Web Application Programming Project Report

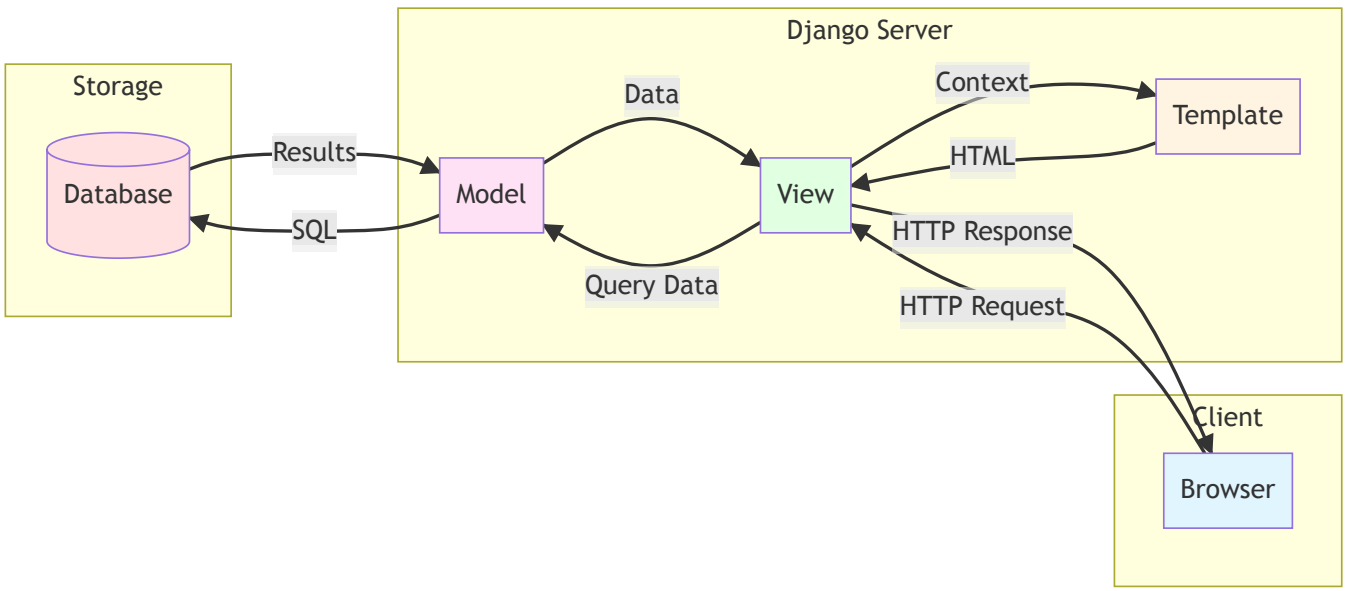
Katarzyna Przech 333114

Idea

This website is designed for a local guinea pig store, allowing them to showcase their guinea pigs and share articles through a blog. Customers can browse the available guinea pigs and read blog posts. The store's staff can log in to manage the listings and blog content, adding new items or updating existing ones. The app is built to look good and be easy to use on various devices, like computers or phones.

Dataflow

This app uses Django's MVT (Model-View-Template) pattern. The browser sends requests to Views, which query Models to fetch data from the Database, then pass that data to Templates for rendering HTML responses back to the browser.



Main view screenshots

Computer

SqueekyPiggy Home Guinea Pigs Blog Welcome, admin Logout

Welcome to the website of the SqueekPig store!

Here you can find the entire offer of the cutest guinea pigs from our stationary store and read our blogs with information needed for every guinea pig lover!

Visit us at: ul. Koszykowa 75, 00-662 Warszawa

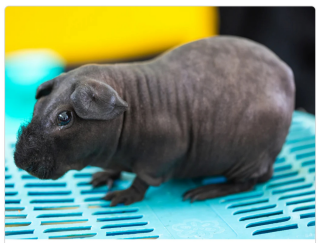
SqueekyPiggy Home Guinea Pigs Blog Welcome, admin Logout

