування БД, існують поняття пропозиції та обміну. Кожна з цих сутностей залежна і впливає на інші. Тому важливо розглянути відповідну послідовність під час реалізації процесу роботи над обміном дитячих іграшок (рис. 3.1).

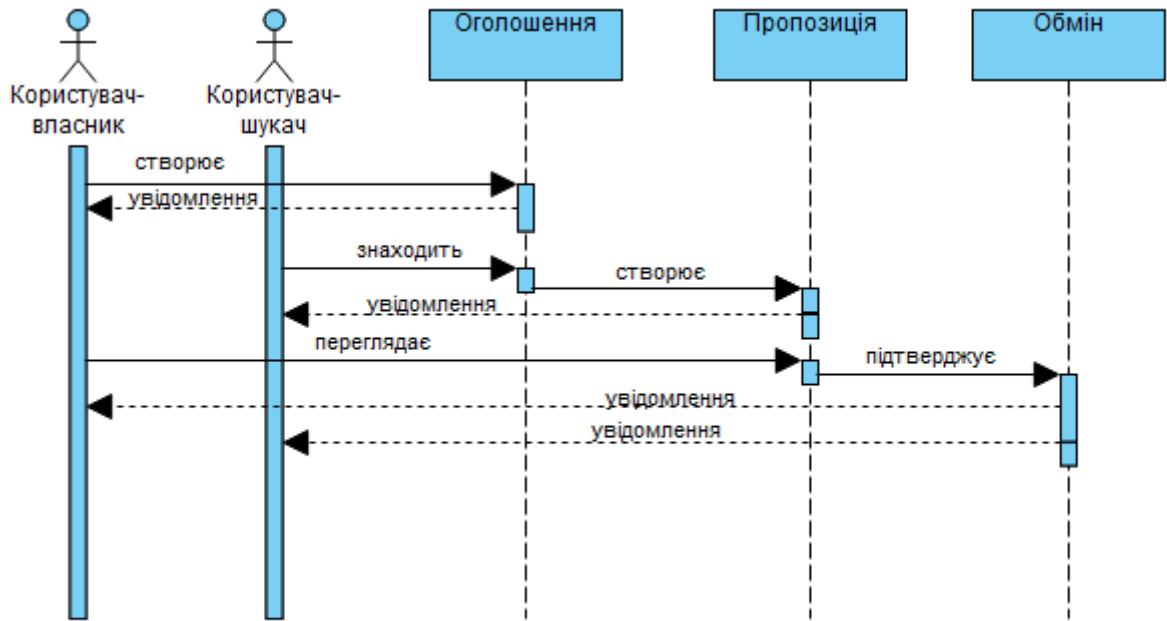


Рисунок 3.1 – Діаграма послідовності роботи над обміном дитячих іграшок

У даному процесі беруть участь два користувачі. Це користувач-власник іграшки і користувач-шукач, який приймає зусилля для того, щоб знайти потрібну йому іграшку на обмін.

Спочатку користувач-власник створює оголошення, таким чином формалізуючи наявну у себе дитячу іграшку, у відповідь він отримує повідомлення про успішне створення оголошення. Після того, як оголошення створено, користувач-шукач знаходить це оголошення і, ідентифікуючи його як потрібне йому, створює пропозицію про обмін дитячими іграшками. У свою чергу користувач-шукач отримує повідомлення про успішне створення пропозиції.

Після цього користувач-власник переглядає створену користувачем-шукачем для нього пропозицію. Якщо він вважає її прийнятною, то він підтверджує обмін і відповідно створюється об'єкт обміну, про успішне створення якого отримує інформацію користувач-шукач і користувач-власник.

3.2 Опис реалізованих класів програми

Реалізацію виконано з застосуванням фреймворку розробки, який підтримує об'єктно-реляційне перетворення і адміністративну панель, Django [9]-[10] і шаблону Model-View-Template (MVT) [11].

Спочатку було реалізовано набір класів, які відповідають таблицям БД, що були представлені в розділі 2. Ці класи були використані для створення після цього структури БД і роботи з нею в подальшому класів представлень. Під кожен відповідний клас представлення було забезпечено відповідний шаблон для заповнення вебсторінки даними представлення. Таким чином, структура класів відповідає шаблону MVT.

Класи моделей мають подібну до таблиць БД структуру, але мають і певні відмінності. Наприклад, клас `Toy`, створений як і всі інші класи моделей наслідуванням класу `models.Model`, має структуру:

– поле `totitle` – назва дитячої іграшки, що є полем символьних даних з максимальною довжиною у 255 символів `models.CharField(max_length=255)`;

– поле `about` – опис дитячої іграшки, що є полем текстових даних `models.TextField()`;

– поле `category_id` – категорія, до якої належить дитяча іграшка, що є полем зовнішнього ключа з моделі `Category` `models.ForeignKey(Category, related_name='categorytoys', on_delete=models.CASCADE)` з використанням зворотного імені для звернення з категорій – `categorytoys`;

– поле `startrecomage` – мінімальний рекомендований вік дитини, що є полем невеликого цілочисельного типу `models.SmallIntegerField(default = 0)` зі значенням за замовчуванням 0;

– поле `endrecomage` – максимальний рекомендований вік дитини, що є полем невеликого цілочисельного типу `models.SmallIntegerField(default = 18)` зі значенням за замовчуванням 18;

– поле `startage` – мінімальний вік, починаючи з якого дитина може грати з іграшкою, що є полем невеликого цілочисельного типу `models.SmallIntegerField(default = 0)` з нульовим значенням за замовчуванням;

– поле `gender` – стать дитини, для якої створена дитяча іграшка, що є символьним полем `models.CharField(max_length=1, choices=GENDERS, default='N')`, що може приймати одне з трьох значень, які відповідають чоловічій, жіночій статі або будь-якій статі;

– поле `toymodel_id` – модель дитячої іграшки, яка є полем зовнішнього ключа з моделі `Toymodel` `models.ForeignKey(Toymodel, related_name='modeltoys', on_delete=models.CASCADE)` з використанням зворотного імені для звернення з моделями дитячих іграшок – `modeltoys`;

– поле `is_certified` – сертифікована дитяча іграшка або не сертифікована, що є полем логічного типу `models.BooleanField()`;

– поле `produced` – рік виробництва іграшки, що є полем невеликого цілочисельного поля `models.SmallIntegerField()`;

– поле `country` – країна виробництва іграшки, що визначається цілим числом, яке на стороні клієнта перетворюється на відповідну назву і представлено полем невеликого цілого числа `models.SmallIntegerField()`;

– поле `color` – колір іграшки, що визначається цілим числом, яке на стороні клієнта перетворюється на відповідну назву і представлене полем невеликого цілого числа `models.SmallIntegerField()`;

– поле `toupic` – шлях до папки з файлами фотографій дитячої іграшки, що представлений файловим полем `models.FileField(upload_to='images/')` зі значенням за замовчуванням;

– поле `materials` – набір матеріалів, з якої створена дитяча іграшка, що є полем, яке визначає зв'язок багато до багатьох між даною моделлю і моделлю `Material` `models.ManyToManyField(Material, related_name = 'mattoys', db_table = 'toy_material')`, що забезпечує створення зв'язуючої таблиці БД `toy_material` шляхом з'єднання ключів відповідних таблиць: це дозволяє не виконувати таке зв'язування вручну, а виконувати це автоматично, що зменшує кількість коду, яку потрібно написати, завдяки чому саме Python і Django використано для розробки.

Окрім того у певних моделях реалізовано особливі методи, необхідні для підтримки базової роботи відповідної сутності.

Наприклад, клас моделі `Opinion` має метод `save(self, *args, **kwargs)`, який забезпечує при збереженні враження користувача одночасне переобчислення середнього балу, виставленого моделі дитячої іграшки, що відповідає моделі дитячої іграшки, за якою створено враження. Також автоматично вноситься поточний момент в якості значення моменту створення враження.

Клас `ExchangerUser` створений на основі наслідування вбудованого класу Django `AbstractUser`, що дозволило основну кількість полів, визначену у відповідній таблиці БД наслідувати без необхідності їх визначення вручну в класі.

Також клас `Statement` має метод `save(self, *args, **kwargs)`, який забезпечує внесення дати створення оголошення у відповідності з поточним під час збереження моментом часу. Аналогічні засоби наявні в класі

пропозиції про обмін `Proposition`. Ще за подібним способом влаштовано класи фактичного обміну `Change` і повідомлення `Message`.

Усі класи представлень розбиті на два окремі файли для зручності роботи.

Файл `toyview.py` містить класи представлень, які стосуються безпосередньо самих дитячих іграшок та оголошень.

Клас `ToyView` визначає представлення деталізації дитячої іграшки наслідуванням класу `DetailView`.

Клас `ToyAddView` визначає представлення додавання дитячої іграшки наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `ToyUpdateView` визначає представлення редагування дитячої іграшки наслідуванням класу `UpdateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `MyStatementsView` визначає представлення відображення власних оголошень наслідуванням класу `ListView`.

Клас `ToyStatementsView` визначає представлення відображення оголошень за заданою дитячою іграшкою наслідуванням класу `ListView`.

Клас `ToyOpinionsView` визначає представлення відображення вражень за заданою дитячою іграшкою наслідуванням класу `ListView`.

Клас `ToyDamagesView` визначає представлення відображення пошкоджень за заданою дитячою іграшкою наслідуванням класу `ListView`.

Клас `StatementView` визначає представлення деталізації оголошення наслідуванням класу `DetailView`.

Клас `StatementByOwnToySearchView` визначає представлення пошуку оголошень за заданою власною дитячою іграшкою наслідуванням класу `ListView`.

Клас `StatementAddView` визначає представлення додавання нового оголошення наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `StatementUpdateView` визначає представлення редагування оголошення наслідуванням класу `UpdateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `StatementSearchView` визначає представлення пошуку оголошень наслідуванням класу `ListView`.

Клас `UserStatementsView` визначає представлення відображення власних оголошень користувача наслідуванням класу `ListView`.

Клас `ToymodelAddView` визначає представлення додавання нової моделі дитячої іграшки наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `ToymodelView` визначає представлення деталізації моделі дитячої іграшки наслідуванням класу `DetailView`.

Клас `UserStatementToysView` визначає представлення відображення дитячих іграшок, які були внесені за оголошеннями авторизованого користувача, наслідуванням класу `ListView`.

Клас `ToymodelUpdateView` визначає представлення редагування моделі дитячої іграшки наслідуванням класу `UpdateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `ToyStatementsView` визначає представлення відображення оголошень про обмін дитячими іграшками за заданою дитячою іграшкою наслідуванням класу `ListView`.

Клас `CategoryStatementsView` визначає представлення відображення активних оголошень за заданою категорією дитячих іграшок наслідуванням класу `ListView`.

Клас `CategoriesView` визначає представлення відображення категорій дитячих іграшок наслідуванням класу `ListView`.

Клас `StatementContentAddView` визначає представлення додавання нової вимоги до оголошення наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `StatementContentUpdateView` визначає представлення редагування вимоги до оголошення щодо обміну дитячими іграшками наслідуванням класу `UpdateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `DamageUpdateView` визначає представлення редагування пошкодження дитячої іграшки наслідуванням класу `UpdateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `DamageAddView` визначає представлення додавання нового пошкодження дитячої іграшки наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `StatementByToyRestrictionsSearchView` визначає представлення пошуку оголошень про обмін дитячими іграшками на основі власного оголошення наслідуванням класу `ListView`.

Другою складовою представлень є файл `changeview.py`, який містить класи представлень, які стосуються фактичних обмінів іграшок, пропозицій з такого обміну та повідомлень між користувачами.

