UNIVERSITY OF PIRAEUS – DEPARTMENT OF INFORMATICS
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Postgraduate Program "Informatics"
Πρόγραμμα Μεταπτυχιακών Σπουδών «Πληροφορική»

MSc Thesis
Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**"Give & Share - A Community Based Donation Web Application"** "Δίνω & Μοιράζομαι - Μια Διαδικτυακή Εφαρμογή Δωρεάς με Επίκεντρο την Κοινότητα"
Student's name-surname: Ονοματεπώνυμο Φοιτητή:	**Maria Bourzikou** Μαρία Μπουρζίκου
Father's name: Πατρώνυμο:	**Ilias** Ηλίας
Student's ID No: Αριθμός Μητρώου:	ΜΠΠΛ/2229
Supervisor: Επιβλέπων:	**Efthimios Alepis, Professor** Ευθύμιος Αλέπης, Καθηγητής

June 2026 / Ιούνιος 2026

3-Member Examination Committee
Τριμελής Εξεταστική Επιτροπή

Alepis Efthymios **Professor**	**Virvou Maria** **Professor**	**Sotiropoulos Dionisios** **Associate Professor**
Αλέπης Ευθύμιος Καθηγητής	Βίρβου Μαρία Καθηγήτρια	Σωτηρόπουλος Διονύσιος Αναπληρωτής Καθηγητής

Abstract

This thesis presents the design and implementation of Give & Share, a community-based donation web application developed to support the exchange of useful items between registered users. The system addresses a practical social problem: objects that remain in good condition may be discarded, while other people may need these same goods but lack a simple way to find them.

The platform enables a donor to publish an item with descriptive information and a category, while a receiver can browse available items, open their details, request an item and communicate with the donor. In this way, the application supports the complete donation workflow, from publication and discovery to communication, status management and user feedback.

Give & Share is implemented as a full-stack web application using Spring Boot for the backend and Thymeleaf/HTML pages for the frontend. Its functionality includes registration, authentication, password recovery, profile management, item donation, category browsing, donor-receiver messaging, request tracking, ratings, community stories and an administration dashboard for system supervision.

The resulting application combines a social purpose with a structured software solution. By encouraging reuse, facilitating communication and supporting administrative control, it provides a practical platform for community cooperation while demonstrating core principles of web application development and organization.

Περίληψη

Η παρούσα μεταπτυχιακή διατριβή παρουσιάζει τον σχεδιασμό και την υλοποίηση του Give & Share, μιας διαδικτυακής εφαρμογής δωρεάς με επίκεντρο την κοινότητα, η οποία αναπτύχθηκε για την ανταλλαγή χρήσιμων αντικειμένων μεταξύ εγγεγραμμένων χρηστών. Το σύστημα αντιμετωπίζει ένα πρακτικό κοινωνικό πρόβλημα: αντικείμενα που παραμένουν σε καλή κατάσταση μπορεί να απορρίπτονται, ενώ άλλοι άνθρωποι μπορεί να τα χρειάζονται χωρίς να διαθέτουν έναν εύκολο τρόπο εντοπισμού τους.

Η πλατφόρμα επιτρέπει σε έναν δωρητή να δημοσιεύσει ένα αντικείμενο με περιγραφή και κατηγορία, ενώ ένας παραλήπτης μπορεί να περιηγηθεί στα διαθέσιμα αντικείμενα, να προβάλει τις λεπτομέρειές τους, να υποβάλει αίτημα και να επικοινωνήσει με τον δωρητή. Με τον τρόπο αυτό, η εφαρμογή υποστηρίζει ολόκληρη τη διαδικασία της δωρεάς, από τη δημοσίευση και την αναζήτηση έως την επικοινωνία, τη διαχείριση της διαθεσιμότητας και την αξιολόγηση των χρηστών.

Το Give & Share έχει υλοποιηθεί ως ολοκληρωμένη διαδικτυακή εφαρμογή με Spring Boot στο backend και σελίδες Thymeleaf/HTML στο frontend. Οι λειτουργίες του περιλαμβάνουν εγγραφή, αυθεντικοποίηση, ανάκτηση κωδικού πρόσβασης, διαχείριση προφίλ, δωρεά αντικειμένων, περιήγηση ανά κατηγορία, ανταλλαγή μηνυμάτων, παρακολούθηση αιτημάτων, αξιολογήσεις, ιστορίες κοινότητας και πίνακα διαχείρισης για την εποπτεία του συστήματος.

Η εφαρμογή συνδυάζει έναν κοινωνικό στόχο με μια οργανωμένη λύση λογισμικού. Ενισχύοντας την επαναχρησιμοποίηση, διευκολύνοντας την επικοινωνία και παρέχοντας διοικητικό έλεγχο, προσφέρει μια πρακτική πλατφόρμα συνεργασίας στην κοινότητα και αναδεικνύει βασικές αρχές ανάπτυξης και οργάνωσης διαδικτυακών εφαρμογών.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Professor Efthymios Alepis, for his guidance and thoughtful feedback throughout the preparation of this thesis. His support helped me organise my ideas, develop the Give & Share application with greater clarity, and complete this work with confidence.

I am also grateful to Professor Maria Virvou for the knowledge and insight I gained through her teaching during my postgraduate studies. Her contribution to my academic background supported my understanding of software development and the principles reflected in this project.

Finally, I would like to thank my family for their patience, encouragement, and unwavering support during the completion of my studies and this thesis. Their presence and confidence in me were invaluable.

Contents

Introduction

The present thesis concerns the design and implementation of Give & Share, a web application intended to support the donation and reuse of useful items within a community. The project originates from an everyday situation: objects that remain functional are often stored without purpose or discarded, while other people may be looking for those same goods. A digital system that connects these needs can make the exchange process easier, clearer, and more organised.