Our Guinea Pigs

Add New Guinea Pig

Min Price Max Price Filter

Dutch Guinea Pig
199.99zł
Edit Delete



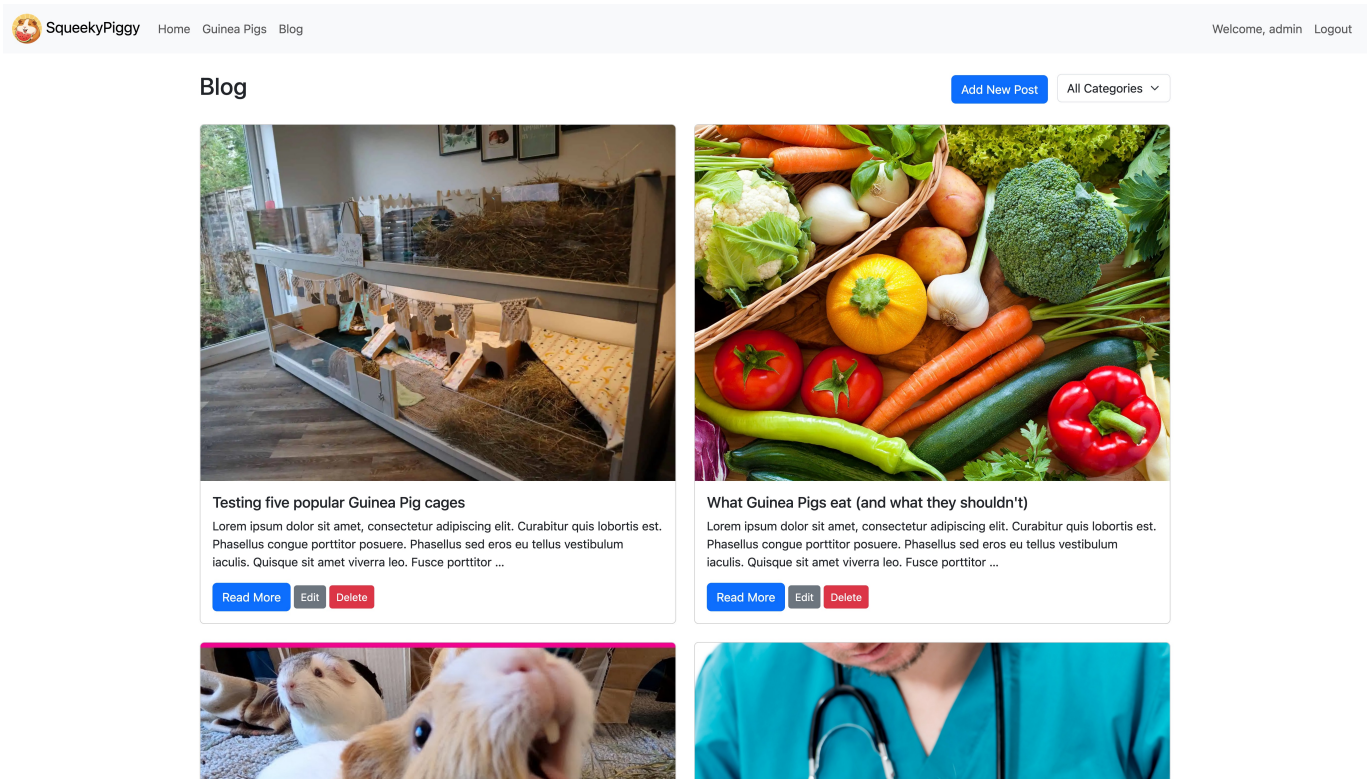
Skinny Pig
150.00zł
Edit Delete

Abyssinian Guinea Pig
250.00zł
Edit Delete

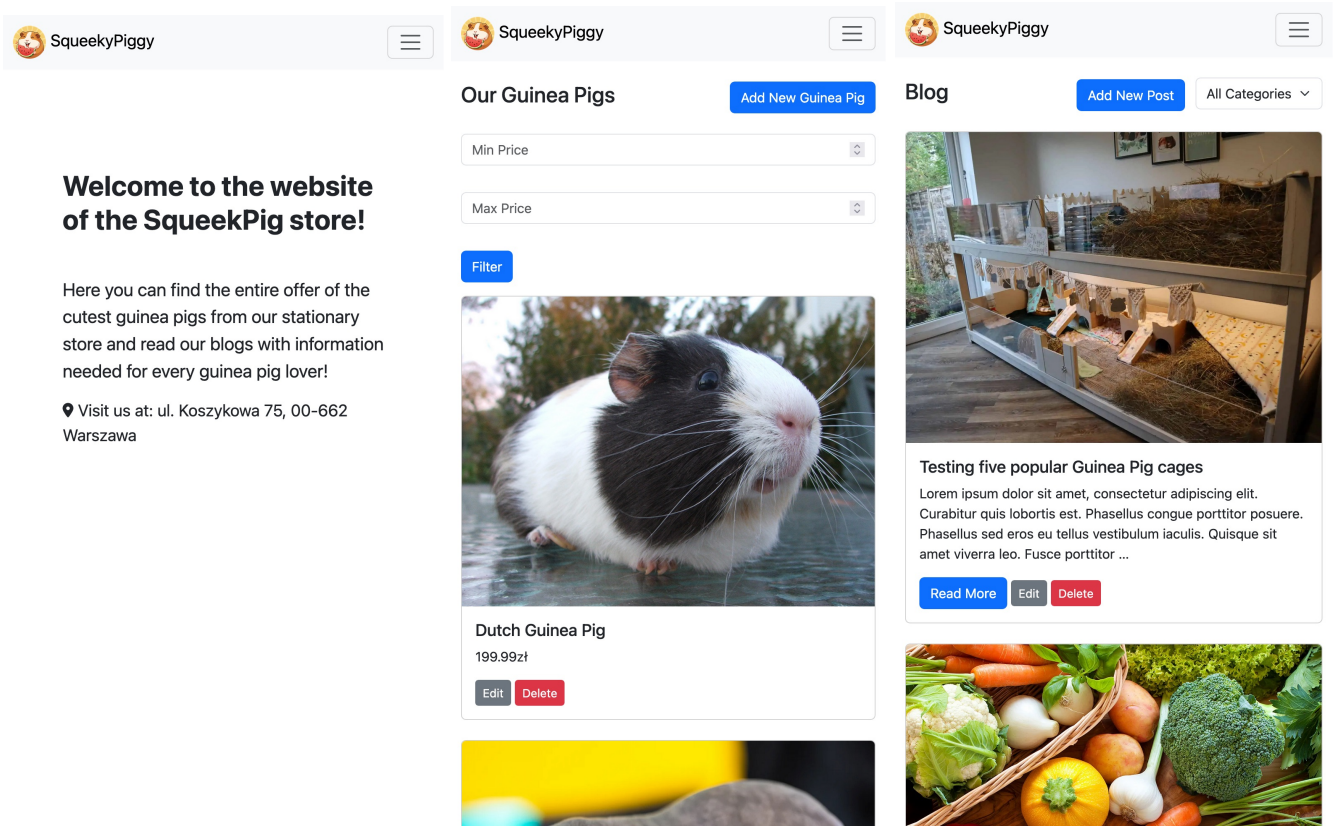
Peruvian Guinea Pig
300.00zł
Edit Delete

Himalayan Guinea Pig
279.99zł
Edit Delete

Crested Guinea Pig
239.99zł
Edit Delete



Smartphone



Requirements

1. Allow users login and password reset

The app provides functionality to login, registration and password reset. It mostly utilizes `django.contrib.auth` package, which provids the patterns for register, login, logout, password change, and password reset functionalities. URLs of auth functionality can be found in `store/urls.py`, which includes both `django.contrib.auth.urls` (django built-in authentication system) and `users.urls` (extended to the django built-in ones) under the `accounts/` path.

Password reset functionality is by default configured to send emails to the console as to not require a SMTP server setup. However, it can be configured by setting the following enviroment variables:

```
EMAIL_BACKEND=django.core.mail.backends.smtp.EmailBackend
EMAIL_HOST=[smtp-host]
EMAIL_PORT=[smtp-port]
EMAIL_USE_TLS=True
EMAIL_HOST_USER=[email-address]
EMAIL_HOST_PASSWORD=[email-password]
```

2. There are at least 3 roles (eg, a normal user, manager and backend admin roles)

There are 3 roles in the app:

- **Normal user:** A standard Django user without any specific group assignments and read-only permissions.
- **Manager:** A user assigned to a custom 'Manager' group. This role can add and modify listings and blog posts.
- **Admin:** A role with the Django admin interface.