Клас `PropositionAddView` визначає представлення додавання нової пропозиції про обмін дитячими іграшками до відповідного оголошення наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `PropositionUpdateView` визначає представлення редагування пропозиції про обмін дитячими іграшками до відповідного оголошення наслідуванням класу `UpdateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `PropositionsByStatementView` визначає представлення відображення пропозицій до оголошення про обмін дитячими іграшками наслідуванням класу `ListView`.

Клас `PropositionContentAddView` визначає представлення додавання нової вимоги до пропозиції про обмін дитячими іграшками до відповідного

оголошення наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `PropositionContentUpdateView` визначає представлення редагування вимоги до пропозиції про обмін дитячими іграшками до відповідного оголошення наслідуванням класу `UpdateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `PropositionContentView` визначає представлення відображення вмісту пропозиції до оголошення щодо обміну дитячими іграшками наслідуванням класу `ListView`.

Клас `ChangeAddView` визначає представлення підтвердження пропозиції про обмін дитячими іграшками до відповідного оголошення наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `MessageAddView` визначає представлення додавання нового повідомлення від одного користувача іншому наслідуванням класу `CreateView` і класу, що вимагає попередньої авторизації користувача, `LoginRequiredMixin`.

Клас `UsersMessView` визначає представлення відображення списку користувачів, які взаємодіяли з авторизованим користувачем, наслідуванням класу `ListView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

Клас `UserMessagesView` визначає представлення відображення списку повідомлень авторизованому користувачу від заданого користувача наслідуванням класу `ListView` і класу, що вимагає попередньої авторизації, `LoginRequiredMixin`.

3.3 Висновки за розділом 3

У третьому розділі розглянуто послідовність роботи над обміном дитячих іграшок на основі відповідних оголошень за допомогою створеної

діаграми послідовності, представлено основи розробки і відповідні створені класи програми, розробленої за допомогою вебфреймворку Django і шаблону MVT.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для підтримки процесу обміну власних дитячих іграшок на інші дитячі іграшки з забезпеченням збереження історії стану кожної дитячої іграшки і вражень користувачів та наданням доступу до такої історії.

4.2 Умови виконання програми

Технічні засоби в мінімальній конфігурації, достатній для роботи програми, охоплюють 1 Гб пам'яті жорсткого диску, 4 Гб оперативної пам'яті, 4 ядра процесору (частота 2,8 ГГц).

4.3 Тестування і опис принципів експлуатації програми

Під час виконання тестування програми перевірялись вимоги згідно з вимогами технічного завдання. Функції перевірялись на різних типах пристроїв з забезпеченням через них доступу до вебсервера. Тестування виявилось успішним.

Під час тестування використовувалась логіка взаємодії з програмою, визначена меню користувачів і функціональними вимогами.

При відкритті вебсайту відкривається початкова сторінка (рис. 4.1). Початкова сторінка містить меню, яке включає пункти «Категорії», звідки доступні категорії дитячих іграшок, звідки можна перейти до оголошень за відповідною категорією, «Бренди», звідки можна перейти від брендів до моделей, а звідти до оголошень за конкретними моделями або за іграшками за моделями, «Пошук», звідки можна виконати пошук оголошень.

Початкова сторінка після авторизації включає ширше меню (рис. 4.2), яке додатково містить пункти «Іграшки», «Оголошення», «Пропозиції».

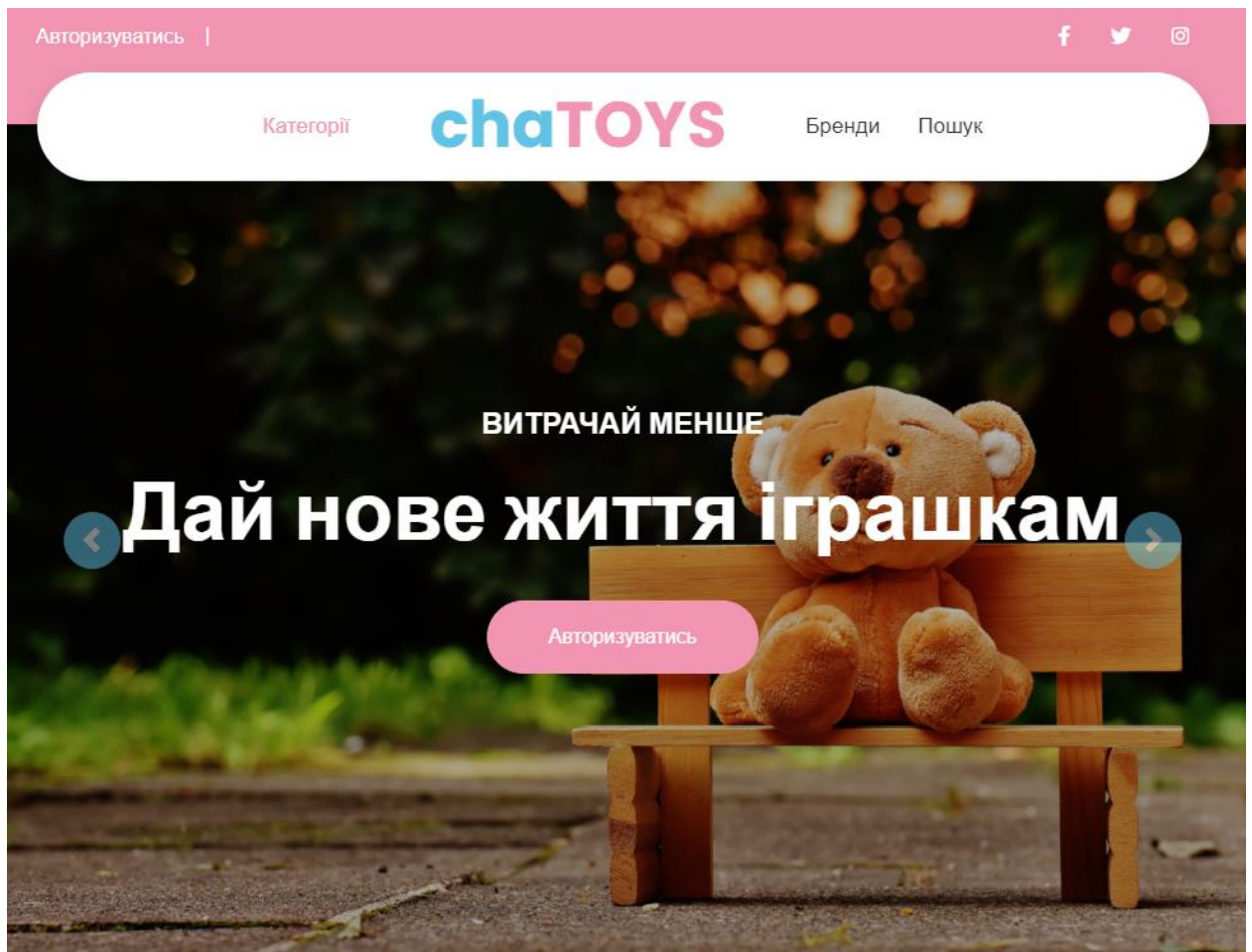


Рисунок 4.1 – Початкова сторінка без авторизації

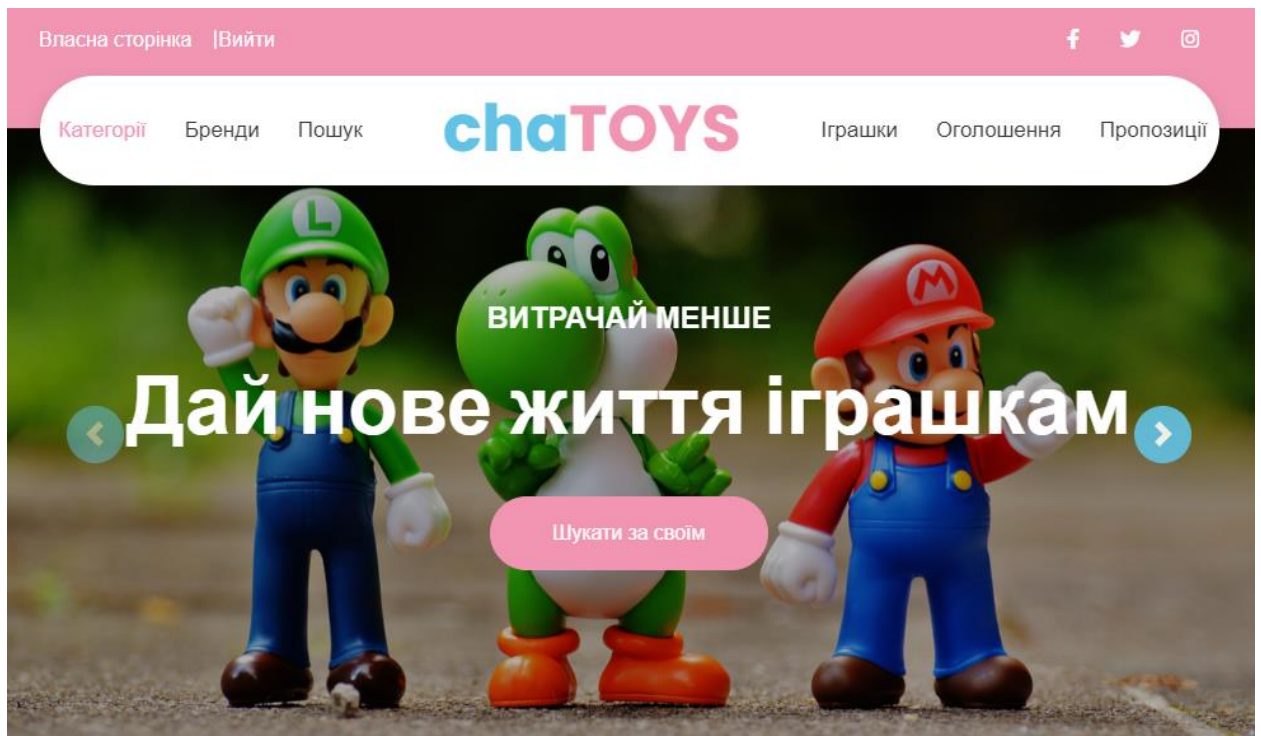


Рисунок 4.2 – Початкова сторінка після авторизації

Список категорій залежить від категорій, які були задані в БД (рис. 4.3).

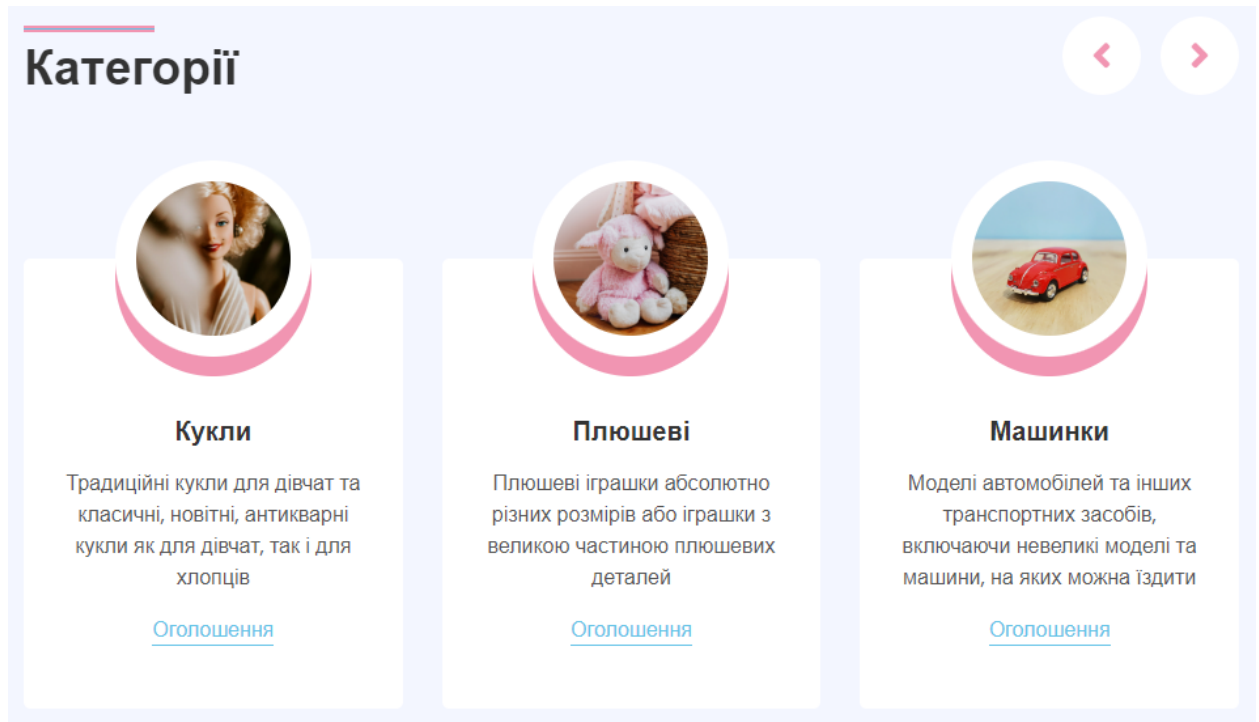


Рисунок 4.3 – Відображення категорій іграшок

Між категоріями можна переходити, перемикаючись до наступних, за допомогою стрілок вліво та вправо.

