4. КОНСТРУЮВАННЯ ПРОГРАМНОГО ПРОДУКТУ

4. 1 Особливості конструювання структур даних

Архітектура програмного забезпечення ‒ це представлення системи програмного забезпечення, яке дає інформацію про складові компоненти системи, про взаємозв'язки між цими компонентами і правила, що регламентують ці взаємозв'язки. Це процес, що передбачає послідовність дій для створення або зміни архітектури системи, і проекту системи по цій архітектурі, з врахуванням безлічі обмежень.

4.1.1 Особливості інсталяції та роботи з СУБД

Розглянуто створення бази даних для роботи рекламного агентства з використанням СУБД PostgreSQL.

До створеної бази даних були створені тригери та функції для маніпулювання даними, занесена демонстраційна інформація. Також були розроблені запити для демонстрації можливостей створеної бази даних. Архітектура бази даних передбачає підключення її до комп’ютерного додатку чи веб-сайту через використання користувачів. Така система дозволяє обмежувати привілеї для захисту даних.

PostgreSQL називають найбільш досконалою з наявних сьогодні СУБД з відкритим вихідним кодом. Таку репутацію вона завоювала завдяки зусиллям розробників протягом десятиліть. Будучи повнофункціональної реляційної СУБД з відкритим вихідним кодом, PostgreSQL володіє багатьма якостями, необхідними для підтримки найважливіших додатків з великим числом транзакцій.

Головне достоїнство PostgreSQL полягає в надійному і захищеному зберіганні даних, якими вона управляє. Починаючи з того, що підключення до бази даних контролюються за допомогою корпоративних механізмів аутентифікації, таких як LDAP або Kerberos. Після аутентифікації весь обмін інформацією з СУБД може здійснюватися по зашифрованим з допомогою Secure Sockets Layer (SSL) підключень, що забезпечує надійний захист даних.

При додаванні або модифікації даних PostgreSQL накладає ряд обмежень, введених користувачем для забезпечення якості даних відповідно до бізнес-правилами. Діапазон цих правил простягається від простих перевірок до більш складних, пов'язаних з наявністю зовнішнього ключа.

Обов’язково потрібно позначити дуже важливу перевагу СУБД PostgreSQL – безліч типів даних, які дозволяють зберігати цифрові значення, строки, тексти, дату, час, географічні координати, ip-адреса тощо. Завдяки цьому програміст бази даних не повинен втрачати свій час на створення нових типів та механізмів взаємодії. У переважній більшості, при розробці будь-якої бази, PostgreSQL вже буде мати необхідні типи.

Діаграма баз даних є структурою частиною системи баз даних описаною формальною мовою, яка підтримується системою керування баз даних і відноситься до організації даних з розподілом на таблиці.

Розглянемо SQL-запити для створення таблиці тварин:

CREATE SEQUENCE IF NOT EXISTS animals\_ids; CREATE TABLE IF NOT EXISTS pets (id int PRIMARY KEY DEFAULT nextval('animals\_ids'), animal\_id int,

age int NOT NULL CHECK (salary > 0),

breed varchar,

description varchar,

name varchar NOT NULL,

photo varchar,

FOREIGN KEY (shelter\_id) REFERENCES shelters (id),

FOREIGN KEY (user\_id) REFERENCES users (id);

Таблиця об’єднує інформацію про тварину, притулки, та користувачів через додавання первісних ключів цих таблиць як зовнішніх для таблиці вакансій.

Для зберігання ключів використовується тип даних integer, бо операції порівняння та пошуку швидше за все працюють з цифровими типами. Таким чином кожен елемент таблиці може бути знайдений за лінійний час роботи алгоритму пошуку. Формування унікальних ідентифікаторів робиться через елемент sequence відповідно до архітектури PostgreSQL.

Діаграму бази даних системи інформаційної системи для допомоги дитячим будинкам наведено на рисунку 4.1.1.