The central aim of the application is to provide a practical environment in which donation is treated as a complete interaction rather than as a simple item listing. Registered users can participate as donors, receivers, or both. A donor can publish an item with its description, category, and image, while a receiver can browse available donations, examine an item, submit a request, and communicate with the donor. In this manner, the platform supports an item from publication to a possible successful exchange.

The problem addressed by Give & Share has both social and technical dimensions. From a social perspective, the application encourages reuse and helps reduce unnecessary waste by making usable goods accessible to other community members. From a technical perspective, the project requires user accounts, donated objects, requests, conversations, ratings, and administrative supervision to be organised in a consistent information system.

The application has been designed around the main actions required in a donation platform. Users can register and log in, recover access through a password-reset workflow, edit profile information, donate items, browse categories, view item details, contact another user, monitor their personal items or requests, and submit ratings after an interaction. A community area allows users to share experiences, while the administrative dashboard enables oversight of users, items, and conversations. Together, these functions represent the full donation lifecycle.

In terms of implementation, the system is developed as a full-stack web application based on Spring Boot. Thymeleaf and HTML pages render the user interface, while controllers receive user actions and coordinate the appropriate responses. Data is represented through entities and accessed through repositories, allowing records to be retrieved or updated according to the current workflow. This structure connects the visible user experience with the backend logic required for authentication, persistence, communication, and administration.

Usability is an important concern of the project because the application is intended for ordinary users rather than only for technically experienced individuals. The interface therefore uses clear navigation choices, direct form actions, category-based browsing, and consistent visual elements across pages. Users should be able to recognise their current position, understand the next action, and receive appropriate feedback after submitting information or moving between donation and communication stages.

Trust and control are also necessary in a platform that facilitates interaction between community members. Conversations are linked to donated items, ratings allow feedback after an exchange, and the administrative area supports the supervision of system content. These mechanisms help maintain an organised environment without making the basic donation workflow difficult to use.

The remainder of this thesis describes the application from both a user-facing and a technical perspective. It presents the system purpose, user roles and principal workflows, followed by registration, authentication, item donation and browsing, messaging, community participation, administrative management and data organisation. Finally, it evaluates the implemented solution and identifies possible improvements. Overall, Give & Share demonstrates how a web application can respond to a real social need while providing a structured full-stack software project for an MSc thesis.

1. Overview

The Give & Share application is a web-based donation platform created to support the exchange of useful items between registered users. The central idea is simple: a person who owns an item that is no longer needed can publish it, while another person can browse the platform, request it, communicate with the donor, and complete the exchange. In this way the application combines a social objective with a technical implementation, because it does not only store information about items but also manages users, roles, messages, requests, ratings, and administrative supervision.

Key points:

- The platform connects donors, receivers, messages, requests, ratings, and administration in one workflow.
- The technical implementation uses a full-stack web structure instead of isolated static pages.
- The thesis focuses on both user experience and backend system organization.

The documented interface and code show that the application was developed as a full stack system using Spring Boot on the backend and Thymeleaf/HTML pages on the frontend. Each visible page is connected with a controller that receives user actions and returns the appropriate template. The interface follows a consistent design with blue and orange gradients, centered forms, rounded cards, and clear buttons. This gives the system a recognizable identity and makes the user flow easier to understand.

The purpose of the documentation is to explain the application not only as a sequence of pages, but as a complete software system. Therefore, each feature is described from both sides: what the user sees in the browser and what happens in the backend. This is important for an Informatics thesis, because it demonstrates the connection between usability, data modeling, controller logic, persistence, and security.

The system includes registration, login, password recovery, user profile editing, item donation, category browsing, item details, donor-receiver communication, request tracking, ratings, community stories, and an admin dashboard. These features form a complete donation lifecycle.

At overview level, the combined feature set is significant because item donation is supported by accounts, messaging, status changes, ratings, and administration. These linked functions establish the application as one coherent system.

The interface is designed to keep each task direct and understandable. Users are normally shown only the information and input fields that are relevant to the action they are performing. Clear button labels and return options support orientation and help reduce common mistakes, especially for users who are not technically experienced.

The frontend implementation relies on Thymeleaf expressions to insert dynamic values such as username, item fields, messages, ratings, and page information. This means that the same template can produce different output depending on the current user and the current database state. The interface is therefore not static; it is connected directly to backend data.

The page also contributes to the general goal of the application, which is to make donation and receiving actions easy to complete. Each visible element has a practical role: identifying the user, showing available data, guiding the next action, or confirming the result of a previous action. This makes the interface functional rather than decorative only.

1.1. Purpose, Motivation and Problem Definition

The motivation behind Give & Share is based on a common real-world problem: many items remain usable but are discarded because the owner does not have an easy way to find someone who needs them. At the same time, other people may need basic goods such as food, clothing, books, furniture, or electronics but may not know where to request them. The platform attempts to reduce this gap by creating a digital meeting point between donors and receivers.

Key points:

- Reduce waste by helping useful items reach people who need them.
- Support community cooperation through a simple digital exchange process.
- Provide a practical software solution for a real social problem.

From a social perspective, the application supports sustainability and community cooperation. Instead of treating unused objects as waste, the system encourages reuse. This is visible in the way the categories are organized: Food, Clothing, Books, Electronics, Furniture, and Other. These categories represent practical goods that users may exchange in everyday life. The application therefore has a clear social purpose and not only a technical one.

From a software engineering perspective, the goal is to implement a structured web application with authentication, database storage, file upload, page rendering, messaging, and role-based access. The user does not interact directly with the database; instead, all actions pass through controllers and repositories. This separation improves maintainability and makes the application easier to extend.

