**Design Patterns Used**

| **Pattern** | **Where Used** | **Why / How** |
| --- | --- | --- |
| Factory Method | models.py (UserManager), forms.py | Централізоване створення користувачів та форм; спрощує розширення та контроль над логікою створення об'єктів |
| Adapter | adapters.py (MySocialAccountAdapter), admin.py (CustomUserAdmin) | Узгодження інтерфейсу бібліотеки allauth і кастомної адмінки під потреби проекту |
| Strategy | settings.py (AUTHENTICATION\_BACKENDS) | Можливість легко змінити спосіб автентифікації (логін/соцмережі) без зміни основної логіки проекту |
| Template Method | views.py (CustomLoginView) | Наслідування стандартного LoginView з перевизначенням лише потрібних кроків (DRY) |
| Decorator | views.py (@login\_required) | Контроль доступу (авторизація) |

**Принципи дизайну (Design Principles)**

**DRY (Don't Repeat Yourself):**

Вся логіка пошуку, реєстрації, роботи з формами та моделями — централізована, відсутнє дублювання коду (див. views.py, forms.py).

**Single Responsibility Principle:**

Кожна форма (forms.py), модель (models.py) і view (views.py) відповідає лише за одну логіку.

**Open/Closed Principle:**

Легко додати нові способи автентифікації, нові поля до моделей, не змінюючи старий код.

**KISS (Keep It Simple, Stupid):**

Кожен view-функція/клас максимально проста, не робить зайвого, легко зрозуміти навіть новому розробнику.

**Де подивитись патерни в коді?**

Factory Method:

* models.py → клас UserManager, методи create\_user, create\_superuser
* forms.py → Meta клас і методи форми

Adapter:

* [adapters.py](reg/adapters.py) → клас MySocialAccountAdapter
* admin.py → клас CustomUserAdmin

Strategy:

* settings.py → змінна AUTHENTICATION\_BACKENDS

Template Method:

* views.py → клас CustomLoginView, метод form\_valid

**Методи рефакторингу, використані у проекті:**

1. *Винесення дубльованого коду (Extract Constant/Variable)*

Однакові рядки ("Пароль", "Повторіть пароль") винесені в константи у forms.py.

Стилі, base.html та інші повторювані блоки — у спільні шаблони (templates) й константи.

1. *Винесення спільної логіки в базові класи/методи (Extract Method/Class)*

Менеджер UserManager (Factory Method) для створення користувача і суперкористувача.

Базові форми через Meta-класи.

1. *Перейменування (Rename)*

Зрозумілі назви методів, змінних, класів (замість test1, test2 — осмислені імена).

Чіткі імена полів, щоб відразу зрозуміти їх суть.

1. *Інкапсуляція логіки (Encapsulate Field/Method)*

Валідація і логіка форми (clean\_xxx) інкапсульовані у форму, а не розкидані по views.

1. *Видалення "мертвого" коду (Remove Dead Code)*

Всі зайві імпорти, неактуальні функції/класи прибрані (unused import, unused arguments).

Міграції і зайві файли не аналізуються.

1. *Винесення налаштувань у .env (Extract Configuration)*

SECRET\_KEY та інші “чутливі” дані винесені у .env, а не захардкожені в settings.py.

1. *Винесення повторюваних шаблонів (Template Extraction)*

base.html та повторювані частини інтерфейсу винесено у загальні шаблони (DRY).

1. *Заміна жорстких шляхів на змінні/резолвери (Replace Hard-Coded Reference)*

Використання {% url %} для формування шляхів у шаблонах.

**Класичні патерни рефакторингу (Martin Fowler):**

* Extract Method
* Extract Class
* Rename Method/Variable
* Remove Dead Code
* Move Method/Field (наприклад, логіку з views у форми)
* Inline Variable (там, де треба — навпаки спростила код)

**Популярні code smells у Django-проектах:**

Duplicated code — однакові рядки або фрагменти коду в різних місцях.

Long function/line — дуже довгі функції або рядки (>100 символів).

Large class — занадто багато логіки в одному класі.

Dead code — не використані змінні, імпорти, функції.

Hard-coded values — захардкожені дані (наприклад, SECRET\_KEY у settings.py).