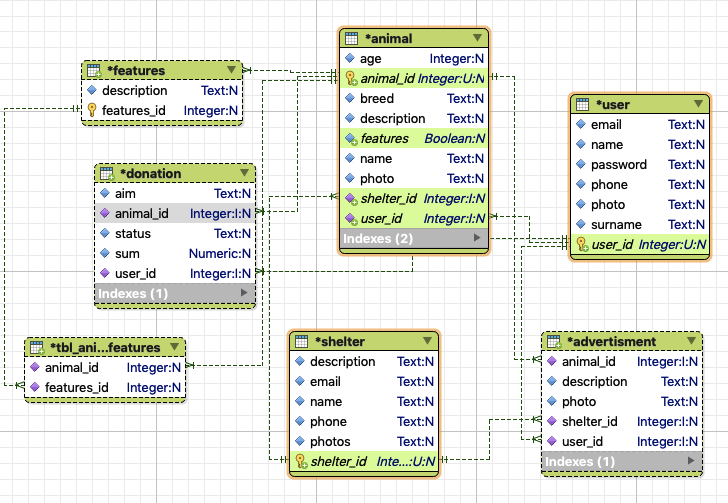


Рисунок 4.1.1 – Діаграма бази даних

База даних складається з наступних таблиць: Тварини, Користувачі, Об’яви, Притулки, Пожертвування, Ознаки та допоміжної таблиці Тварина\_Ознака, що забезпечує зв’язок «багато до багатьох» між таблицями «Тварини» та «Ознаки».

Таблиця «Користувачі» містить інформацію про усіх користувачів системи. У ній присутні такі поля, як унікальний ідентифікатор анкети (первинний ключ), ім’я, прізвище, номер телефону, фото, пароль та пошта (ім’я користувача) для входу у систему.

Таблиця «Тварини» містить інформацію про анкету тварини. У ній присутні такі поля, як унікальний ідентифікатор анкети, ім’я тварини, її вік, код анкети (первинний ключ), ознаки, фото тварини, код притулку (вторинний ключ), код користувача (вторинний ключ).

Таблиця «Пожертвування» містить інформацію про пожертвування дітям. У ній присутні такі поля, як ім’я та прізвище спонсора, напрямок пожертвування, сума пожертвування або інша його фіксація, статус виконання пожертвування організацією, дата надходження пожертвування та дата виконання організацією, код тварини (вторинний ключ), ідентифікатор користувача (вторинний ключ).

Таблиця «Притулки» містить інформацію по конкретному притулку. У ній присутні такі поля, як унікальний ідентифікатор притулку(первинний ключ), опис, пошта, телефон та фото.