The main objective is to provide a simple but functional donation workflow. A donor can publish an item with a photo and category. A receiver can discover it and contact the donor. The donor can answer, mark the item unavailable, and both users can rate each other. This creates a complete interaction rather than a static catalogue.

The straightforward interaction model is relevant to the social aim: people should be able to offer or request useful objects without unnecessary procedural complexity. Clear stages make participation more accessible for community members.

In terms of usability, the page keeps the interaction focused. The user is not required to process unnecessary options, and the available controls guide the next step clearly. This makes navigation easier and supports a smoother experience for non-technical users.

The platform translates this motivation into usable functionality by presenting donated items according to stored categories and availability. As new donations are added, receivers are able to view relevant objects and begin an exchange through the application rather than relying on informal or disconnected communication.

In this way, the identified problem is addressed through an organised digital process. Usable goods can move from people who no longer need them to receivers who are actively looking for them, while the platform records the steps required to support a practical and responsible donation exchange.

1.2. Overall System Architecture

The structure of the application follows the common Model-View-Controller pattern. The model layer contains entities such as User, Item, DonorMessage, ChatMessage, Rating, and CommunityPost. These classes represent the data that is stored in database tables. They include fields, identifiers, relationships through usernames or ids, and timestamps that help the system track when records were created.

Key points:

- Model classes represent stored data such as users, items, messages, ratings, and community posts.
- Controllers receive user actions and decide which page or operation should happen next.
- Repositories provide structured access to the database through Spring Data JPA.

The controller layer is responsible for receiving HTTP requests. For example, AuthController handles login and registration pages, DonateController handles item donations, ContactDonorController and ContactReceiverController handle the conversation workflow, CommunityController manages community posts, and AdminController manages the administrative dashboard. Each controller has a clear responsibility, which makes the code easier to read and test.

The repository layer connects the application with the database. Repositories such as UserRepository, ItemRepository, DonorMessageRepository, ChatMessageRepository, RatingRepository, and CommunityPostRepository provide access to stored data. Spring Data JPA reduces the need for manual SQL by generating queries from method names, for example finding items by category or finding messages by conversation id.

The view layer is implemented with Thymeleaf templates. These templates combine static HTML and dynamic expressions. Values such as username, item name, status, average rating, and message lists are inserted into the page by the backend model. This architecture allows the same page to show different content depending on the logged-in user and the data stored in the database.

The visual structure is repeated in a controlled way throughout the application. This gives users a stable sense of place while they move between registration, donation, browsing, messaging, and administration pages.

The page follows a simple interaction model. Each form or action area includes only the details needed for that part of the workflow, while buttons and back-navigation links make the available choices easy to understand. This reduces confusion and supports practical use of the platform.

Within this architecture, Thymeleaf operates specifically as the connection between controller data and the rendered view. A controller prepares model values, such as item status or a message list, and the selected template presents those values to the logged-in user. This clarifies the responsibility of the view layer without duplicating the role of persistence.

Consequently, the architecture maintains a clear separation of concerns. Repositories manage data access, controllers coordinate requests and decisions, and templates organise the visible presentation. This separation makes the system easier to understand, maintain, and extend as additional functionality is introduced.

1.3. User Roles and Main Functional Areas

The application distinguishes between different user roles at the functional level. During registration, a user can select Donor, Receiver, or Both. This choice describes how the user intends to use the platform. In addition, the system stores a security role such as ROLE_USER or ROLE_ADMIN. This creates a distinction between application role and security authority.

A donor is mainly responsible for adding items to the platform. The donor uses the Donate Item page, fills in the item information, uploads a picture, and then monitors interest through My Items. The donor can also view receiver messages, respond to requests, change the status of an item, and rate a receiver after an interaction.

A receiver is mainly responsible for finding and requesting items. The receiver uses the Find What You Need icon, browses categories, opens item details, and contacts the donor. The receiver later checks My Requests to see responses and continue the conversation. After the interaction, the receiver can rate the donor.

A user with the Both role can participate in both flows. This makes the platform flexible because the same person can donate one item and request another. Finally, the administrator is a special account that can access the /admin route and manage users, items, and conversations. The admin area is protected by role checking so normal users cannot access it.

This separation of functions keeps responsibility understandable: donors manage offered items, receivers manage their requests, and administrators supervise records. The role model therefore supports both practical navigation and controlled access.

Usability is supported through a clean structure and clear navigation. Instead of presenting many unrelated choices, the page guides the user toward the next useful action. This helps users complete tasks with fewer errors and without needing technical knowledge.

The distinction between roles also influences the information and controls that become relevant during use. A donor needs item-management and receiver-contact options, while a receiver needs browsing and request-follow-up features. An administrator requires oversight functions rather than the ordinary actions used to complete an exchange.

In this way, the role model improves both clarity and control across the platform. The same application can support different participation needs while maintaining understandable navigation, guiding each user towards the relevant functions, and reserving administrative actions for accounts with the appropriate authority.

2. User Registration Functionality

Registration is the first controlled entry point into the Give & Share system. The registration page is presented as a centered white form container on a gradient background. This design matches the visual identity of the rest of the application and makes the form stand out clearly. The required fields are username, email, password, and user role. The role selection gives the platform information about whether the user wants to donate, receive, or use both types of functionality.

Key points:

- Registration creates the user account and stores the selected functional role.
- Validation protects the system from incomplete or duplicated account data.
- The stored user data supports later profile editing and personalized workflows.

The registration form uses HTML required fields so the browser can prevent empty submissions before the request reaches the server. The username field identifies the account, the email field is used for identity and possible password recovery, and the password field protects access. The userRole select menu contains Donor, Receiver, and Both. This field is stored separately from the security role, which is important because a user can have ROLE_USER while still being categorized functionally as Donor or Receiver.