God object — один клас/файл, що “робить усе”.

Long parameter list — функції з купою параметрів.

Feature envy — коли функція занадто часто звертається до полів інших класів.

Lack of encapsulation — важлива логіка розкидана по проекту, не схована у класах/методах.

**Які code smells були у мене:**

*1. Дублювання рядків*

* "Пароль", "Повторіть пароль" у forms.py

Виправлено: винесено у константи.

*2. Захардкожені секрети*

* 'SECRET\_KEY = ...' у settings.py

Виправлено: винесено у .env.

*3. Unused imports*

* import tempfile у tests.py

Виправлено: видалено.

*4. Довгі рядки*

* Деякі рядки >100 символів

Можна залишати (не критично для Django).

*5. Відсутність docstrings у класах/методах*

Це не критично, але Pylint сварився (можна ігнорити).

**У моєму проекті дотримано всі основні принципи SOLID**

*1. S — Single Responsibility Principle (Принцип єдиної відповідальності)*

Кожен клас і кожен файл має чітко визначену відповідальність. Наприклад:

* forms.py — тільки форми
* models.py — тільки моделі
* views.py — тільки бізнес-логіка відображення
* adapters.py — лише адаптація логіки логіну

*2. O — Open/Closed Principle (Відкритість/закритість)*

Можна розширювати функціонал через нові форми, моделі, view-класи без зміни старого коду. Нові поля у User, нові типи авторизації — легко додаються через налаштування, не ламаючи існуючий код.

*3. L — Liskov Substitution Principle (Принцип підстановки Лісков)*

Всі дочірні класи (наприклад, кастомні форми, CustomLoginView) можна підставити замість батьківських (UserCreationForm, LoginView) — і система працює коректно. Наслідування Django views та forms дотримується цього принципу.

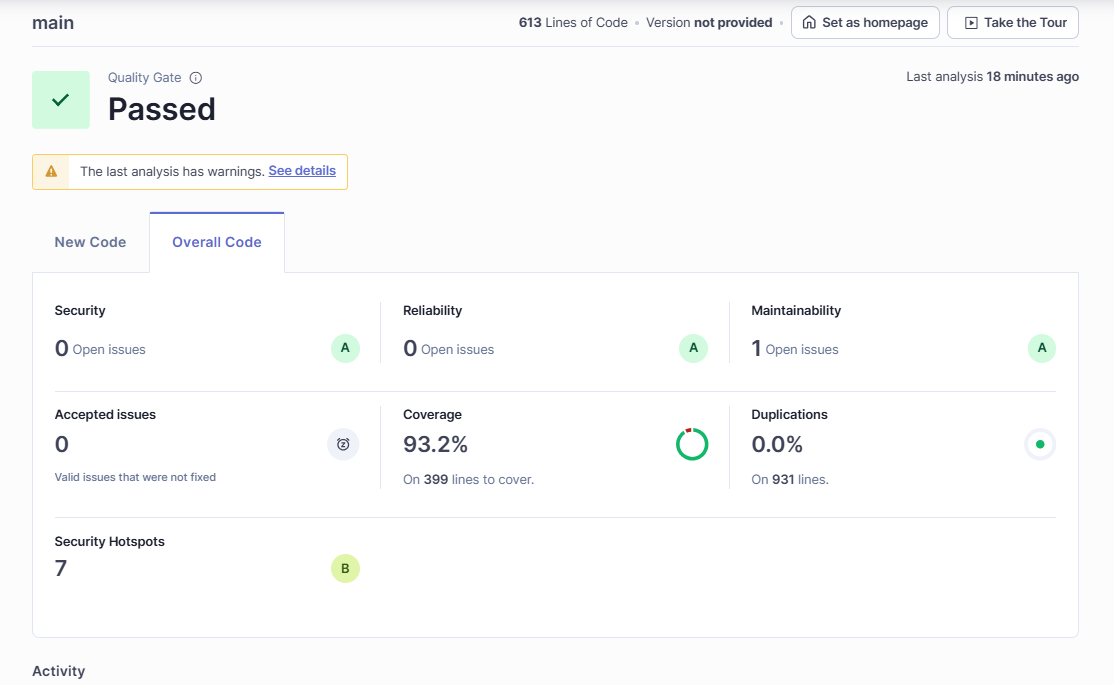
*4. I — Interface Segregation Principle (Принцип розділення інтерфейсів)*

Кожен клас/форма/в’юшка використовує лише ті методи/поля, які йому потрібні, нічого зайвого. Наприклад, окремі форми для реєстрації, логіну, редагування профілю.