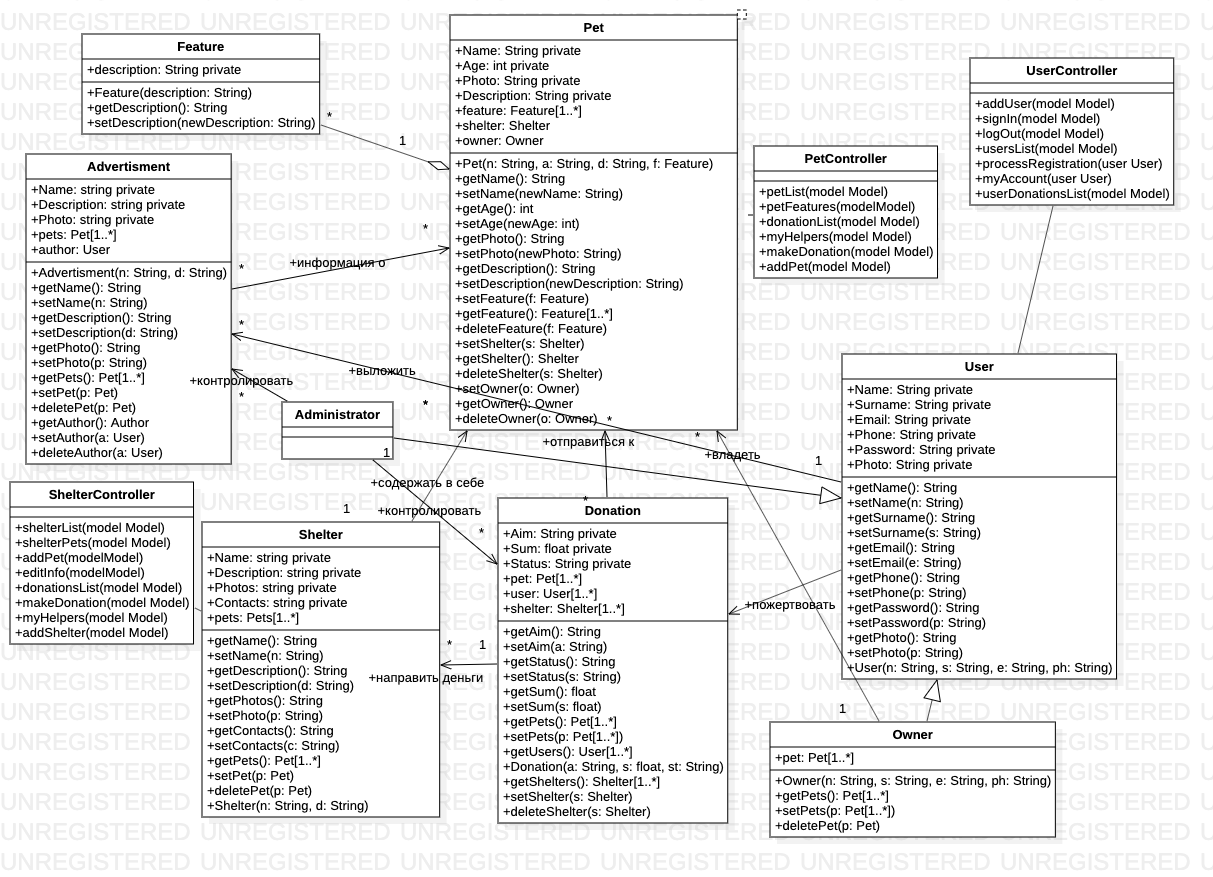
Таблиця «Об’ява» зберігає в собі інформацію про об’яву: унікальний ідентифікатор (первинний ключ), фото, опис, ідентифікатор користувача (вторинний ключ), ідентифікатор тварини (вторинний ключ), ідентифікатор притулку (вторинний ключ).

4.1.2 Особливості створення структур даних

У найпростішому випадку, процес проектування архітектури зводиться до складання статичних і динамічних UML-діаграм.

Діаграма в UML – це графічне представлення набору елементів, найчастіше зображеного у вигляді зв'язного графа вершин (сутностей) і дуг (зв'язків). Діаграма служить для візуалізації системи з різних точок зору, тому окрема діаграма – це проекція системи.

У даній роботі використано такі структури даних, як програмні класи, тому нижче наведено UML діаграму програмних класів продукту.



*Рисунок 4.1 – UML-діаграма програмних класів*

4.1.3 Висновки до підрозділу

За результатами даного розділу була спроектована система, що розробляється в ході даної курсової роботи. Були виділені і описані найбільш важливі класи, встановлені зв'язки між цими класами, визначено їх основні властивості. Наступною стадією розробки є програмна реалізація спроектованої системи.

4.2 Особливості конструювання структур даних

Для реалізації структур, описаних у розділі вище, було обрано створення веб-застосунку, де і буде реалізовано структури даних за допомогою програмних класів.

4.2.1 Особливості роботи з інтегрованим середовищем розробки

Інформаційна система «Лапка допомоги» для допомоги притулкам представляє собою клієнт-серверне застосування.

Для реалізації серверної частини спроектованої системи була обрана мова об’єктно-орієнтованого програмування Java із використанням фреймворку Spring MVC. Спираючись на шаблон модель-уявлення-контролер, фреймворк Spring MVC допомагає будувати веб-додатки, так само гнучкі і слабо пов'язані, як сам фреймворк Spring.

Фронт-енд системи реалізований з використанням фреймворку Bootstrap, який включає в себе HTML- і CSS- шаблони оформлення різних компонентів веб-інтерфейсу.