From the backend side, AuthController contains the GET mapping that displays the registration form and the POST mapping that saves the new user. When the form is submitted, Spring maps the input values to a User object. The controller then sets the security role to ROLE_USER and saves the selected userRole. Finally, the UserRepository stores the user record in the database and the user is redirected to login.

This process shows a basic but functional account creation workflow. It connects a simple user interface with entity binding, role initialization, and persistence.

The described failed registration attempts are especially useful for evaluation because they show that the interface is not only designed for the ideal case. The system responds when the username already exists, when the email already exists, or when the password is missing or too short. These cases show that the application protects the quality of stored account data.

The role field in registration has functional meaning. A user who selects Donor is expected to donate items, a Receiver is expected to request items, and Both can participate in both sides. Even though the current navigation still gives access to the major pages, the stored role can support future improvements such as showing role-specific menus or statistics.

The registration process also prepares later personalization. Fields such as full name, phone, address, city, country, postal code, and bio are not all collected at registration time, but they exist in the User model and can be edited later in the profile page. This keeps registration short while still allowing richer profile data after the account is created.

A clearly structured registration form is significant because it introduces users to the system while collecting only the data needed to create a valid account. Validation feedback helps users correct errors before entry.

2.1. Registration Validation and Error Examples

The registration examples describe both successful and unsuccessful attempts. This is important because a registration system must not only save correct data, but also reject incomplete or invalid input. The User entity includes validation annotations such as NotBlank, Email, and Size. These annotations express the rules for acceptable values directly inside the model class.

The username is marked as required and must have a length between 3 and 20 characters. This prevents extremely short or empty usernames. The email is also required and must follow a valid email format. The password is required and must have at least 6 characters. These validation rules are simple, but they already improve the reliability of the user database.

The model also uses unique constraints for username and email. This prevents two accounts from sharing the same identifier. In a real system this is essential because login is based on username, and password recovery is related to email. If duplicate values were allowed, authentication and recovery could become ambiguous.

The unsuccessful attempts described in the documentation are useful for evaluation because they demonstrate how the interface reacts when the user submits invalid data. For example, if a username already exists or if the password is too short, the user receives a message explaining the problem. These messages reduce confusion and guide the user toward a successful registration. The successful attempt then redirects the user to the login page, completing the account creation process.

The validation examples also show the practical role of model constraints in the interface. When a value violates a rule, the submitted account is not stored and the user is directed to correct the relevant field. This behavior links the annotations of the User entity with visible feedback during registration.

Validation is also applied to the role-selection step because a new account must be linked with a supported functional role. Recording Donor, Receiver, or Both as a defined value avoids incomplete account records and ensures that later screens can rely on consistent role information.

Although the registration form intentionally gathers only essential values, the checks at this stage create a reliable starting record. Additional personal information can be completed later, after the account has been successfully validated and saved. This division keeps first access simple without weakening the integrity of the stored user data.

At this stage, usable feedback matters because invalid submissions must be corrected without confusing the applicant. Specific validation messages connect database reliability with a manageable first experience of the system.

2.2. User Entity and Stored Profile Data

The User entity is more than a minimal login table. Besides id, username, email, password, role, and userRole, it also contains additional profile fields such as fullName, phone, address, city, country, postalCode, and bio. These fields support the profile page and allow the user to personalize the account after registration.

The entity also includes resetToken, which is used by the password recovery mechanism. This field allows the system to generate a temporary token when a user requests a password reset. The token can then be

used to validate the reset link and connect it with the correct account. After the password is changed, the token should be cleared so that it cannot be reused.

The use of annotations in the User entity makes the data model self-descriptive. For example, Column(unique = true, nullable = false) on username and email shows that these fields are required and must be unique in the database. This is a stronger guarantee than frontend validation alone, because it is enforced at persistence level.

The existence of both role and userRole is an important design choice. The role field is connected to authorization, for example ROLE_USER or ROLE_ADMIN. The userRole field describes the user's practical behavior in the application, such as Donor, Receiver, Both, or ADMIN. This separation helps the system manage security and user experience independently.

Within the User entity, validation is represented as stored-data rules rather than only form behavior. Required and unique fields protect the account record itself, while format and length constraints help ensure that username, email, and password values remain usable by later application processes.

The two role-related fields have separate purposes inside the entity. The security value determines authorisation, while userRole records whether the user participates as a Donor, Receiver, Both, or ADMIN. Storing both values allows access control and platform participation to be managed independently.

The additional profile fields extend the account without making the first form too long. Because these attributes belong to the User entity, later profile updates can be saved consistently and connected to the same account used for donations, messages, and ratings, while registration remains focused on the information required for initial access.

The profile page uses disabled fields by default, which is a good interface decision. Users can read their information without accidentally changing it. The Edit button then switches the form into editing mode. This two-step design is especially useful for profile data, where accidental modification could create confusion later.

My Items and My Requests are role-related pages. My Items supports donors by showing what they uploaded and whether receivers contacted them. My Requests supports receivers by showing the conversations they started. These two pages mirror each other and make the donor-receiver model visible in the interface.

3. Login and Authentication Process

The login process controls access to the internal pages of the application. Before login, the public screen contains the main icons, but when an unauthenticated user attempts to use protected features, the system redirects them to the login page. This behavior ensures that actions such as donating items, requesting items, editing a profile, or sending messages cannot be performed anonymously.

Key points:

- Authentication prevents anonymous users from performing protected actions.
- Session identity allows controllers to connect actions with the correct account.
- Password recovery extends the login system with a reset-token workflow.

The login page uses the same visual style as the registration page: a central white card on a blue-orange gradient background. It contains a username field, a password field, a login button, a forgot-password link, and a link to the registration page. This makes the authentication process clear and familiar for users who have already visited the registration form.