*5. D — Dependency Inversion Principle (Інверсія залежностей)*

Всі залежності на зовнішні сервіси (логін через Google, параметри з .env, бекап через allauth) — підключаються через налаштування (settings.py, adapters.py), а не жорстко “зашиті” у коді. Весь конфіг через settings.py/.env, імплементації легко змінити без зміни основного коду.

**Певевірка якості коду**



\*\*\*\*\*\*\*\*\*\*\*\*\* Module reg.adapters

reg\adapters.py:12:37: W0613: Unused argument 'request' (unused-argument)

\*\*\*\*\*\*\*\*\*\*\*\*\* Module reg.forms

reg\forms.py:15:0: C0301: Line too long (101/100) (line-too-long)

reg\forms.py:28:4: R0903: Too few public methods (0/2) (too-few-public-methods)

reg\forms.py:52:4: R0903: Too few public methods (0/2) (too-few-public-methods)

reg\forms.py:61:4: R0903: Too few public methods (0/2) (too-few-public-methods)

reg\forms.py:82:4: R0903: Too few public methods (0/2) (too-few-public-methods)

reg\forms.py:102:4: R0903: Too few public methods (0/2) (too-few-public-methods)

\*\*\*\*\*\*\*\*\*\*\*\*\* Module reg.models

reg\models.py:75:4: R0903: Too few public methods (0/2) (too-few-public-methods)

reg\models.py:108:4: R0903: Too few public methods (0/2) (too-few-public-methods)

reg\models.py:134:4: R0903: Too few public methods (0/2) (too-few-public-methods)

\*\*\*\*\*\*\*\*\*\*\*\*\* Module reg.tests

reg\tests.py:98:0: C0301: Line too long (101/100) (line-too-long)

reg\tests.py:33:19: E1101: Class 'Pin' has no 'objects' member (no-member)

reg\tests.py:36:25: E1101: Class 'Pin' has no 'objects' member (no-member)

reg\tests.py:46:19: E1101: Class 'Pin' has no 'objects' member (no-member)

reg\tests.py:47:21: E1101: Class 'Board' has no 'objects' member (no-member)

reg\tests.py:51:25: E1101: Class 'Board' has no 'objects' member (no-member)

reg\tests.py:123:19: E1101: Class 'Pin' has no 'objects' member (no-member)

reg\tests.py:163:24: E1101: Class 'Pin' has no 'objects' member (no-member)

\*\*\*\*\*\*\*\*\*\*\*\*\* Module reg.views

reg\views.py:18:15: E1101: Class 'Pin' has no 'objects' member (no-member)

reg\views.py:20:15: E1101: Class 'Pin' has no 'objects' member (no-member)

reg\views.py:74:18: W0613: Unused argument 'request' (unused-argument)

------------------------------------------------------------------

Your code has been rated at 8.48/10 (previous run: 8.45/10, +0.03)

**Як тестувати проект**

python manage.py test

Тести для перевірки реєстрації, логіну, створення піна — у файлі reg/tests.py.

**Тести**

test\_board\_creation (reg.tests.BoardModelTest.test\_board\_creation) ... ok

test\_str\_method (reg.tests.BoardModelTest.test\_str\_method) ... ok

test\_edit\_profile\_valid (reg.tests.EditProfileFormTest.test\_edit\_profile\_valid) ... ok

test\_pin\_form\_pic\_invalid\_type (reg.tests.PinFormTest.test\_pin\_form\_pic\_invalid\_type) ... ok

test\_pin\_form\_pic\_too\_large (reg.tests.PinFormTest.test\_pin\_form\_pic\_too\_large) ... ok