Для реалізації аутентифікації і авторизації в проекті використаний фреймворк Spring Security, який забезпечує безпеку інформації, надає можливість декларативного управління безпекою додатків на основі фреймворку Spring.

Всі дані в створеній системі зберігаються в базі даних PostgreSQL, яка розгорнута на хмарної PaaS-платформі Heroku.

Розглянемо принцип функціонування системи на прикладі роботи зі списком анкет дітей (має функції додавання нової анкети, видалення анкети, редагування анкети, отримання списку анкет дітей), що відбувається по схемі «Model (сутність) – View (представлення) – Controller (контролер)».

4.2.2 Особливості створення програмної структури з урахуванням спеціалізованого Фреймворку

Моделлю у даній системі служить клас AnimalService, який дозволяє працювати з базою даних за допомогою використання Spring Data JPA.

package com.example.demo.animal;

@Service

public class AnimalService {

private AnimalRepository repository;

@Autowired

public void setRepository(AnimalRepository repository) {

this.repository = repository;

}

public void saveAnimal(Animal animal) {

repository.save(animal);

}

public void addAnimal(Animal animal) {

repository.save(animal);

}

Public Animal getChildById(long id)

{

return repository.getOne(id);

}

public List<Animal> getAllAnimals () {

return repository.findAll();

}

public void deleteAnimal (long id) {

repository.deleteById(id);

}

}

4.2.3 Особливості створення програмних класів

Сутність (Entity) – це Java-клас, який представляє бізнес-логіку програми та визначає дані, які будуть зберігатися в базі даних і вилучатись з неї. Зазвичай клас суті представляє таблицю в базі даних, поля або властивості класу являють собою колонки в таблиці, а об'єкт суті являє собою один запис у таблиці.

Нижче описана сутність Animal – клас, який містить поля для описання тварини та методи, для роботи з ними (getters/setters).

package com.example.demo.animal;

@Entity

@Table(name = "animal")

public class Child {

public enum Gender {

male ("мужской"),

female ("женский");

private String name;

Gender(String name) {

this.name = name;

}

@Override

public String toString() {

return this.name;

}

}

@Id

@GeneratedValue

private long id;

@Column(name = " name")

@Size(min = 2, max= 50, message = "First name should be from 2 to 50 characters")

private String name;

@Column(name = "year\_of\_birth")

@Range(min = 2000, max= 2020, message = "Year of birth should be from 2000 to 2020")

private int yearOfBirth;

@Column(name = "gender")

private Gender gender;

@Size(min = 4, max= 6, message = "Code should be from 4 to 6 characters")

@Column(name = "code")

private String code;

@Size(min = 2, max= 50, message = "Hobby should be from 2 to 50 characters")

@Column(name = "age")

private int age;

@Column (name = "photo\_name")

private String photoName;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getName() {

return name;

}

public int getYearOfBirth() {

return yearOfBirth;

}

public void setYearOfBirth(int yearOfBirth) {

this.yearOfBirth = yearOfBirth;

}

public Gender getGender() {

return gender;

}

public void setGender(Gender gender) {

this.gender = gender;

}

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getPhotoName() {

return photoName;

}

public void setPhotoName(String photoName) {

this.photoName = photoName;

}

public int getAge() {

return 2020-this.yearOfBirth;

}

public void setAge(int age) {

this.age = age;

}

4.2.4 Особливості розробки алгоритмів методів програмних класів або процедур/функцій

Контролер - це компонент Spring, що обробляє запит. Він дозволяє компонентам Spring MVC впроваджуватися і вільно взаємодіяти шляхом запуску і обробки подій.

