Good Bite Report

Delivery 2

Security Applications

MCYBERS

Student: Anna Melkumyan Canosa
Student: Cecília Maria Rodrigues Correia
Student: Irene Cerván Barriga
Student: Johanna Nuñez Diaz
Student: Oriol Ramos Puig

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Facultat d'Informàtica de Barcelona

Contents

Introduction

Our team is developing a secure web application inspired by Too Good To Go, where vendors can register surplus food packages for sale, and customers can view and eventually purchase them (currently only with cash, as card payments are not implemented yet).
Vendors manage and publish the products they want to sell, while regular users access the platform to view the available products.
The project emphasizes security, containerization, and role-based access control, so that both user data and product information are safely managed within isolated environments.

Framework and technologies used

To handle the business logic in a secure way:

Backend

The backend is implemented using Django, a Python web framework that provides:

- Management of the business logic and web structure (models, views, and templates).
- A built-in authentication system, handling login, registration, and password validation.
- Mapping of Python classes to database tables.
- Separation between logic, database operations, and presentation layers.

The Django project is organized as follows:

- **core/:** project-level configuration (settings.py, urls.py, etc.).
- **GoodBite/**: main application containing all the web features (models, forms, views, templates).

Database

The system uses PostgreSQL as its relational database due to its reliability and compatibility with Django.

The database currently includes tables (models) for:

- **Users**: extended to include roles (customer, seller, admin) and additional profile data.
- **Products**: containing fields such as name, description, image, price, stock, and seller ownership.

Frontend

The frontend uses Django Templates and standard HTML, following the MVT architecture.
Each page dynamically renders data from the backend (products, user profile, statistics, etc.), ensuring synchronization between the interface and the database.

Templates are organized as:

- **templates/main/** → main pages such as home, product list, confirm delete, etc.
- **templates/registration/** → login and register views.
- **templates/widgets/** → reusable UI components.

The design follows a clean, functional style inspired by Too Good To Go, using a green color palette and minimalistic layout.

Additionally, the templatetags/ folder includes a custom filter (mul) used to multiply two values directly within templates.

Security

The project follows OWASP ASVS 4.0.3 as the main security framework, ensuring:
- Strong authentication practices
- Safe data handling and storage
- Proper access control
- General adherence to industry-standard security guidelines

Also, passwords are hashed before being stored so an attacker wouldn't be able to reconstruct the passwords in case of entering the BD and file uploads are restricted to JPG/PNG, and file size is validated, because files bigger than 5Mb are not accepted.

Docker

The entire system runs inside Docker containers.
Docker ensures isolated and reproducible environments, protects configuration data, simplifies dependency management, and helps keep the deployment secure.

Basic Functionalities Implemented

User registration and Login

- Implemented using a customized creation form (username, first name, last name, email, password).
- Passwords are encrypted and securely stored in PostgreSQL.
- Email format and password complexity are validated
- Successful registration logs the user in automatically.
- Validation errors are displayed through Django's message system.
- Newly registered users default to the Customer role.
- Users who want to become Sellers must send an email to seller@goodbite.com. Then, an admin will review the requests and update their role.

Product management

All the products can be seen by all the users but every role has its own features.
Products appear dynamically on the main page, using template rendering.

Sellers can:
- Create new products by entering a name, description, price, stock and an image (which has to fulfill previously explained requirements).
- Edit the information or delete only of their own products.

Customers can:
- View the list of all the products available.
- Purchase products.

Also, there is a search bar which allows them to search a product by name, description or seller username, first name, or last name

Statistics management

Admins access a dedicated statistics section located in the main bair which shows:

- Total number of products
- Units sold
- Total revenue
- Total stock value
- Average product price
- Rankings (top sellers, best-selling products, highest stock, most expensive products)
- Low stock alerts (<20%)
- Out-of-stock items

This allows admins to maintain full operational control of the app.

Profile management

A new Profile section can be found in the main navigator bar once the user is logged in. This section allows users to:

- View personal information (name, surname, email, joined date).
- Add phone number and birthday.
- Edit details through an editable form which has a Save and a Cancel buttons.
- Upload, update, or remove a profile image. Otherwise, a placeholder is shown.
- When trying to save the form, a validation is done. And for example if the age requirement is not fulfilled (minimum 16 years to use the app), it would appear in red informing the user about the error.
- Otherwise, if the validation is correctly done, a success green message appears

Admin management

Admins can:

- Change user roles.
- Deactivate accounts in case of suspicious behaviour.
- Manage all users and products via the Django admin interface.
- View global system statistics

Security features completed so far

To this stage we have been working on:

- Password policies (length, special characters, etc.)
- Role-based access controls
- Password protection in the database
- Secure file uploads (only JPG and PNG files and a size not bigger than 5Mb)
- Initial OWASP ASVS 4.0.3 testing
- Automated vulnerability scanning using:
  - pip-audit
  - bandit (static security analysis)

From both OWASP-based security assessments, no known vulnerabilities were identified in the project's dependencies.

However, the source code analysis did reveal some issues:

- The first scan reported one low-severity and one medium-severity finding. The low-severity issue has already been resolved.
- The medium-severity issue remained unresolved and reappeared in the second analysis, where it was the only vulnerability flagged.

Current status and next steps

Completed

- Docker environment setup (web + database containers).
- Core Django configuration.
- App structure (Good Bite/).
- User registration and login with validation and encrypted passwords.
- Role-based access (admin, seller, customer).
- Product CRUD (create, update, delete) for sellers.
- Profile section (editable personal data, upload profile image).
- Admin console functionalities (role assignment, account deactivation).
- Database seeding (seed.py) and clearing scripts (clear_data.py)
- Set up roles script (setup_roles.py).
- Product purchase workflow.
- Product filtering (by name, description, seller.).
- Admin statistics section
- Two OWASP ASVS security testing.

Pending

- Final round of OWASP ASVS security testing
- Creating and installing an HTTPS server certificate
- Solve the possible binding to all interfaces by hostname or IP filter
- MFA (although the logic is done, the design to align it with the rest of the app is missing)

Recommendations to perform

Recommendation to our team was to be careful with the libraries of django chosen to use: checked with the two audits done, where they indicated the libraries used don't have any vulnerabilities.
Another one was related to security features. The suggestion was to add one extra to our list. Proposals at that moment were to add MFA or content encryption, which at the end we decided to go with MFA and implement it.

Bibliography

Code repositories https://github.com/pypa/pip-audit

Our repository: https://github.com/sissamrcorreia/AS-project

List of reviewed webs

OWASP : https://atenea.upc.edu/pluginfile.php/6584636/mod_resource/content/6/OWASP%20Application%20Security%20Verification%20Standard%204.0.3-es.pdf

Django allauth: https://docs.allauth.org/en/latest/

Bandit: https://bandit.readthedocs.io/en/latest/

Pip-audit: https://pypi.org/project/pip-audit/,

Docker: https://www.youtube.com/watch?v=KiACzzCtz1s

Let’s encrypt: https://letsencrypt.org/