Пошук відрізняється для авторизованого і неавторизованого користувачів.

Для неавторизованого користувача виконується тільки загальний пошук, який містить наступні параметри:

- назву іграшки;
- категорію іграшки;
- список брендів;
- місто, де має знаходитись іграшка (або можна не задавати місто, при цьому іграшка може бути і відправлена службою доставки);
- список матеріалів, з яких може складатися іграшка;
- вік дитини;

- стаття дитини;
- модель іграшки;
- період років, протягом яких має бути вироблена іграшка;
- країна виробництва;
- колір іграшки.

У випадку авторизації з пункту меню «Пошук» доступні одразу три варіанти за допомогою відповідних кнопок «Звичайний», який був описаний вище, «За іграшкою», «За оголошенням» (рис. 4.4).

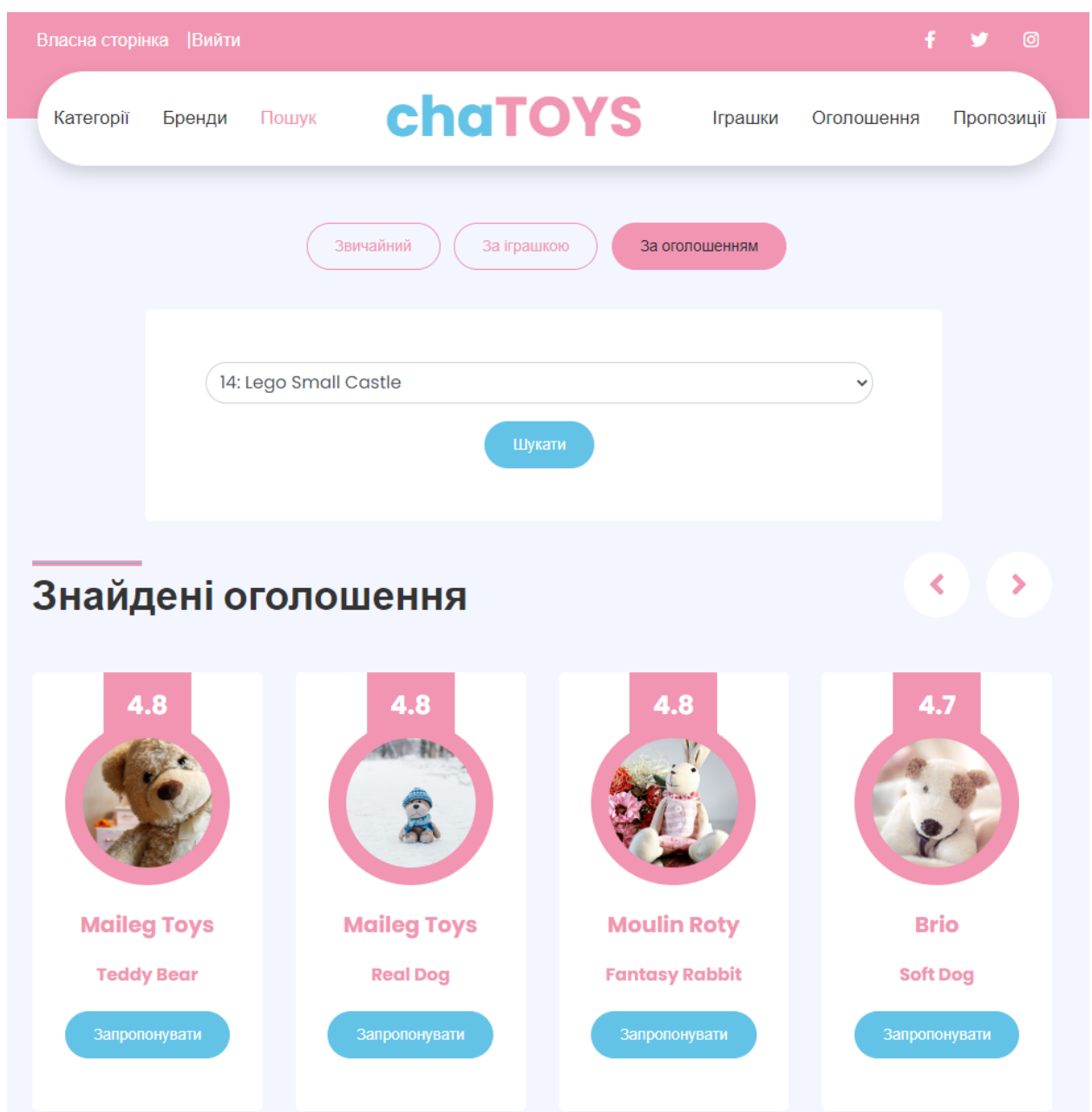


Рисунок 4.4 – Пошук оголошень за власним оголошенням

При пошуку за іграшкою обирається потрібне власне оголошення і знаходяться всі оголошення інших користувачів, вимогам яких відповідає іграшка з оголошення авторизованого користувача.

При пошуку за оголошенням одночасно виконується пошук оголошень, вимогам яких задовольняє іграшка з власного обраного зі списку оголошення, а відповідно іграшка зі знайденого оголошення має відповідати вимогам обраного зі списку оголошення.

За результатом пошуку відображається карусель з результатами пошуку, між якими можна переключатися за допомогою стрілочок. У цій каруселі присутньо 10 оголошень. Внизу сторінки є номери сторінок, які дозволяються перемикатися між поточними і наступними, попередніми 10 оголошеннями.

За кожним оголошенням в результатах пошуку відображається середня оцінка моделі іграшки, зображення самої іграшки (якщо натиснути на картинку, то відкриється сторінка самого оголошення), виробник іграшки та модель іграшки, натисканням на ці посилання можна перейти до відповідних сторінок. За допомогою кнопки «Запропонувати» можна перейти до створення пропозиції до оголошення.

При переході на сторінку оголошення можна звідти перейти до сторінки самої дитячої іграшки. У випадку авторизованого користувача при цьому, якщо дана іграшка використовувалась в його оголошеннях, йому доступні засоби редагування даних іграшки. Також з цієї сторінки користувач може додавати пошкодження, а також підтверджувати або скасовувати існуючі пошкодження.

З пункту меню «Іграшки» користувач може переглянути список всіх дитячих іграшок, які використовувались в оголошеннях авторизованого користувача. Звідси також за допомогою кнопок створення, редагування, а також пошкодження можна переходити і виконувати відповідні дії.

З пункту меню «Оголошення» користувачу доступні всі його оголошення (рис. 4.5) при виборі кнопки «Оголошення». При виборі кнопки «Обміни» відображаються оголошення, які отримали фактичний обмін.

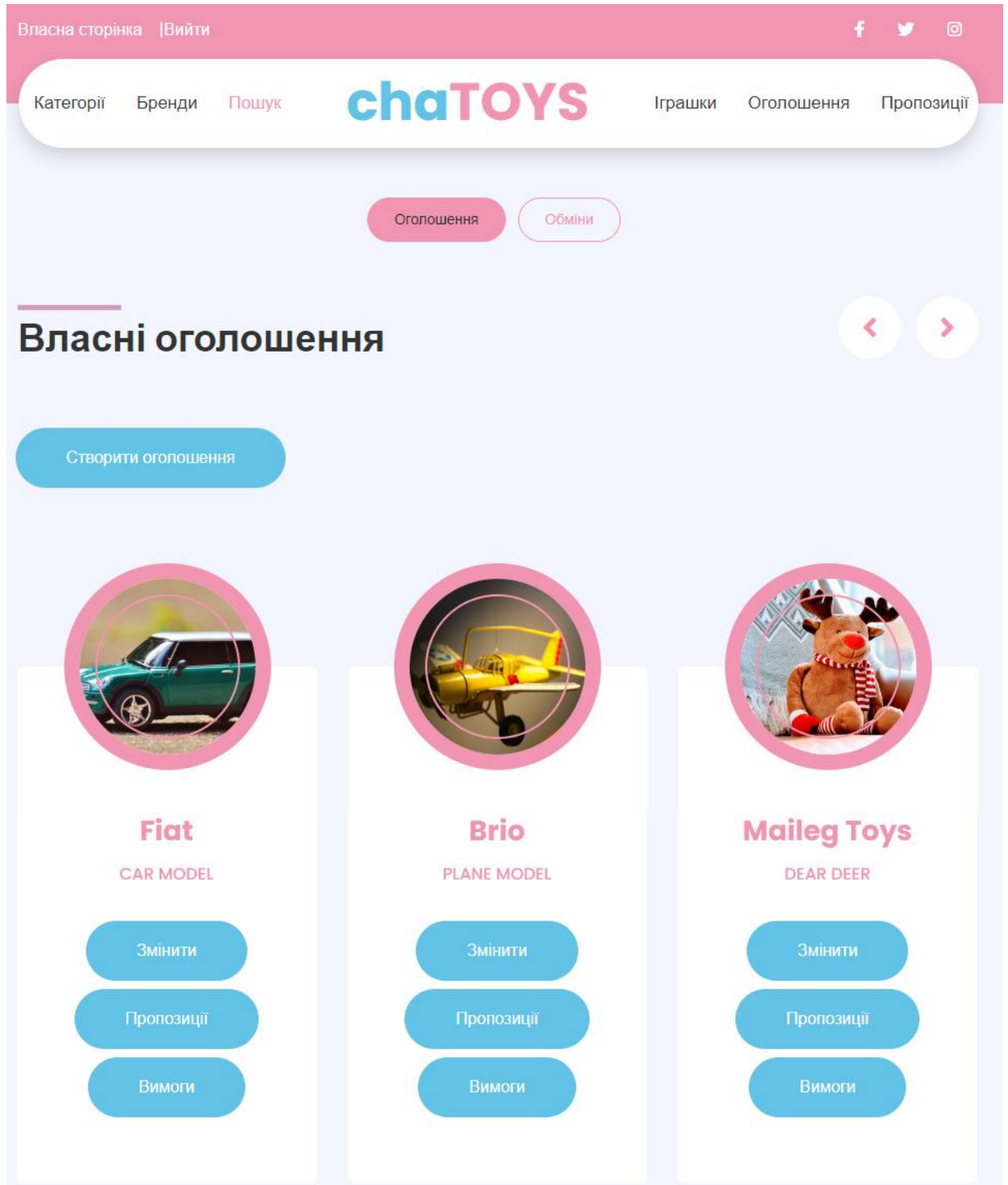


Рисунок 4.5 – Керування власними оголошеннями

За кожним оголошенням доступна можливість до переходу до набору пропозицій або вимог. При переході до пропозицій доступні кнопки «Обмін» для фактичного підтвердження пропозиції і виконання обміну. Під час обміну можна вказати також адресу і службу доставки.