Нижче описаний контролер AnimalController, що містить безпосередньо весь функціонал роботи з запитами щодо анкет тварин. Так як контролер – це обробник клієнтських запитів, він містить такі методи, як отримання списку анкет, додання нової анкети, видалення та редагування існуючих та інші. У своїй реалізації він працює з сервісами.

package com.example.demo.admin;

@Controller

public class AnimalController {

private AnimalService service;

private UserService userService;

private DonationService donationService;

private MyUserDetailsService userDetailsService;

private FavouriteService favouriteService;

private CompanyService companyService;

@Autowired

public void setUserDetailsService(MyUserDetailsService userDetailsService) {

this.userDetailsService = userDetailsService;

}

@Autowired

public void setService(AnimalService service) {

this.service = service;

}

@Autowired

public void setUserService(UserService service) {

this.userService = service;

}

@Autowired

public void setFavouriteService(FavouriteService service) {

this.favouriteService = service;

}

@GetMapping("/admin/child\_list")

public String adminChildList(Model model) {

model.addAttribute("children", service.getAllChildren());

return "admin/child\_list";

}

@GetMapping("/admin/child")

public String processAddStudentForm(Model model, @RequestParam int id) {

model.addAttribute("child", service.getChildById(id));

return "admin/child";

}

@PostMapping("/admin/child")

public String processAddFavourite(Model model, @RequestParam long id, Principal principal) {

Child child = service.getChildById(id);

Favourite favourite = new Favourite(child.getId(), child.getFirstName(), child.getYearOfBirth(),

child.getGender(), child.getCode(), child.getHobby(), child.getPhotoName(), child.getIllness());

final MyUserDetails userDetails = userDetailsService.loadUserByUsername(principal.getName());

long user\_id = userDetails.getUser\_id();

User user = userService.getUserById(user\_id);

favouriteService.addFavourite(favourite, user\_id);

userService.saveUser(user);

model.addAttribute("favourites", user.getFavourites());

return "admin/favourites";

}

@GetMapping("/index")

public String reindex(Model model) {

model.addAttribute("children", service.getAllChildren());

return "index";

}

@GetMapping("/child")

public String processAddStudentForm(Model model, @RequestParam int id) {

model.addAttribute("child", service.getChildById(id));

return "child";

}

@GetMapping("/child\_list")

public String childList(Model model) {

model.addAttribute("children", service.getAllChildren());

return "child\_list";

}

@GetMapping("/details")

public String childDetails(Model model) {

model.addAttribute("children", service.getAllChildren());

return "details";

}

@GetMapping("/illness")

public String illness(Model model) {

List<Child> children = service.getAllChildren();

List<Child> illness = new ArrayList<>();

for (Child child : children)

{

if (child.getOriginalIllness() != null)

illness.add(child);

}

model.addAttribute("children", illness);

return "illness";

}

@GetMapping("/donation")

public String donation(Model model) {

model.addAttribute("company",companyService.getCompanyById(1));

return "donation";

}

@GetMapping("/donations\_list")

public String donationsList(Model model) {

model.addAttribute("donations", donationService.getAllDonations());

return "donations\_list";

}

@GetMapping("/about\_project")

public String aboutProject() {

return "about\_project";

}

@GetMapping("/contacts")

public String contacts(Model model) {

model.addAttribute("company",companyService.getCompanyById(1));

return "contacts";

}

@GetMapping("/admin/add\_child")

public String addChild(Model model) {

model.addAttribute("child", new Child());

return "admin/add\_child";

}

@PostMapping("/admin/add\_child")

public String processAddStudentForm(@Valid Child child, BindingResult bindingResult, @RequestParam("file") MultipartFile file) {

if (bindingResult.hasErrors()) {

return "admin/add\_child";

}

// normalize the file path

String fileName = StringUtils.cleanPath(file.getOriginalFilename());

// save the file on the local file system

try {

Path path = Paths.get("/Users/disvik/Documents/ONPU/2course2sem/oop/course\_work/src/main/resources/static/images/" + fileName);

Files.copy(file.getInputStream(), path, StandardCopyOption.REPLACE\_EXISTING);

child.setPhotoName("../images/" + fileName);

} catch (IOException e) {

e.printStackTrace();

}

if(child.getIllness().length()<5)

child.setIllness(null);

service.saveChild(child);

return "redirect:/admin/index";

}

@GetMapping("/admin/edit\_child")

public String editChild(Model model, @RequestParam int id) {

model.addAttribute("child", service.getChildById(id));

service.deleteChild(id);

return "admin/edit\_child";