test\_pin\_form\_valid (reg.tests.PinFormTest.test\_pin\_form\_valid) ... ok

test\_pin\_creation (reg.tests.PinModelTest.test\_pin\_creation) ... ok

test\_str\_method (reg.tests.PinModelTest.test\_str\_method) ... ok

test\_signup\_form\_invalid\_email (reg.tests.SignUpFormTest.test\_signup\_form\_invalid\_email) ... ok

test\_signup\_form\_invalid\_username (reg.tests.SignUpFormTest.test\_signup\_form\_invalid\_username) ... ok

test\_signup\_form\_valid (reg.tests.SignUpFormTest.test\_signup\_form\_valid) ... ok

test\_str\_method (reg.tests.UserModelTest.test\_str\_method) ... ok

test\_superuser\_creation (reg.tests.UserModelTest.test\_superuser\_creation) ... ok

test\_user\_creation (reg.tests.UserModelTest.test\_user\_creation) ... ok

test\_create\_pin\_logged (reg.tests.ViewAccessTest.test\_create\_pin\_logged) ... ok

test\_create\_pin\_post (reg.tests.ViewAccessTest.test\_create\_pin\_post) ... ok

test\_login\_post (reg.tests.ViewAccessTest.test\_login\_post) ... ok

test\_login\_view\_get (reg.tests.ViewAccessTest.test\_login\_view\_get) ... ok

test\_logout (reg.tests.ViewAccessTest.test\_logout) ... ok

test\_mainpage\_view (reg.tests.ViewAccessTest.test\_mainpage\_view) ... ok

test\_pin\_detail\_view (reg.tests.ViewAccessTest.test\_pin\_detail\_view) ... ok

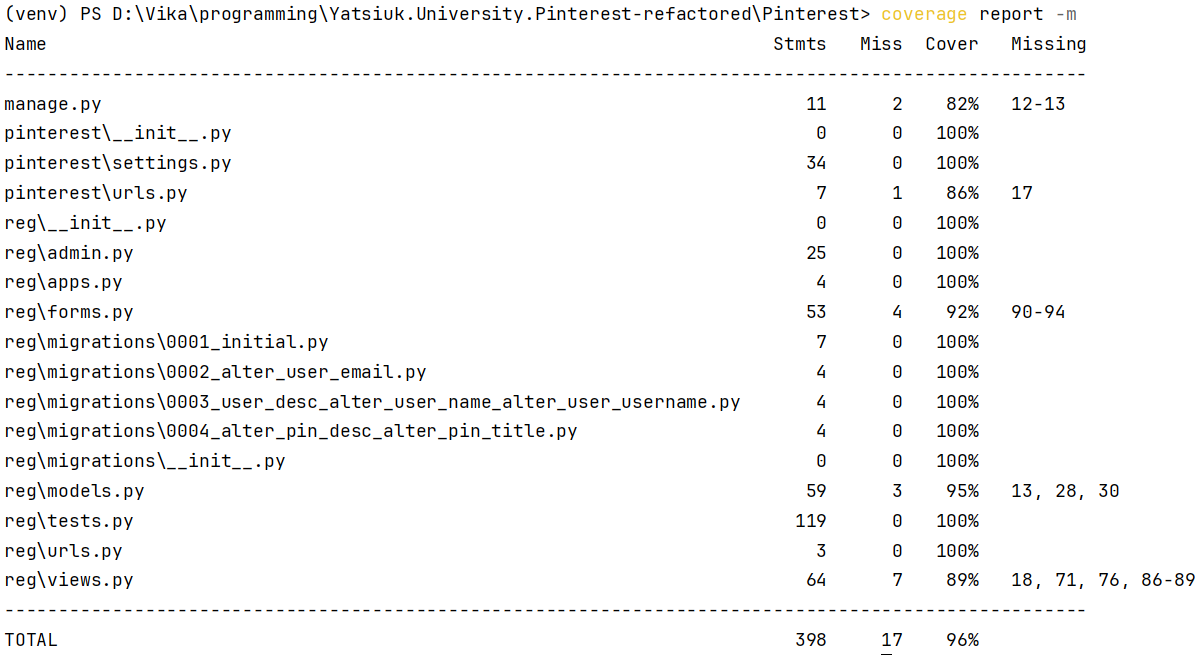
test\_register\_post (reg.tests.ViewAccessTest.test\_register\_post) ... ok

test\_register\_view\_get (reg.tests.ViewAccessTest.test\_register\_view\_get) ... ok