Аналогічно влаштована сторінка «Пропозиції», звідки доступні власні пропозиції користувача, а також обміни, які за цими пропозиціями були ухвалені.

4.4 Висновки за розділом 4

Виписано процес роботи авторизованого і неавторизованого користувачів. Забезпечено тестування програми, що завершилось успіхом в підсумку.

ВИСНОВКИ

У дипломній роботі успішно досягнуто мету розроблення програмного забезпечення обміну дитячими іграшками для зменшення витрат людей.

Проведено весь процес розробки програми, включаючи аналіз проблеми, визначення і моделювання вимог до програми, проєктування БД, розробку програми за допомогою фреймворку Django, що призвело до отримання відповідного практичного інструменту.

У дипломній роботі забезпечено реалізацію прийняття рішення щодо вибору мови програмування Python, моделювання сценаріїв роботи неавторизованого і авторизованого користувачів з програмою за допомогою діаграм варіантів використання, проєктування БД з визначенням всіх таблиць для забезпечення функціональних вимог до програми і зв'язків між ними, розглянуто послідовність роботи над обміном дитячих іграшок на основі відповідних оголошень за допомогою створеної діаграми послідовності, представлено основи розробки і відповідні створені класи програми, розробленої за допомогою вебфреймворку Django на основі шаблону проєктування MVT, описано процес взаємодії користувача з програмною системою.

Виконана розробка представлена програмою, що призначена для підтримки процесу обміну власних дитячих іграшок на інші дитячі іграшки з забезпеченням збереження історії стану кожної дитячої іграшки і вражень користувачів та наданням доступу до такої історії. Це дозволяє батькам уникнути фінансових витрат у випадку їх неможливості або зменшити такі витрати за потреби у процесі розвитку дитини, використовуючи наявну в сім'ї базу іграшок. Тестування програми вказало на успішну реалізацію програми.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Toys Market Size, Share, Growth. Global Industry Trends [2028] [Electronic resource]. – Access mode : <https://www.fortunebusinessinsights.com/toys-market-104699>.
2. The European Toys Market. Spring GDS [Electronic resource]. – Access mode : <https://www.spring-gds.com/support/market-insights/toys-market-insights/>.
3. Grimmer, L. Why kids should not have lots of toys – and what to do if yours have too many [Electronic resource] / L. Grimmer, M. Grimmer. – Access mode : <https://www.abc.net.au/news/2021-12-20/why-kids-shouldnt-have-lots-of-toys-what-do-too-many/100704836>.
4. Etsy Ukraine – Shop for handmade, vintage, custom, and unique gifts for everyone [Electronic resource]. – Access mode : <https://www.etsy.com/>.
5. Toy Exchange Club. Educational Wooden Toy Subscription Box [Electronic resource]. – Access mode : <https://toyexchange.ca/>.
6. Matthes, E. Python Crash Course [Text] : A Hands-On, Project-Based Introduction to Programming / E. Matthes. – San Francisco : No Starch Press, 2019. – 544 p.
7. Ramalho, L, Fluent Python [Text] : Clear, Concise, and Effective Programming / L. Ramalho. – Newton, Massachusetts : O'Reilly Media, 2022. – 1012 p.
8. Grippa, V. M. Learning MySQL [Text] : Get a Handle on Your Data / V. M. Grippa, S. Kuzmichev. – Newton, Massachusetts : O'Reilly Media, 2021. – 632 p.
9. Vincent, W. S. Django for Beginners [Text] : Build Websites with Python and Django (Welcome to Django) / W. S. Vincent. – Bristol : WelcomeToCode, 2020. – 317 p.
10. Vincent, W. S. Django for Professionals [Text] : Production websites with Python & Django (Welcome to Django) / W. S. Vincent. – Bristol :

WelcomeToCode, 2020. – 314 p.

11. Django Model View Template (MVT) Overview [Electronic resource]. – Access mode : <https://www.onlinetutorialspoint.com/django/django-model-view-template-mvt-overview.html>.

ДОДАТОК А
Технічне завдання

Вступ

Специфіка взаємодії дитини з речима така, що вони фактично постійно змінюються. Така постійна зміна призводить до великих витрат. Не всі батьки можуть постійно дозволити собі таким витрати в повному обсязі. При цьому необхідність оновлення дитячих речей, зокрема дитячих іграшок, є достатньо високою. Одним із рішень цієї проблеми є обмін дитячих іграшок, які є у власні батьків, на інші дитячі іграшки. На це і направлена програма, що має бути створена.

А.1 Підстави для розробки

Розробка в межах дипломної кваліфікаційної роботи реалізується на підставі наказу № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення обміну дитячими іграшками».

А.2 Призначення розробки

Програма призначена для підтримки процесу обміну власних дитячих іграшок на інші дитячі іграшки з забезпеченням збереження історії стану кожної дитячої іграшки і вражень користувачів та наданням доступу до такої історії.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Функції, які виконуватиме програма за результатами розробки:

- ознайомлення з дитячою іграшкою;
- додавання дитячої іграшки;

- редагування даних дитячої іграшки;
- відображення виставлених на обмін власних дитячих іграшок;
- відображення оголошень про обмін за дитячою іграшкою;
- відображення вражень про дитячу іграшку;
- відображення пошкоджень дитячої іграшки;
- ознайомлення з оголошенням про обмін дитячої іграшки;
- пошук оголошень про обмін дитячих іграшок;
- додавання оголошення про обмін дитячої іграшки;
- редагування оголошення про обмін дитячої іграшки;
- пошук оголошень, що відповідають наявній у користувача дитячій іграшці;
- пошук оголошень, що відповідають власному оголошенню користувача;
- відображення оголошень користувача про обмін дитячих іграшок;
- додавання моделей дитячих іграшок;
- відображення дитячих іграшок заданої моделі;
- редагування моделі дитячої іграшки;
- відображення оголошень за заданою дитячою іграшкою;
- відображення оголошень про обмін дитячих іграшок заданої категорії;
- відображення списку категорій дитячих іграшок;
- додавання вимоги до оголошення;
- редагування вимоги в оголошенні;
- додавання пошкодження дитячої іграшки;
- редагування даних пошкодження дитячої іграшки;
- відображення моделей дитячих іграшок за брендом;
- надсилання повідомлень користувачу;
- відображення отриманих повідомлень від користувачів;
- висунення пропозиції про обмін дитячої іграшки на оголошення;
- підтвердження пропозиції про обмін дитячої іграшки;

- додавання враження;
- авторизація і реєстрація користувачів.

A.3.2 Умови експлуатації

Рекомендовано для забезпечення роботи програми виділити системного адміністратора, до зобов'язань якого віднести підтримку роботи вебсервера, сервера БД та внесення за потреби базових даних.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічні засоби в мінімальній конфігурації, достатній для роботи програми, охоплюють 1 Гб пам'яті жорсткого диску, 4 Гб оперативної пам'яті, 4 ядра процесору (частота 2,8 ГГц).

A.4 Порядок контролю та приймання

Остаточна перевірка отриманих результатів виконання роботи здійснюється під час захисту дипломної роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла toyview.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.edit import CreateView
from django.views.generic.detail import DetailView
from django.urls import reverse_lazy
from .models import Damage
from .models import Opinion
from .models import Toy
from .models import Statement
from .models import Category
from .models import Statement_content
from .models import ExchangerUser
from .models import Toymodel
from .models import Brand

class ToyView (DetailView):
    template_name = 'toy.html'
    model = Toy

    def get_context_data(self, **kwargs):
        toydata = super().get_context_data(**kwargs)
        dettoy = get_object_or_404(Toy, id = self.object.pk)
        toydata ['damages'] = Damage.objects.filter(toy_id =
dettoy.id).order_by('-created')
        toydata ['statements'] =
Statement.objects.filter(toy_id = dettoy.id).order_by('-created')
        toydata ['opinions'] = Opinion.objects.filter(toy_id =
dettoy.id).order_by('-created')
        return toydata

class ToyAddView(LoginRequiredMixin, CreateView):
    model = Toy
    fields = ['totitle', 'about', 'category_id',
'sartrecomage', 'endrecomage', 'startage', 'gender',
'model_id', 'is_certified', 'produced', 'country', 'color',
'toypic']
    template_name = 'toyadd.html'

    def get_success_url(self):

```

```

        return reverse_lazy('toyaddview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.category_id = get_object_or_404(Category,
id=self.kwargs['category_id'])
        self.toymodel_id = get_object_or_404(Toymodel,
id=self.kwargs['toymodel_id'])
        kwargs['category_id'] = self.category_id
        kwargs['toymodel_id'] = self.toymodel_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.category_id = get_object_or_404(Category,
id=self.kwargs['category_id'])
        self.toymodel_id = get_object_or_404(Toymodel,
id=self.kwargs['toymodel_id'])
        form.instance.category_id = self.category_id
        form.instance.toymodel_id = self.toymodel_id
        return super().form_valid(form)

class ToyUpdateView(LoginRequiredMixin, UpdateView):
    model = Toy
    fields = ['totitle', 'about', 'category_id',
'startrecomage', 'endrecomage', 'startage', 'gender', 'model_id',
'is_certified', 'produced', 'country', 'color', 'toypic']
    template_name = 'toyupdate.html'

    def get_success_url(self):
        return reverse_lazy('toyupdateview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.category_id = get_object_or_404(Category,
id=self.kwargs['category_id'])
        self.toymodel_id = get_object_or_404(Toymodel,
id=self.kwargs['toymodel_id'])
        kwargs['category_id'] = self.category_id
        kwargs['toymodel_id'] = self.toymodel_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.category_id = get_object_or_404(Category,
id=self.kwargs['category_id'])

```

```

        self.toymodel_id = get_object_or_404(Toymodel,
id=self.kwargs['toymodel_id'])
        form.instance.category_id = self.category_id
        form.instance.toymodel_id = self.toymodel_id
        return super().form_valid(form)

```

```

class MyStatementsView(ListView):
    model = Statement
    template_name = 'mystatements.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        fulluser = ExchangerUser.objects.get(id = user.id)
        stats = Statement.objects.filter(exchangeruser_id =
fulluser)

        stats = stats.filter(is_active = True)
        stats = stats.order_by('-created')
        return stats

```

```

class ToyStatementsView(ListView):
    model = Statement
    template_name = 'toystatements.html'

    def get_queryset(self):
        toymodel_id = self.request.GET.get('toy')
        stats = Statement.objects.filter(toymodel_id = toymodel_id)
        stats = stats.order_by('-created')
        return stats

    def get_context_data(self, **kwargs):
        statresults = super(ToyStatementsView,
self).get_context_data(**kwargs)
        toymodel_id = self.request.GET.get('toy')
        statresults['toy'] = toymodel_id
        return statresults

```

```

class ToyOpinionsView(ListView):
    model = Opinion
    template_name = 'toyopinions.html'

    def get_queryset(self):
        toymodel_id = self.request.GET.get('toy')
        stats = Opinion.objects.filter(toymodel_id = toymodel_id)
        stats = stats.order_by('-created')

```