The application utilizes Django's built-in `User` and `Group` models to manage roles. `has_group` template filter has been added in `users/templatetags/auth_extras.py` indicates that custom Manager group is used to define specific roles beyond the default Django user types.

3. The application allows only the logged-in user to access restricted resources

Restricted resources, specifically the views for creating, updating, and deleting Guinea Pig listings and Blog Posts, are protected using Django's `UserPassesTestMixin`. This mixin,

combined with a function that checks for membership in the 'Manager' group, ensures that only logged-in users with the 'Manager' role can access these functionalities.

- In `products/views.py`, `GuineaPigCreateView`, `GuineaPigUpdateView`, and `GuineaPigDeleteView` all inherit from `UserPassesTestMixin` and implement `test_func` to check `self.request.user.groups.filter(name='Manager').exists()`.
- Similarly, in `blog/views.py`, `BlogPostCreateView`, `BlogPostUpdateView`, and `BlogPostDeleteView` use `UserPassesTestMixin` with the same `test_func` logic.

4. At least 2 tables with items browsable by the regular user, which can be filtered

The application provides two main browsable tables: Guinea Pig listings and Blog Posts. Both are accessible to regular users and include filtering capabilities. For Guinea Pigs, filtering is based on min_price and max_price. Blog Posts can be filtered by category.

5. All forms data is validated

Form data validation is implemented across the application using Django's `ModelForm`s and custom validation logic within `forms.py` files.

- `products/forms.py`:
  - `GuineaPigForm` is a `ModelForm` for the `GuineaPig` model, inheriting its validation. It also includes a `forms.NumberInput` widget for the `price` field with `min='0'` for client-side validation.
  - `GuineaPigFilterForm` includes `DecimalField`s with `min_value=0` and a custom `clean` method to ensure `max_price` is not less than `min_price`.
- `blog/forms.py`:
  - `BlogPostForm` is a `ModelForm` for the `BlogPost` model, inheriting its validation.

User creation / login forms extend `django.contrib.auth` ones, which handles validation.

6. The application allows you to save or modify records

The application utilizes Django's class-based `CreateView` and `UpdateView` to provide functionality for saving new records and modifying existing ones for both Guinea Pig listings and Blog Posts.

- In `products/views.py`:
  - `GuineaPigCreateView` is responsible for creating new `GuineaPig` instances.
  - `GuineaPigUpdateView` is responsible for modifying existing `GuineaPig` instances.

- In `blog/views.py`:
  - `BlogPostCreateView` is responsible for creating new `BlogPost` instances.
  - `BlogPostUpdateView` is responsible for modifying existing `BlogPost` instances.

7. Use of AJAX technology or similar technique

The application incorporates AJAX technology for filtering blog posts. In `templates/blog/blopost_list.html` in `<script>` tag JavaScript and `fetch()` is used to fetch the data from the `blog/api/posts/` endpoint asynchronously, and when a response is received, it updates the HTML on the website without needing a full page refresh.

8. The application follow good programming practices

The application demonstrates good programming practices through its modular design, and adherens to Django conventions.

- **Modular organization:** The project is logically divided into distinct Django applications (`products` , `blog` , `users`), each encapsulating related functionalities (models, views, forms, URLs).
- **Django best practices**: Code effectively utilizes Django's built-in features and patterns, such as `ModelForm` s for form handling, class-based views (`ListView` , `CreateView` , `UpdateView` , `DeleteView`) for CRUD operations, `UserPassesTestMixin` for access control, and `django.contrib.auth` for authentication.

9. Application easily deployed on other computers (Docker prefferd) or web hosted

The application includes a `Dockerfile` that enables easy containerization and deployment using Docker. It can be built and ran with:

```
docker build -t squeeky-piggy .
docker run -p 8000:8000 squeeky-piggy
```

And then, the app can be accessed at http://localhost:8000

10. Unit test, integration test, Automatic tests (at least 70% cover)

The application includes unit and integration tests, executed using `pytest` , and achieves a 96% code coverage percentage. The output from the test run confirms that all 18 tests passed successfully.