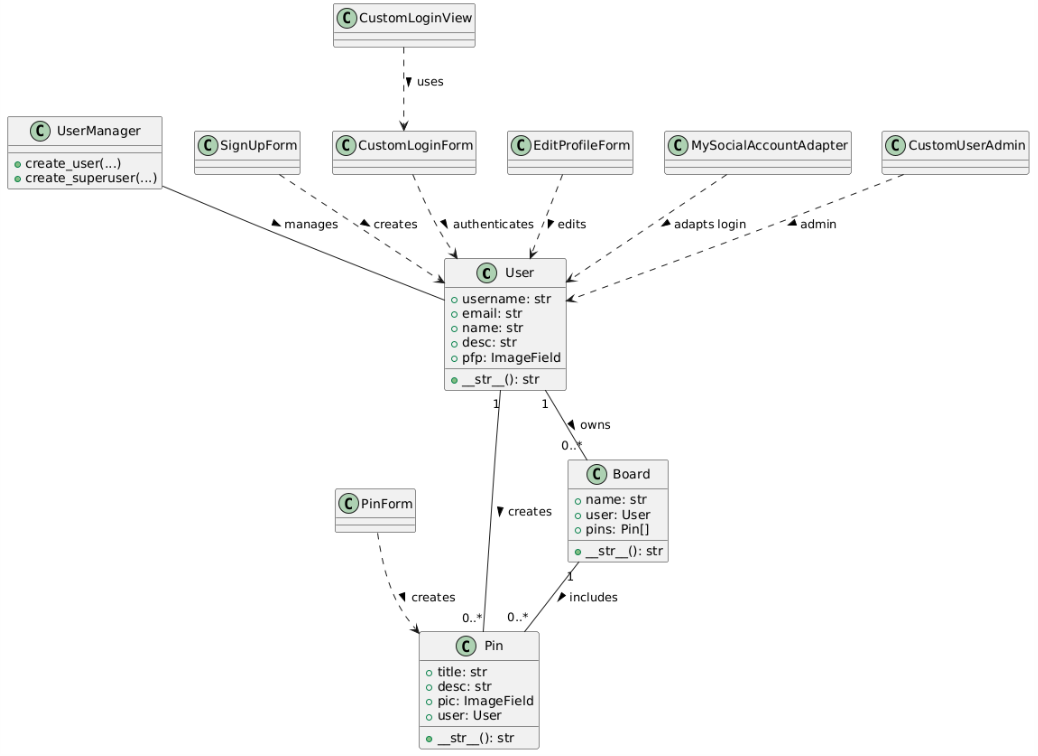
test\_userprofile\_logged (reg.tests.ViewAccessTest.test\_userprofile\_logged) ... ok

test\_userprofile\_redirect\_if\_not\_logged (reg.tests.ViewAccessTest.test\_userprofile\_redirect\_if\_not\_logged) ... ok

Ran 25 tests in 15.842s



**Діаграма UML**



В центрі діаграми — модель User:

* User має
* 6+
* + власні атрибути (username, email, name, desc, pfp).
* User керується менеджером UserManager (патерн Factory Method).
* User авторизується через CustomLoginForm та CustomLoginView (Template Method, Strategy).
* User створюється через SignUpForm, редагується через EditProfileForm.

Pin (пости/піни):

* Кожен Pin належить User і створюється через PinForm.
* User може створити багато Pin (звʼязок 1 до багато).

Board (дошка):

* Кожна Board належить User і включає набір Pin-ів (звʼязок 1 до багато).
* Board, Pin, User повʼязані асоціаціями.

Адміністрування та авторизація:

* CustomUserAdmin керує User в адмінці (Adapter).
* MySocialAccountAdapter адаптує логіку логіну через соціальні мережі.

Всі взаємозвʼязки і підписи відображають реальні залежності проекту:

* “creates”, “owns”, “authenticates”, “edits”, “manages”, “adapts login”, “admin”.