```

        return stats

    def get_context_data(self, **kwargs):
        statresults = super(ToyOpinionsView,
self).get_context_data(**kwargs)
        toyid = self.request.GET.get('toy')
        statresults ['toy'] = toyid
        return statresults

class ToyDamagesView(ListView):
    model = Damage
    template_name = 'toydamages.html'

    def get_queryset(self):
        toyid = self.request.GET.get('toy')
        stats = Damage.objects.filter(toy_id__id = toyid)
        stats = stats.order_by('-created')
        return stats

    def get_context_data(self, **kwargs):
        statresults = super(ToyDamagesView,
self).get_context_data(**kwargs)
        toyid = self.request.GET.get('toy')
        statresults ['toy'] = toyid
        return statresults

class StatementView (DetailView):
    template_name = 'statement.html'
    model = Statement

class StatementByOwnToySearchView(ListView):
    model = Statement
    template_name = 'statebyowntoysearch.html'
    paginate_by = 10

    def get_queryset(self):
        statementid = self.request.GET.get('statement')
        stat = Statement.objects.get(id = statementid)
        toycateg = stat.toy_id.category_id
        stats = Statement_content.objects.filter(category_id =
toycateg)

        stats = stats.filter(statement_id__is_active = True)
        stats = stats.filter(age__gte =
statementid.toy_id.startage)

```

```

        stats = stats.filter(Q(gender = 'N') | Q(gender =
stat.toy_id.gender))
        stats = stats.filter(is_certified =
stat.toy_id.is_certified)
        stats = stats.filter(startproduced__lte =
stat.toy_id.produced)
        stats = stats.filter(endproduced__gte =
stat.toy_id.produced)
        stats = stats.filter(country = stat.toy_id.country)
        stats = stats.filter(color = stat.toy_id.color)
        stats = stats.filter(startpoint__lte =
stat.toy_id.toymodel_id.point)
        stats = stats.filter(endpoint__gte =
stat.toy_id.toymodel_id.point)
        newstats = stats
        for s in stats:
            donemater = True
            for mater in stat.toy_id.materials:
                if not s.filter(materials__in = [mater.id]):
                    donemater = False
            if s.brand_id not in stat.brands:
                donemater = False
            if not donemater:
                newstats.filter(id = s.id).delete()
        st = Statement.objects.filter(id__in = newstats)
        return st

    def get_context_data(self, **kwargs):
        statresults = super(StatementByOwnToySearchView,
self).get_context_data(**kwargs)
        stat = self.request.GET.get('statement')
        statresults ['statement'] = stat
        return statresults

class StatementAddView(LoginRequiredMixin, CreateView):
    model = Statement
    fields = ['toy_id', 'city']
    template_name = 'statementadd.html'

    def get_success_url(self):
        return reverse_lazy('stataddview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):

```

```

        self.toy_id = get_object_or_404(Toy,
id=self.kwargs['toy_id'])
        kwargs['toy_id'] = self.toy_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.toy_id = get_object_or_404(Toy,
id=self.kwargs['toy_id'])
        form.instance.toy_id = self.toy_id
        user = self.request.user
        fulluser = ExchangerUser.objects.get(id = user.id)
        form.instance.exchangeruser_id = fulluser
        form.is_active = True
        return super().form_valid(form)

class StatementUpdateView(LoginRequiredMixin, UpdateView):
    model = Statement
    fields = ['toy_id', 'is_active', 'city']
    template_name = 'statupdate.html'

    def get_success_url(self):
        return reverse_lazy('statupdateview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.toy_id = get_object_or_404(Toy,
id=self.kwargs['toy_id'])
        kwargs['toy_id'] = self.toy_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.toy_id = get_object_or_404(Toy,
id=self.kwargs['toy_id'])
        form.instance.toy_id = self.toy_id
        return super().form_valid(form)

class StatementSearchView(ListView):
    model = Statement
    template_name = 'statesearch.html'
    paginate_by = 10

    def get_queryset(self):
        stats = Statement.objects.filter(is_active = True)
        toytitle = self.request.GET.get('title')
        if toytitle:

```

```

        stats = stats.filter(toy_id__totitle__contains =
toytitle)
        city = self.request.GET.get('city')
        if city:
            stats = stats.filter(city__contains = city)
        categ = self.request.GET.get('category')
        cat = Category.objects.filter(id = categ)
        if categ:
            stats = stats.filter(toy_id__category_id = cat)
        brandids = self.request.GET.getlist('brands')
        if brandids:
            stats =
stats.filter(toy_id__model_id__brand_id__in = brandids)
        materials = self.request.GET.getlist('materials')
        if materials:
            newstats = stats
            for stat in stats:
                left = 0
                ss =
Statement_content.objects.filter(statement_id = stat)
                for s in ss:
                    l = True
                    for m in s.materials:
                        if m not in materials:
                            l = False
                    if l and left == 0:
                        left = 1
            if left == 0:
                newstats.filter(id = stat.id).delete()
        stats = newstats
        age = self.request.GET.get('age')
        if age:
            stats = stats.filter(toy_id__startage_lte = age)
        gender = self.request.GET.get('gender')
        if gender:
            stats = stats.filter(toy_id__gender = gender)
        toymodel = self.request.GET.get('toymodel')
        tm = Toymodel.objects.filter(id = toymodel)
        if tm:
            stats = stats.filter(toy_id__toymodel_id = tm)
        prod_min = self.request.GET.get('prod_min')
        if prod_min:
            stats = stats.filter(toy_id__produced__gte =
prod_min)
        prod_max = self.request.GET.get('prod_max')

```

```

        if prod_max:
            stats = stats.filter(toy_id__produced__lte =
prod_max)

        cntry = self.request.GET.get('country')
        if cntry:
            stats = stats.filter(toy_id__country = cntry)
        color = self.request.GET.get('color')
        if color:
            stats = stats.filter(toy_id__color = color)
        return stats.order_by('toy_id__toymodel_id__point')

    def get_context_data(self, **kwargs):
        statresults = super(StatementSearchView,
self).get_context_data(**kwargs)
        toytitle = self.request.GET.get('title')
        statresults ['title'] = toytitle
        city = self.request.GET.get('city')
        statresults ['city'] = city
        categ = self.request.GET.get('category')
        statresults ['category'] = categ
        brandids = self.request.GET.getlist('brands')
        statresults ['brandids'] = brandids
        materials = self.request.GET.getlist('materials')
        statresults ['materials'] = materials
        age = self.request.GET.get('age')
        statresults ['age'] = age
        gender = self.request.GET.get('gender')
        statresults ['gender'] = gender
        toymodel = self.request.GET.get('toymodel')
        statresults ['toymodel'] = toymodel
        prod_min = self.request.GET.get('prod_min')
        statresults ['prod_min'] = prod_min
        prod_max = self.request.GET.get('prod_max')
        statresults ['prod_max'] = prod_max
        cntry = self.request.GET.get('country')
        statresults ['country'] = cntry
        color = self.request.GET.get('color')
        statresults ['color'] = color
        return statresults

class UserStatementsView(ListView):
    model = Statement
    template_name = 'userstatements.html'
    paginate_by = 10

```

```

def get_queryset(self):
    userid = self.request.GET.get('user_id')
    fulluser = ExchangerUser.objects.get(id = userid)
    stats = Statement.objects.filter(exchangeruser_id =
fulluser)

    stats = stats.filter(is_active = True)
    stats = stats.order_by('-created')
    return stats

def get_context_data(self, **kwargs):
    stats = super(UserStatementsView,
self).get_context_data(**kwargs)
    userid = self.request.GET.get('user_id')
    stats ['user_id'] = userid
    return stats

class UserStatementToysView(ListView):
    model = Toy
    template_name = 'userstattoys.html'
    paginate_by = 10

def get_queryset(self):
    userid = self.request.GET.get('user_id')
    fulluser = ExchangerUser.objects.get(id = userid)
    stats = Statement.objects.filter(exchangeruser_id =
fulluser)

    toys = Toy.objects.filter(toy_id__in = stats)
    return toys

def get_context_data(self, **kwargs):
    stats = super(UserStatementToysView,
self).get_context_data(**kwargs)
    userid = self.request.GET.get('user_id')
    stats ['user_id'] = userid
    return stats

class ToymodelAddView(LoginRequiredMixin, CreateView):
    model = Toymodel
    fields = ['modtitle', 'about', 'brand_id', 'point']
    template_name = 'toymodeladd.html'

def get_success_url(self):
    return reverse_lazy('toymodeladdview', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        self.brand_id = get_object_or_404(Brand,
id=self.kwargs['brand_id'])
        kwargs['brand_id'] = self.brand_id
        return super().get_context_data(**kwargs)
    def form_valid(self, form):
        self.brand_id = get_object_or_404(Brand,
id=self.kwargs['brand_id'])
        form.instance.brand_id = self.brand_id
        return super().form_valid(form)

class ToymodelView (DetailView):
    template_name = 'toymodel.html'
    model = Toymodel

    def get_context_data(self, **kwargs):
        toydata = super().get_context_data(**kwargs)
        dettoy = get_object_or_404(Toymodel, id =
self.object.pk)
        toydata ['toys'] = Toy.objects.filter(toymodel_id =
dettoy.id).order_by('totitle')
        return toydata

class ToymodelUpdateView(LoginRequiredMixin, UpdateView):
    model = Toymodel
    fields = ['modtitle', 'about', 'brand_id', 'point']
    template_name = 'toymodelupdate.html'

    def get_success_url(self):
        return reverse_lazy('toymodelupdateview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        self.brand_id = get_object_or_404(Brand,
id=self.kwargs['brand_id'])
        kwargs['brand_id'] = self.brand_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.brand_id = get_object_or_404(Brand,
id=self.kwargs['brand_id'])
        form.instance.brand_id = self.brand_id
        return super().form_valid(form)

```

```

class ToyStatementsView(ListView):
    model = Statement
    template_name = 'toystatements.html'

    def get_queryset(self):
        toyid = self.request.GET.get('toy')
        stats = Statement.objects.filter(toy_id__id = toyid)
        stats = stats.filter(is_active = True)
        stats = stats.order_by('-created')
        return stats

    def get_context_data(self, **kwargs):
        statresults = super(ToyStatementsView,
self).get_context_data(**kwargs)
        toyid = self.request.GET.get('toy')
        statresults ['toy'] = toyid
        return statresults

class CategoryStatementsView(ListView):
    model = Statement
    template_name = 'categstatements.html'

    def get_queryset(self):
        categid = self.request.GET.get('category')
        stats = Statement.objects.filter(toy_id__category_id =
categid)

        stats = stats.filter(is_active = True)
        stats = stats.order_by('-created')
        return stats

    def get_context_data(self, **kwargs):
        statresults = super(CategoryStatementsView,
self).get_context_data(**kwargs)
        categid = self.request.GET.get('category')
        statresults ['category'] = categid
        return statresults

class CategoriesView(ListView):
    model = Category
    template_name = 'categories.html'
    ordering = ['ctitle']

class StatementContentView(LoginRequiredMixin, CreateView):
    model = Statement_content

```

```

        fields = ['statement_id', 'category_id', 'age', 'gender',
'is_certified', 'startproduced', 'endproduced', 'country', 'color',
'startpoint', 'endpoint', 'amount', 'about', 'together', 'brands',
'materials']
        template_name = 'statecontadd.html'

    def get_success_url(self):
        return reverse_lazy('statcontaddview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.category_id = get_object_or_404(Category,
id=self.kwargs['category_id'])
        self.statement_id = get_object_or_404(Statement,
id=self.kwargs['statement_id'])
        kwargs['category_id'] = self.category_id
        kwargs['statement_id'] = self.statement_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.category_id = get_object_or_404(Category,
id=self.kwargs['category_id'])
        self.statement_id = get_object_or_404(Statement,
id=self.kwargs['statement_id'])
        form.instance.category_id = self.category_id
        form.instance.statement_id = self.statement_id
        scont = form.save(commit = False)
        scont.save()
        scont.materials = None
        materials = str(self.request.POST.get('materials'))
        materials = matials.split(',')
        matlist = []
        for m in materials:
            matlist.append(Material.objects.(id = m))
        scont.materials.add(matlist)
        scont.save()
        scont.brands = None
        brands = str(self.request.POST.get('brands'))
        brands = brands.split(',')
        brandlist = []
        for br in brands:
            brandlist.append(Brand.objects.(id = br))
        scont.brands.add(brandlist)
        scont.save()

```