}

@PostMapping("/admin/edit\_child")

public String editPostChild(@Valid Child child, BindingResult bindingResult, @RequestParam("file") MultipartFile file) {

if (bindingResult.hasErrors()) {

return "admin/edit\_child";

}

// normalize the file path

String fileName = StringUtils.cleanPath(file.getOriginalFilename());

// save the file on the local file system

try {

Path path = Paths.get("/Users/disvik/Documents/ONPU/2course2sem/oop/course\_work/src/main/resources/static/images/" + fileName);

Files.copy(file.getInputStream(), path, StandardCopyOption.REPLACE\_EXISTING);

child.setPhotoName("../images/" + fileName);

} catch (IOException e) {

e.printStackTrace();

}

if(child.getIllness().length()<5)

child.setIllness(null);

service.saveChild(child);

return "redirect:/admin/animal\_list";

}

4.2.5 Особливості використання спеціалізованих бібліотек та АРІ

Для реалізації пожертвування треба застосувати стороннє платіжне АРІ. Буде використано АРІ FONDY 1.0.

Взаємодія з АРІ відбувається наступним шляхом:

1. Клієнт на сайті вводить суму та інші ознаки пожертвування.
2. ПП на сайті відображає кнопку оплати, при натисканні на яку буде сформована HTML форма для відправки на платіжний шлюз FONDY
3. На сайті користувач натискає кнопку оплати, і торговець перенаправляє покупця в браузері методом редиректу на url https://api.fondy.eu/api/checkout/redirect/, передаючи параметри методом HTTPS POST
4. Користувач вводить платіжні реквізити на сайті платіжного шлюзу FONDY, платіжний шлюз здійснює списання коштів через зовнішню платіжну систему (банк-еквайєр)
5. Платіжний шлюз FONDY, отримавши від зовнішньої платіжної системи відповідь з результатом платежу, перенаправляє покупця в браузері на сайт торговця методом редиректу, передаючи параметри з результатом оплати методом HTTPS POST. ПП на сайті відображає сторінку з результатом оплати.
6. Для більшої надійності доставки відповіді про результат платежу на бік торговця, платіжний шлюз FONDY також дублює відповідь з результатами оплати на сервер сайту торговця методом host-to-host HTTPS POST.

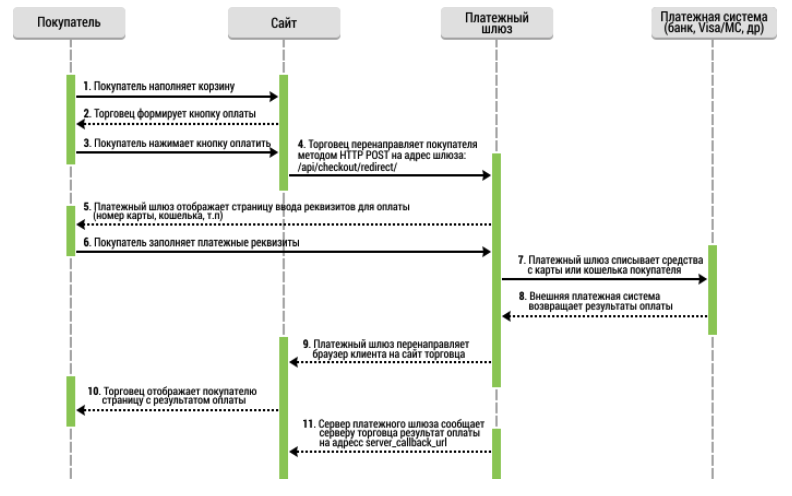


Рис. 4.2.5 – Схема взаємодії застосунку з зовнішнім АРІ

4.2.6 Висновки до підрозділу

Реалізація інформаційної системи для допомоги дитячим будинкам виконана на мові об'єктно-орієнтованого програмування Java із використанням фреймворку Spring MVC. Спираючись на шаблон модель-уявлення-контролер, Spring MVC допомагає будувати веб-додатки, так само гнучкі і слабо пов'язані, як сам фреймворк Spring. Також використано стороннє платіжне АРІ FONDY.