A useful frontend detail is the password visibility toggle. It is implemented with JavaScript and changes the password input type between password and text. This feature helps users avoid typing mistakes while still keeping the field hidden by default. The interface also displays error messages when login fails, such as invalid username or password.

After successful authentication, the user is redirected to the initial screen of the application. The username appears in the top-right menu, confirming that the session belongs to the logged-in user. This small visual indicator is important because the application contains user-specific pages such as My Items and My Requests.

The login workflow is connected to route protection. When a visitor who is not authenticated tries to use protected functionality, the system sends the user to the login page. This is visible in the behavior of the app before login, where clicking internal actions leads to authentication instead of unrestricted access.

The password recovery pages extend the authentication system beyond a simple username-password form. The forgot-password page collects an email, the backend creates a reset token, and the reset-password page allows the user to create a new password. The console reset link shown in development proves that the token workflow is functioning.

The hidden CSRF token in the login form is an important technical detail because it shows that the system is aware of form security. Together with session-based Principal access, it provides a basic but meaningful security structure for the application.

The authentication screen is intentionally familiar because users may visit it repeatedly. Keeping login controls clear, providing recovery access, and confirming session state helps users enter protected workflows with fewer mistakes.

3.1. Failed Login and User Feedback

The unsuccessful login example shows how the system responds when the entered credentials do not match a registered account. Instead of allowing access or failing silently, the page displays a clear error message. This message informs the user that the username or password is invalid, which is important for usability.

In the login template, the error message is shown when the URL contains the error parameter. The corresponding block uses Thymeleaf condition th:if="$□param.error□". This means the backend or security configuration redirects the user back to the login page with a parameter indicating failure. The page then renders the message inside a styled red alert area.

The application also supports logout feedback. If the user logs out successfully, the login page can display a green message confirming that the session has ended. This is helpful because users should know whether they are still authenticated or not. It also gives a clean ending to the session lifecycle.

The error handling in this part is simple, but it follows a good principle: every important user action should produce visible feedback. When login fails, the user sees what went wrong. When logout succeeds, the user sees confirmation. When login succeeds, the user is taken to the home page and sees their username. Together, these states make the authentication behavior understandable.

The failed-login example also shows that protected navigation is interrupted when authentication does not succeed. Instead of continuing to an internal page, the user remains in the login flow and receives feedback, which makes the result of the attempted action clear. This reinforces the purpose of the error message displayed on the login page.

Recovery is relevant here as a user-facing alternative after unsuccessful access. The forgot-password link prevents repeated guessing by giving a user who cannot remember a password a clear next step from the login screen and a controlled way to regain access. This option therefore supports usability without weakening the authentication process.

The login form combines clear feedback with safe submission behavior. Its feedback states and secure request handling support a controlled entry process without requiring the user to understand the underlying authentication configuration.

Visible feedback is central in this example: an unsuccessful attempt leaves the user informed, while successful authentication or logout produces an understandable next state.

3.2. Forgot Password and Reset Token Workflow

The password recovery feature supports users who cannot access their account because they forgot their password. From the login page, the user clicks the Forgot your password link. The next page asks for the registered email address. If the email belongs to an account, the system generates a reset token and associates it with that user.

The User entity contains a resetToken field specifically for this purpose. In development mode, the reset link can be printed in the console, as described in the documented workflow. In a production environment, the same link would normally be sent by email. The logic is still useful because it demonstrates the basic structure of secure account recovery.

When the user opens the reset link, the application displays a form for entering a new password. The token connects the reset request with the correct account. After the password is updated, the reset token should

no longer be valid. This prevents a previously used link from being reused by another person.

This part of the system adds reliability to the authentication flow. A user is not permanently locked out after forgetting a password. At the same time, the token mechanism ensures that password changes are not performed without an identifier. The recovery process therefore balances usability and security.

The recovery workflow is linked to account protection because a reset operation must be associated with an existing user and a temporary token. This ensures that the new password is set through the recovery process rather than through ordinary unauthenticated navigation. The feature therefore preserves control over changes to stored credentials.

The console link used during development allows the reset sequence to be verified without altering its functional structure. In deployment, the delivery method can change to email while the same token-based validation and password-update steps remain applicable to the user. This separates testing practice from the intended recovery design.

The reset process guides the user through controlled actions: requesting a link, opening the tokenised address, and submitting a replacement password. This structure keeps recovery understandable while retaining the required account checks.

For password recovery, a focused layout helps the user complete a sensitive operation safely. The page requests only the information required for requesting and completing a valid credential reset.

The layout is intentionally straightforward. Input fields are limited to the needs of the current task, and action buttons use clear wording. As a result, the page remains accessible and helps users move through the donation workflow with confidence.

3.3. Security Elements in Authentication

The login form contains a hidden CSRF field. This is visible in the code through the Thymeleaf expression that inserts the CSRF parameter name and token value. CSRF protection is important for forms that submit state-changing requests, because it helps prevent unauthorized form submissions from other sites.

The application also relies on the authenticated Principal object in several controllers. For example, DonateController uses Principal to set the donorUsername of a new item, and the messaging controllers use Principal to identify the current sender. This design prevents the user from manually submitting another username through the form and pretending to be someone else.

The system separates authenticated users from anonymous visitors. Protected operations are not available until login succeeds. This is visible in the described workflow, where clicking features as a public visitor redirects to the login page. This ensures that database-changing operations are associated with a real account.

The current implementation already demonstrates the essential security structure: login, logout, CSRF token, session identity, and role checking for admin. Possible improvements could include password hashing verification, custom access-denied pages, stronger password rules, and email- based reset delivery. However, the existing workflow is enough to show a complete authentication lifecycle in the context of this thesis project.