```

        form.save_m2m()
        return HttpResponseRedirect(self.get_success_url())

    class StatementContentUpdateView(LoginRequiredMixin,
UpdateView):
        model = Statement_content
        fields = ['statement_id', 'category_id', 'age', 'gender',
'is_certified', 'startproduced', 'endproduced', 'country', 'color',
'startpoint', 'endpoint', 'amount', 'about', 'together', 'brands',
'materials']
        template_name = 'statecontupdate.html'

        def get_success_url(self):
            return reverse_lazy('statcontupdateview',
kwargs={'pk': self.object.pk})

        def get_context_data(self, **kwargs):
            self.category_id = get_object_or_404(Category,
id=self.kwargs['category_id'])
            self.statement_id = get_object_or_404(Statement,
id=self.kwargs['statement_id'])
            kwargs['category_id'] = self.category_id
            kwargs['statement_id'] = self.statement_id
            return super().get_context_data(**kwargs)

        def form_valid(self, form):
            self.category_id = get_object_or_404(Category,
id=self.kwargs['category_id'])
            self.statement_id = get_object_or_404(Statement,
id=self.kwargs['statement_id'])
            form.instance.category_id = self.category_id
            form.instance.statement_id = self.statement_id
            scont = form.save(commit = False)
            scont.save()
            scont.materials = None
            materials = str(self.request.POST.get('materials'))
            materials = materials.split(',')
            matlist = []
            for m in materials:
                matlist.append(Material.objects.(id = m))
            scont.materials.add(matlist)
            scont.save()
            scont.brands = None
            brands = str(self.request.POST.get('brands'))
            brands = brands.split(',')

```

```

        brandlist = []
        for br in brands:
            brandlist.append(Brand.objects.(id = br))
        scont.brands.add(brandlist)
        scont.save()
        form.save_m2m()
        return HttpResponseRedirect(self.get_success_url())

class DamageUpdateView(LoginRequiredMixin, UpdateView):
    model = Damage
    fields = ['toy_id', 'about']
    template_name = 'damageupdate.html'

    def get_success_url(self):
        return reverse_lazy('damageupdateview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.toy_id = get_object_or_404(Toy,
id=self.kwargs['toy_id'])
        kwargs['toy_id'] = self.toy_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.toy_id = get_object_or_404(Toy,
id=self.kwargs['toy_id'])
        if self.kwargs['approve']:
            form.instance.approved = form.instance.approved +
1
        else:
            form.instance.unapproved = form.instance.approved
+ 1

        form.instance.toy_id = self.toy_id
        return super().form_valid(form)

class DamageAddView(LoginRequiredMixin, CreateView):
    model = Damage
    fields = ['toy_id', 'about']
    template_name = 'damageadd.html'

    def get_success_url(self):
        return reverse_lazy('damageaddview', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        self.toy_id = get_object_or_404(Toy,
id=self.kwargs['toy_id'])
        kwargs['toy_id'] = self.toy_id

    def form_valid(self, form):
        self.toy_id = get_object_or_404(Toy,
id=self.kwargs['toy_id'])
        form.instance.approved = 1
        form.instance.created = datetime.now
        user = self.request.user
        fulluser = ExchangerUser.objects.get(id = user.id)
        form.instance.author_id = fulluser
        return super().form_valid(form)

class StatementByToyRestrictionsSearchView(ListView):
    model = Statement
    template_name = 'stateownrestrtoysearch.html'
    paginate_by = 10

    def get_queryset(self):
        statementid = self.request.GET.get('statement')
        stat = Statement.objects.get(id = statementid)
        toycateg = stat.toy_id.category_id
        stats = Statement_content.objects.filter(category_id =
toycateg)

        stats = stats.filter(statement_id__is_active = True)
        stats = stats.filter(age__gte =
statementid.toy_id.startage)
        stats = stats.filter(Q(gender = 'N') | Q(gender =
stat.toy_id.gender))
        stats = stats.filter(is_certified =
stat.toy_id.is_certified)
        stats = stats.filter(startproduced__lte =
stat.toy_id.produced)
        stats = stats.filter(endproduced__gte =
stat.toy_id.produced)
        stats = stats.filter(country = stat.toy_id.country)
        stats = stats.filter(color = stat.toy_id.color)
        stats = stats.filter(startpoint__lte =
stat.toy_id.toymodel_id.point)
        stats = stats.filter(endpoint__gte =
stat.toy_id.toymodel_id.point)
        newstats = stats
        for s in stats:

```

```

        donemater = True
        for mater in stat.toy_id.materials:
            if not s.filter(materials__in = [mater.id]):
                donemater = False
        if s.brand_id not in stat.brands:
            donemater = False
        if not donemater:
            newstats.filter(id = s.id).delete()
        newstats = stats
        osts = Statement_content.objects.filter(statement_id =
statementid)
        newosts = osts
        res = True
        st = Statement.objects.filter(id__in = newosts)
        for o in osts:
            cures = True
            if stat.toy_id.toymodel_id.brand_id.id not in
o.brands:
                cures = False
            if stat.toy_id.category_id != o.category_id:
                cures = False
            if stat.toy_id.startage > o.age:
                cures = False
            if o.gender != 'N' and stat.toy_id.gender !=
o.gender:
                cures = False
            if o.is_certified != stat.toy_id.is_certified:
                cures = False
            if not (stat.toy_id.produced >= o.startproduced
and stat.produced <= o.endproduced):
                cures = False
            if o.country != stat.toy_id.country:
                cures = False
            if o.color != stat.toy_id.color:
                cures = False
            if not (stat.toy_id.toymodel_id.point >=
o.startpoint and stat.toy_id.toymodel_id.<= o.endpoint):
                cures = False
            if not stat.together and not cures:
                newosts.filter(id = o.id).delete()
            elif stat.together and not cures:
                st.filter(id = o.statement_id.id).delete()
        donemater = True
        for mater in stat.toy_id.materials:
            if not o.filter(materials__in = [mater.id]):

```

```

        donemater = False
        if not donemater:
            newosts.filter(id = o.id).delete()
        if not stat.together:
            st = Statement.objects.filter(id__in = newosts)
        return st

    def get_context_data(self, **kwargs):
        statresults = super(class
StatementByToyRestrictionsSearchView(ListView):
, self).get_context_data(**kwargs)
        stat = self.request.GET.get('statement')
        statresults ['statement'] = stat
        return statresults

```

Б.2 Текст файла models.py

```

from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils import timezone
from django.urls import reverse

GENDERS = [
    ('F', 'хлопчик'),
    ('M', 'дівчинка'),
    ('N', 'будь-який'),
]

class Category (models.Model):
    ctitle = models.CharField(max_length=100)
    about = models.TextField()
    catpic = models.FileField(upload_to='images/')

    def get_absolute_url(self):
        return reverse('categoryview', kwargs={'pk': self.pk})

class Toy (models.Model):
    totitle = models.CharField(max_length=255)
    about = models.TextField()
    category_id = models.ForeignKey(Category,
related_name='categorytoys', on_delete=models.CASCADE)
    startrecomage = models.SmallIntegerField(default = 0)
    endrecomage = models.SmallIntegerField(default = 18)

```

```

startage = models.SmallIntegerField(default = 0)
gender = models.CharField(
    max_length=1,
    choices=GENDERS,
    default='N',
)
toymodel_id = models.ForeignKey(Toymodel,
related_name='modeltoys', on_delete=models.CASCADE)
is_certified = models.BooleanField()
produced = models.SmallIntegerField()
country = models.SmallIntegerField()
color = models.SmallIntegerField()
toypic = models.FileField(upload_to='images/')
materials = models.ManyToManyField(Material, related_name =
'mattoys', db_table = 'toy_material')

    def __str__(self):
        return self.totitle + ' (' +
self.model_id.brand_id.ctitle + ')'

    def get_absolute_url(self):
        return reverse('toyview', kwargs={'pk': self.pk})

class Brand (models.Model):
    ctitle = models.CharField(max_length=30)
    about = models.TextField()

    def get_absolute_url(self):
        return reverse('toybrandview', kwargs={'pk': self.pk})

    def __str__(self):
        return self.ctitle

class Toymodel (models.Model):
    modtitle = models.CharField(max_length=50)
    about = models.TextField()
    brand_id = models.ForeignKey(Brand,
related_name='brandmodels', on_delete=models.CASCADE)
    point = models.DecimalField(max_digits = 2, decimal_places =
1, default = 0)

    def __str__(self):
        return self.modtitle

```

```

    def get_absolute_url(self):
        return reverse('toymodelview', kwargs={'pk': self.pk})

class Material (models.Model):
    mattitle = models.CharField(max_length=20)
    natural = models.BooleanField()
class Opinion (models.Model):
    customuser_id = models.ForeignKey(ExchangerUser,
related_name='useropinions', on_delete=models.CASCADE)
    created = models.DateTimeField()
    point = models.SmallIntegerField()
    statepoint = models.SmallIntegerField()
    toy_id = models.ForeignKey(Toy, related_name='toyopinions',
on_delete=models.CASCADE)
    detailopinion = models.TextField()

    def get_absolute_url(self):
        return reverse('opinionview', kwargs={'pk': self.pk})

    def save(self, *args, **kwargs):
        toym = Toymodel.objects.get(id =
self.toy_id.model_id.id)
        opinions = Opinion.objects.filter(toy_id__model_id =
self.toy_id.model_id.id)
        avpoint = 0
        q = opinions.count()
        for op in opinions:
            avpoint = avpoint + op.point / q
        toym.point = avpoint
        toym.save()
        self.created = datetime.now
        super(Opinion, self).save(*args, **kwargs)

class Damage (models.Model):
    about = models.TextField()
    toy_id = models.ForeignKey(Toy, related_name='toydamages',
on_delete=models.CASCADE)
    author_id = models.ForeignKey(Toy,
related_name='toydamages', on_delete=models.CASCADE)
    created = models.DateTimeField()
    approved = models.SmallIntegerField(default = 0)
    unapproved = models.SmallIntegerField(default = 0)

    def save(self, *args, **kwargs):
        self.created = datetime.now

```

```

        super(Damage, self).save(*args, **kwargs)

class ExchangerUser(AbstractUser):
    about = models.TextField()

    def __str__(self):
        return self.username

class Statement (models.Model):
    toy_id = models.ForeignKey(Toy,
related_name='toystatements', on_delete=models.CASCADE)
    exchangeruser_id = models.ForeignKey(ExchangerUser,
related_name='userstatements', on_delete=models.CASCADE)
    created = models.DateTimeField()
    is_active = models.BooleanField()
    together = models.BooleanField()
    city = models.CharField(max_length=50)

    def get_absolute_url(self):
        return reverse('statementview', kwargs={'pk':
self.pk})

    def __str__(self):
        return self.toy_id.totitle

    def save(self, *args, **kwargs):
        self.created = datetime.now
        super(Statement, self).save(*args, **kwargs)

class Statement_content (models.Model):
    statement_id = models.ForeignKey(Statement,
related_name='statementcontents', on_delete=models.CASCADE)
    category_id = models.ForeignKey(Category,
on_delete=models.CASCADE)
    brands = models.ManyToManyField(Brand, related_name =
'brandstatcont', db_table = 'statement_content_brand')
    age = models.SmallIntegerField(default = 0)
    gender = models.CharField(
        max_length=1,
        choices=GENDERS,
        default='N',
    )
    is_certified = models.BooleanField()
    startproduced = models.SmallIntegerField(default = 0)
    endproduced = models.SmallIntegerField(default = 0)

```

```

        country = models.SmallIntegerField()
        color = models.SmallIntegerField()
        startpoint = models.DecimalField(max_digits = 2,
decimal_places = 1, default = 0)
        endpoint = models.DecimalField(max_digits = 2,
decimal_places = 1, default = 0)
        amount = models.SmallIntegerField()
        about = models.TextField()
        materials = models.ManyToManyField(Material, related_name =
'matstatcont', db_table = 'statement_content_material')

```

```

class Proposition (models.Model):
    statement_id = models.ForeignKey(Statement,
related_name='statementpropositions', on_delete=models.CASCADE)
    exchangeruser_id = models.ForeignKey(ExchangerUser,
related_name='userpropositions', on_delete=models.CASCADE)
    created = models.DateTimeField()
    is_approved = models.BooleanField()
    comment = models.TextField()
    city = models.CharField(max_length=50)

    def get_absolute_url(self):
        return reverse('propositionview', kwargs={'pk':
self.pk})

    def save(self, *args, **kwargs):
        self.created = datetime.now
        super(Proposition, self).save(*args, **kwargs)

class Proposition_content (models.Model):
    proposition_id = models.ForeignKey(Proposition,
related_name='propositioncontents', on_delete=models.CASCADE)
    toy_id = models.ForeignKey(Toy, on_delete=models.CASCADE)

class Change (models.Model):
    proposition_id = models.ForeignKey(Proposition,
on_delete=models.CASCADE)
    approved = models.DateTimeField()
    comment = models.TextField()
    delivery = models.TextField()

    def get_absolute_url(self):
        return reverse('changeview', kwargs={'pk': self.pk})

```

```
def save(self, *args, **kwargs):
    self.created = datetime.now
    super(Change, self).save(*args, **kwargs)

class Message (models.Model):
    author_id = models.ForeignKey(ExchangerUser,
on_delete=models.CASCADE)
    receiver_id = models.ForeignKey(ExchangerUser,
on_delete=models.CASCADE)
    created = models.DateTimeField()
    messtext = models.TextField()