The login workflow is connected to route protection. When a visitor who is not authenticated tries to use protected functionality, the system sends the user to the login page. This is visible in the behavior of the

app before login, where clicking internal actions leads to authentication instead of unrestricted access.

From a security perspective, the reset-token feature complements authentication by limiting password changes to requests linked with a valid temporary identifier. It therefore provides a controlled recovery path when the user has lost access, without exposing account changes to ordinary anonymous navigation.

CSRF protection applies the same protective principle to submitted forms. A state-changing request must contain the expected token, helping prevent unauthorised external submissions during authenticated use of the application.

Security protections should operate without making normal use confusing. Token validation and protected submissions remain backend concerns, while users are guided through recognisable login and recovery steps.

4. Home Page and User Menu After Login

After successful login, the user reaches the initial authenticated screen of the application. This page acts as the central navigation point. The most visible elements are the three main icons in the middle of the page and the username menu in the top-right corner. The username confirms that the session is active and identifies the logged-in user.

The top-right menu contains personal actions. When the user opens it, the dropdown displays Profile, My Items, My Requests, and Logout. This design groups account-related actions in one place. It avoids cluttering the main dashboard while keeping important pages accessible from anywhere.

The menu is repeated across several pages, including Profile, My Items, My Requests, Community, Donate Item, and Item Details. This consistency is important because users can move through the application without losing access to their account options. The same menu structure also reinforces the identity of the application as a coherent system.

From a technical perspective, the menu uses Thymeleaf to display the username from the model. Controllers add the username attribute using the authenticated Principal. This connects the frontend view with the backend session. The dropdown itself is implemented with CSS hover behavior or JavaScript toggling depending on the page. The result is a simple but effective navigation element.

After authentication, the home page acts as the user's personal starting point.
Its available navigation options lead to authenticated functions, while the displayed username confirms the current session and gives the interface an immediate personal context.
This makes the transition from login to active platform use visible.

Links within the user menu make account-related actions reachable without returning to a separate authentication screen. Profile, item, request, and logout options are grouped together so navigation remains compact, consistent, and predictable on the home page.
The user can therefore continue with the required task from one recognised location.

This arrangement supports a clear transition from access control to ordinary use of the platform. Security remains active in the background, while the user mainly interacts with dashboard actions that are relevant after a successful login.

The Profile option in the menu gives direct access to stored personal information from the authenticated home page. Its placement makes account management discoverable without distracting from the three primary donation-related actions presented in the dashboard.
It therefore extends navigation while keeping the home screen organised.

The My Items and My Requests entries complement the main dashboard by providing personal follow-up areas. Rather than beginning a new workflow, users can return directly to previously donated items or initiated requests and continue their existing interactions.
These options connect the central menu with ongoing activity in the platform.

4.1. Profile Page and Editable User Data

The Profile page allows users to view and update personal information. The form includes full name, email, username, password placeholder, phone number, address, city, country, postal code, role, and bio. The design uses a centered white container similar to other forms, which keeps the interface clean and readable.

A key usability detail is that most fields are disabled at first. This prevents accidental editing. The user must click the Edit button to enable the inputs. The username remains read-only, which is reasonable because the username is a unique identity value used across the system. Changing it casually could affect item ownership, messages, and ratings.

After editing, the user clicks Save. The form is submitted to the backend using the /profile route, and the updated values are stored in the User entity. This shows model binding in practice: the form fields correspond to properties inside the user object.

The profile feature may seem secondary compared to donation or messaging, but it supports trust between users. Contact information such as city, phone, and bio can provide context when arranging an item exchange. The selected role also describes whether the user participates as donor, receiver, or both. Therefore, profile management contributes to personalization and practical interaction.

The initial disabled state shows that the page first presents saved data for review. Enabling edits deliberately separates viewing from modification, so the user can examine the account record before submitting updated personal information through the profile form. This preserves the intended focus of the page on controlled profile management.

Because profile information can support later contact and trust, this page connects personal data with activity elsewhere in the system. Updated details remain associated with the user who donates items or begins requests, while the form remains the central location for managing those stored account attributes.

The rating box on these pages also connects the personal dashboard with reputation. Even when a user has no ratings yet, the page communicates that ratings are part of the system. When ratings exist, the average is shown, helping users understand how others evaluated their interactions.

The visual structure is repeated in a controlled way throughout the application. This gives users a stable sense of place while they move between registration, donation, browsing, messaging, and administration pages.

The page also supports ease of use by avoiding unnecessary form elements and unclear controls. Users can identify what they need to do, complete the action, and return to the previous area when necessary. This approach improves clarity and reduces the chance of user error.

4.2. My Items Page for Donors

The My Items page is mainly used by donors. It lists the items uploaded by the logged-in user and gives the donor a way to monitor their donations. For a new user, the page can be empty and displays a message such as No items yet. This empty state is important because it explains why no data is visible instead of leaving a blank page.

When the donor has uploaded items, each item is shown with image, name, description, category, status, and action buttons. The donor can open an item through View or delete it if necessary. The delete action includes a confirmation prompt to reduce accidental removal. The page also displays the donor’s average rating, which is calculated from ratings received by the user.

A very important part of My Items is the notification for new messages. If a receiver has contacted the donor about a specific item, the item card can show a label indicating a new conversation message. This turns My Items into more than an inventory page; it becomes a management and communication dashboard.

Technically, this page depends on ItemRepository.findByDonorUsername. The system retrieves items where donorUsername matches the logged-in principal. This guarantees that the donor sees only their own contributions. Message indicators depend on the conversation data related to each item.

For donors, the My Items page concentrates on objects they have published and on the responses those objects receive. Its item cards act as management records, allowing the donor to review each donation, its status, and available actions before taking a next step. This makes the page specific to donor-owned content rather than general profile editing.