def save(self, *args, **kwargs):
    self.created = datetime.now
    super(Message, self).save(*args, **kwargs)
```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
кафедра програмних засобів

Програмне забезпечення обміну дитячими іграшками **Kids Toy Exchange Software**

Виконав студ. групи КНТ-129сп

І. К. Голубєв

Керівник:

Т. В. Голуб

2022

Рисунок В.1 – Слайд № 1

2

Об'єкт роботи – процес обміну дитячими іграшками між людьми.

Предмет роботи – програмне забезпечення обміну дитячими іграшками.

Мета роботи – розробити програмне забезпечення обміну дитячими іграшками для зменшення витрат людей.

Рисунок В.2 – Слайд № 2

Завдання роботи

3

- дослідження проблеми;
- визначення вимог до розробки;
- проєктування програмного забезпечення обміну дитячими іграшками;
- програмна реалізація програмного забезпечення обміну дитячими іграшками.

Рисунок В.3 – Слайд № 3

Вибір мови розробки програми

4

Критерій	C#	Python
Фреймворк розробки вебзастосунків	ASP.NET	Django, Flask
Типізація	Статична	Динамічна
Придатність для швидкої розробки	Середня	Висока
Відкрита ліцензія	Так	Так

Рисунок В.4 – Слайд № 4

Діаграми прецедентів

5

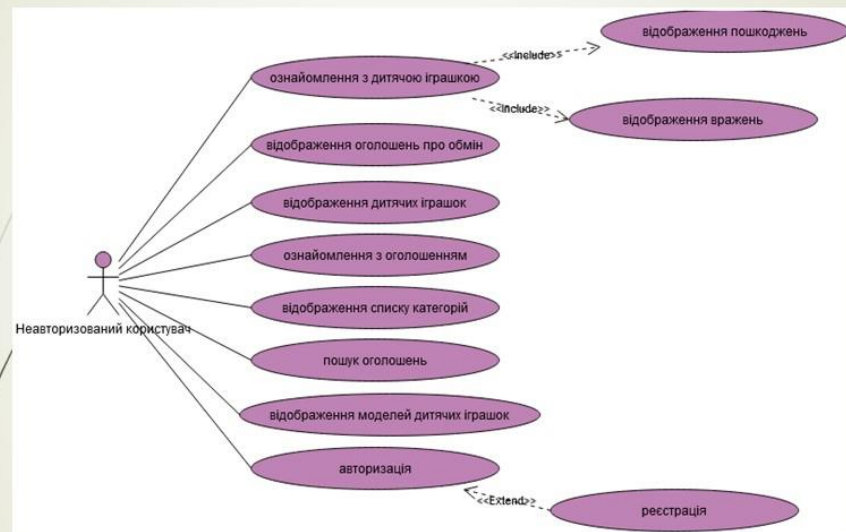


Рисунок В.5 – Слайд № 5

Діаграми прецедентів

6

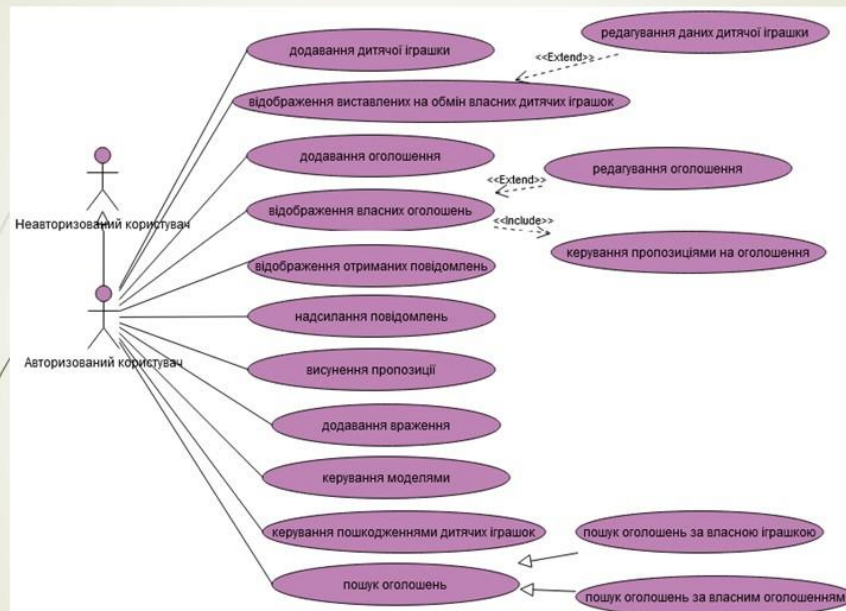


Рисунок В.6 – Слайд № 6

Схема бази даних

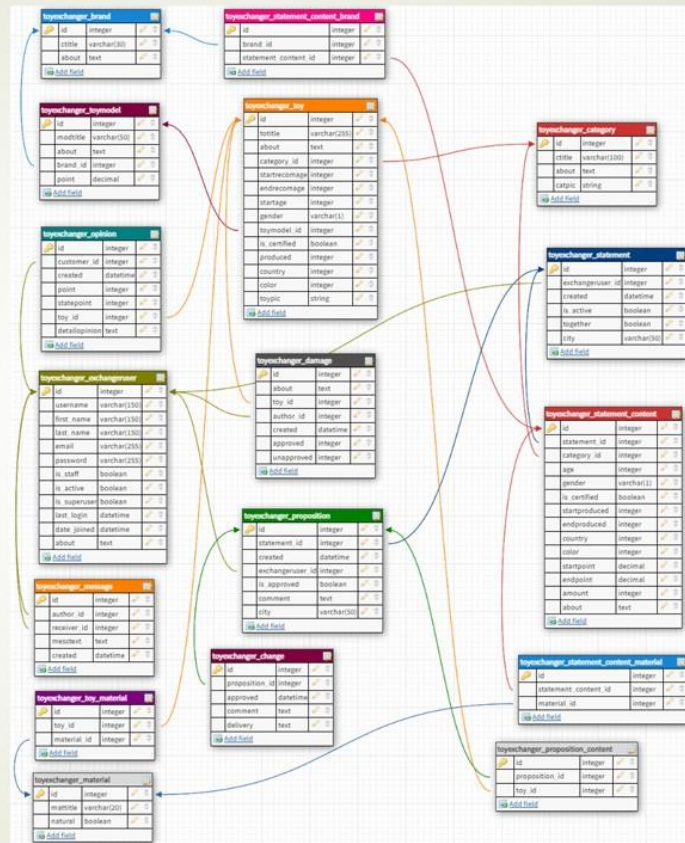


Рисунок В.7 – Слайд № 7

Діаграма послідовності роботи над обміном дитячих іграшок

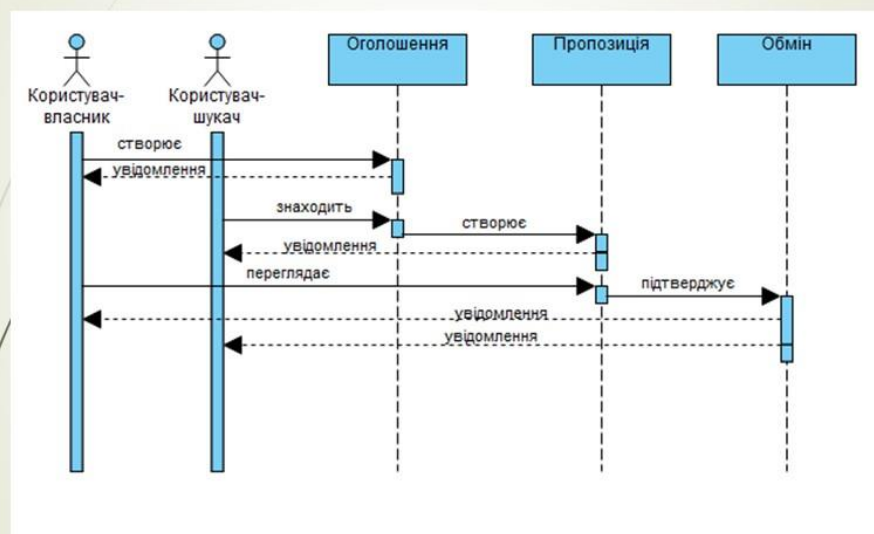


Рисунок В.8 – Слайд № 8

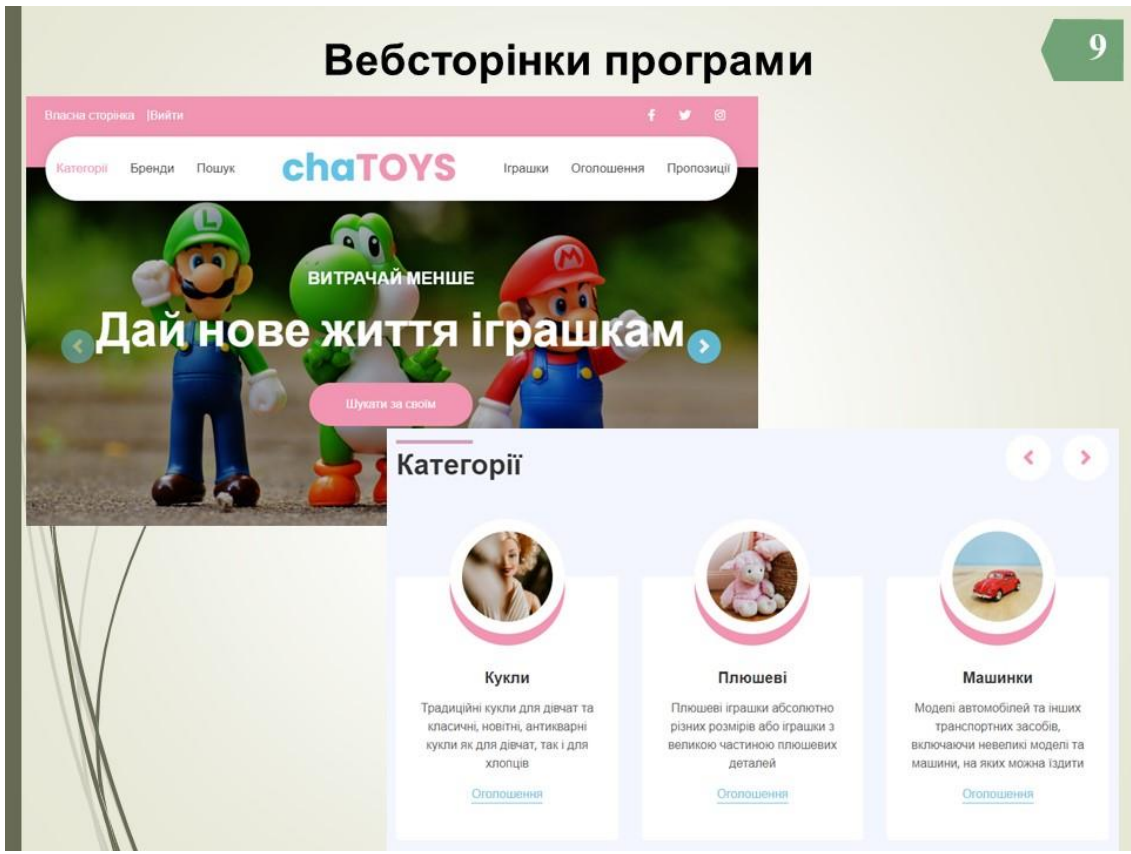


Рисунок В.9 – Слайд № 9

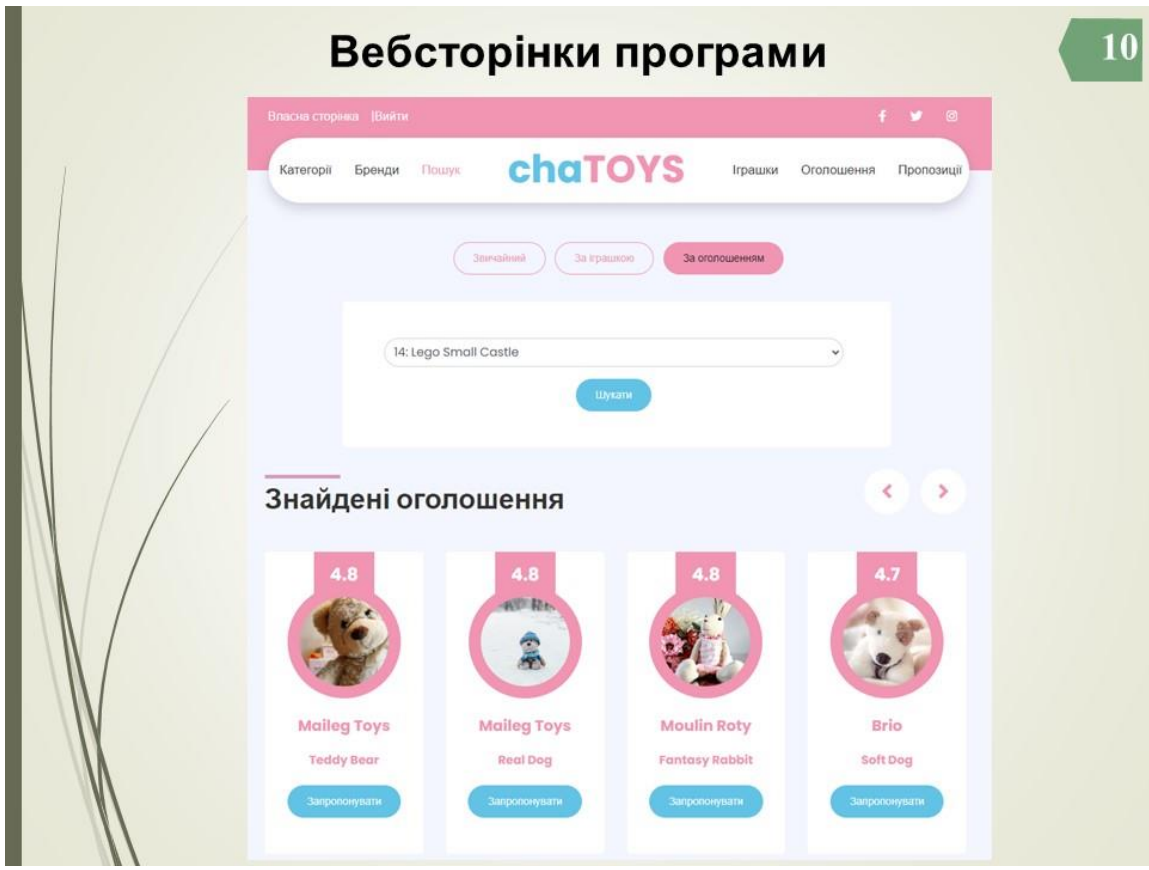


Рисунок В.10 – Слайд № 10

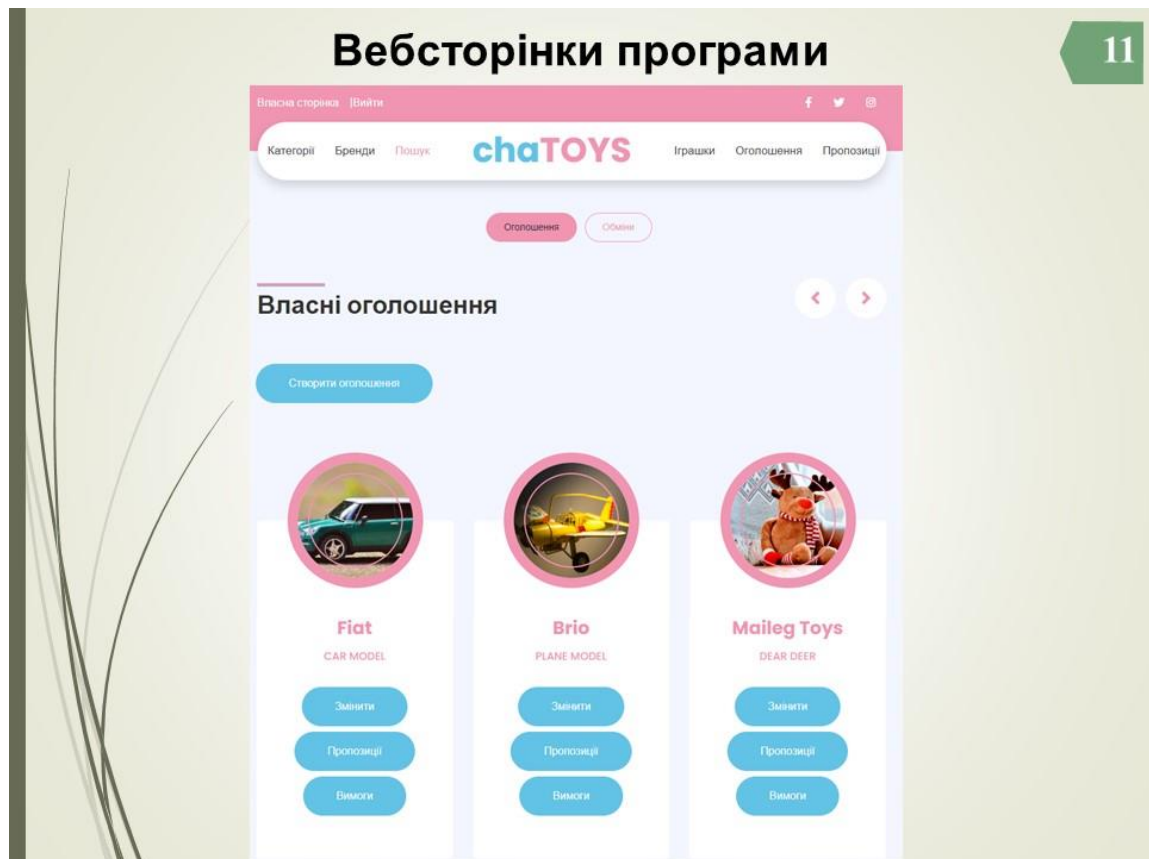


Рисунок В.11 – Слайд № 11

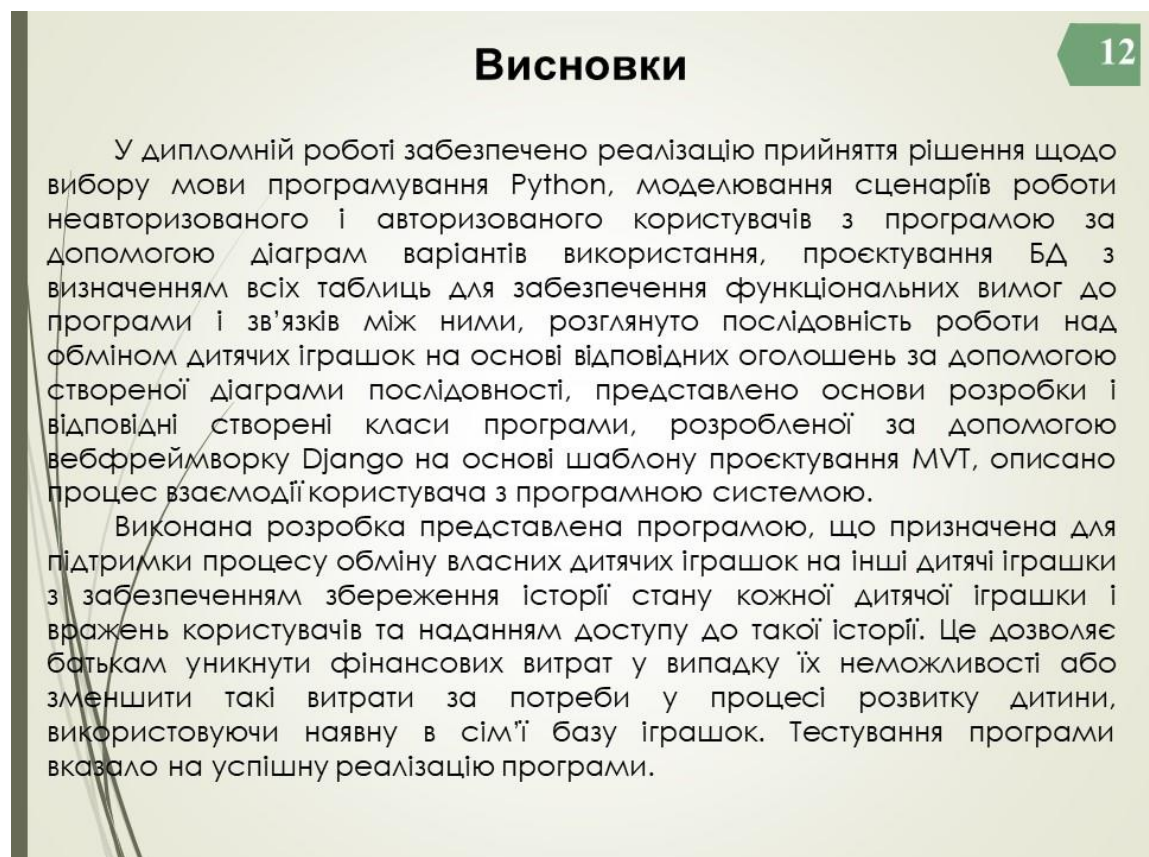


Рисунок В.12 – Слайд № 12