The relationship with requests is presented from the donor's perspective: incoming interest is attached to the relevant donated item. This allows the donor to follow a receiver message without searching through unrelated activity, and it links communication with the correct object in the donation workflow.

The average rating shown in this area gives the donor a summary of feedback received through completed interactions. Here, reputation information complements item management by representing the donor's earlier participation and helping connect future exchanges with the history of user feedback recorded by the application.

The donor-receiver scenario is the strongest proof that the features are connected. The receiver discovers the biscuits item, contacts the donor, waits for an answer, confirms the meeting, and later can rate the donor. The donor sees the request through My Items, replies, changes status, and can rate the receiver.

The messaging system uses DonorMessage as the conversation summary and ChatMessage as the chronological message history. This separation allows My Items and My Requests to show compact summaries while the contact pages show the complete conversation.

4.3. My Requests Page for Receivers

The My Requests page is the receiver-side counterpart of My Items. It displays the requests and conversations initiated by the logged-in receiver. For a new user, the page can be empty. This matches the documented case where a newly registered account has no items or requests yet.

When the receiver contacts a donor, the system creates a DonorMessage record and associates it with the item, donor, and receiver. My Requests can then display that record as a request card. The receiver can see the item image, item name, donor username, and first message. If the donor replies, a new message indicator can appear, helping the receiver know that there is new activity.

The page supports continuity. Without My Requests, the receiver would need to search the category again, find the item again, and open the conversation manually. With My Requests, all initiated contacts are gathered in one place. This improves usability and makes the donation process easier to follow.

From a system design perspective, My Requests demonstrates how conversation data can be presented differently depending on the role of the current user. The same underlying DonorMessage and ChatMessage records are used, but the receiver views them as requests, while the donor views them as interest in their items.

In the context of My Requests, saved conversations are presented as receiver activity rather than profile data. The page therefore helps the receiver review previous requests, continue an open discussion, and understand which donation interactions still require attention.

The page also complements My Items by showing the opposite side of the same exchange. Instead of listing objects owned by the donor, it lists requests started by the receiver and keeps each request connected with the correct donor, item, and message history. This makes the receiver workflow visible and organised.

Rating information can support the receiver after a request has been completed. When feedback exists, it helps the user understand the reliability of the person involved in the exchange. In this subsection, the rating element is relevant because it is connected with receiver follow-up and trust.

This page therefore closes the gap between discovery and continued communication. After a receiver has expressed interest in an item, My Requests becomes the place where the user returns to check replies, reopen the discussion, and continue toward an agreed exchange.

By using the stored conversation records, the same communication can be shown from the receiver point of view without duplicating data. This keeps request tracking compact while still allowing the complete message history to remain available when the conversation is opened.

5. Main Dashboard Icons and Navigation Strategy

The initial authenticated dashboard uses three main icons in the center of the screen: Donate Items, Find What You Need, and Share With Others. These icons represent the main actions of the application. Their placement makes the user flow easy to understand immediately after login.

Key points:

- Donate Items represents the donor workflow.
- Find What You Need represents the receiver workflow.
- Share With Others represents the community engagement workflow.

The left icon leads to the donation form. It is primarily intended for donors who want to publish items. The middle icon leads to category browsing and is the most important path for receivers looking for goods. The right icon leads to the Community page, where users can share impressions and stories about using the application.

This three-icon design is effective because it transforms the main functional areas into simple choices. Instead of exposing every route in a large menu, the application presents the user with three meaningful paths. This is especially useful for a community donation platform, where users may not be technical and should not need to understand the backend structure.

The interface description shows that this design is supported by a clean layout and a consistent color scheme. Large action cards, short labels, and familiar icons help users navigate. From a usability perspective, the home page acts like a task-based dashboard. The user chooses what they want to do, and the system sends them to the correct workflow.

The item cards shown in category pages are designed to give enough information for quick selection. A receiver can inspect the image, name, description, category, and availability without opening every item. This is efficient and matches real browsing behavior on donation or marketplace platforms.

The status field is essential because it communicates whether the item can still be requested. In the documented examples, available and unavailable statuses appear in listings and details pages. This prevents receivers from wasting time contacting donors about objects that have already been assigned.

The item details page forms the bridge between browsing and communication. The Contact Donor button appears only for users who are not the donor. For donors, the same item page becomes a management screen with status control and receiver messages. This is a good example of conditional rendering based on user context.

The three main choices reflect the platform's organisation: offering an item, seeking a donation, or participating in the community. Presenting them together makes the application's purpose immediately apparent after login.

From the user experience side, the page is organized around clarity rather than complexity. Required actions are presented in a visible order, and navigation controls help users understand where they are in the system. This is important for a donation platform intended for a wide range of users.

5.1. Category Selection in Find What You Need

The Find What You Need feature begins with a category selection page. The application organizes donations into six categories: Clothing, Food, Books, Electronics, Furniture, and Others. This structure is visible in the described interface and is also supported by the category field in the Item entity.

Category-based navigation is useful because it reduces the amount of information shown at once. If all items were listed on one page, the user could become overwhelmed, especially as the database grows. By selecting a category first, the receiver narrows the search to items that match a real need.

Each category is represented visually, making the page easier to scan. The layout follows the same design identity as the dashboard. The categories also reflect practical donation types. Food and clothing cover basic needs, books support education, electronics and furniture cover household items, and Other provides flexibility for items that do not fit the predefined groups.

From the backend perspective, category filtering can be supported by repository methods such as findByCategoryIgnoreCase. This method retrieves only items matching the selected category. The filtered results are then passed to the Thymeleaf view. This is a clear example of how a user click on the frontend leads to a database query and then to a dynamic results page.

In this category-selection context, item cards act as the first filtered result view.
They help the receiver compare objects from the chosen category quickly by showing the most important visible details before the user decides whether to open a specific donation.
This keeps browsing efficient after the category choice.

Availability information is also important at this stage because category lists may contain items in different states. Showing status early helps the receiver understand which results are still realistic options and which ones should only be viewed as previous or unavailable donations.

The link from a category result to an item-details page creates the next step in the search process. After filtering by category, the receiver can inspect one item in detail and then decide whether communication with the donor is appropriate for that specific object.
This connects category navigation with the request workflow.

In category browsing, clarity depends on grouping donated objects according to practical need. The category cards shorten the search process and make it easier for receivers to reach relevant available items.

5.2. Item Listing Pages

After selecting a category, the user is redirected to a listing page showing the items that belong to that category. In the documented interface, category pages display item cards with images, titles, descriptions, category labels, and status. This card-based structure is appropriate for donated items because users often rely on visual inspection before opening an item.

The image is especially important. A receiver can immediately understand the condition or type of item. The title provides a short identifier, and the description gives additional context. The status field tells the user whether the item is available or unavailable. This avoids unnecessary contact attempts for items that have already been taken.

The listing page also supports incremental exploration. The user does not need to open every item. They can quickly scan the cards and decide which item is worth viewing in detail. This improves efficiency and

makes the platform feel responsive.

Technically, the listing page depends on dynamic data passed from the backend. The Thymeleaf template loops through item objects and renders each one. If the selected category contains no items, the page should ideally show an empty state. This pattern is similar to My Items and My Requests, where empty pages are explained to the user instead of remaining blank.

The listing view is organised for comparison rather than data entry. Images, descriptions, and status indicators give receivers enough information to identify promising donations before opening a details page.

The design keeps the user journey practical. Each visible control has a clear role, whether it is submitting information, opening details, returning to a previous page, or confirming an action. This makes the interface easier to learn and use.

For item listings, Thymeleaf is mainly used to repeat the visual card structure for each item returned by the selected category query. The template therefore turns repository data into a readable set of results, while still allowing each card to carry links and values that belong to its specific item.

The listing page also supports the browsing stage of the donation process. Its purpose is not only to show data, but to help the receiver move from a category result to a concrete item decision. This makes the page part of the discovery workflow rather than a decorative catalogue.

On the backend, this part of the system follows the controller-repository-entity pattern. The controller receives the request, the repository retrieves or saves data, and the entity defines the structure of the stored information. This pattern is repeated throughout the application and gives the project a clear organization.

5.3. Item Details Page and Conditional Actions

When the user clicks an item card, the Item Details page opens. This page displays a larger image, the item name, description, category, and status. It also includes a Back button so the user can return to the previous page. This structure gives the receiver enough information to decide whether to contact the donor.

The item details page uses conditional behavior. If the current user is not the donor, the page displays the Contact Donor button. If the current user is the donor, the page instead displays donor-specific actions such as changing status and viewing receiver messages. This means the same page adapts to the relationship between the logged-in user and the selected item.

This conditional rendering is important for role-aware interfaces. It would not make sense for a donor to contact themselves. Similarly, a receiver should not be allowed to change the item status. By showing different actions depending on isDonor, the system protects the workflow and makes the interface clearer.

The page also includes a messages section for donors. If receivers have contacted the donor about that item, their messages can be displayed with buttons to continue the conversation. This connects item management directly with the communication feature.

At this point, the user has already selected one listing card and needs more detail. The item-details page expands the selected record with a larger view, status information, and action buttons that depend on the current user.

This is efficient and matches real browsing behavior on donation or marketplace platforms.

Availability on the item-details page gives the receiver a final check before sending a message. It also gives the donor a clear visible state for the object they manage. This status information therefore supports both decision-making and donor control on the same detail screen.

The conditional action area is the main functional part of the details page. It separates the receiver action of contacting a donor from the donor action of managing the item. In this way, the page protects the workflow while still using one shared item view for different users.

For the details view, predictable controls support a clear decision: return to browsing, contact the donor, or manage an owned item. These actions preserve the distinction between receiver and donor responsibilities.

The page is structured so that users can complete the intended action without unnecessary distraction. Form fields, buttons, and navigation links are placed in a predictable way, which improves readability and makes the workflow easier to follow.

5.4. Item Entity and Repository Design

The Item entity represents donated objects in the database. It contains an automatically generated id, name, description, status, category, imageUrl, imagePath, donorUsername, and createdAt. These fields are enough to support browsing, item details, donation, donor ownership, and status management.

The status field is central to the item lifecycle. When a new item is donated, the DonateController sets status to available. Later, the donor can change it to unavailable after an agreement with a receiver. This status is shown in item cards and details pages, so all users see the current availability.

The donorUsername field connects the item with the user who uploaded it. This is used by My Items and by the contact workflow. When a receiver contacts a donor, the system reads the donor username from the selected item and creates a message addressed to that donor.

The ItemRepository extends JpaRepository, giving the application standard operations such as save, findAll, findById, and deleteById. It also defines custom methods such as findByCategoryIgnoreCase, findByStatus, and findByDonorUsername. These methods directly support the main user flows: browsing categories, filtering by status, and displaying a donor's own items.

From the entity perspective, the card information comes directly from stored Item fields. The same name, description, category, image path, and status values that appear in the interface are defined by the data model, which keeps the listing view connected to the stored record.

The status value is especially important inside the entity because it records the lifecycle state of the donation. It is not only a label for the interface; it is stored data that allows the application to show availability consistently in category pages, details pages, and donor management views.

The contact workflow also depends on entity data. When a receiver contacts a donor, the selected Item provides the donor username and item identifier used to create the related message record. The Item entity therefore supplies the information needed to connect browsing with communication.

The stored item record supports listing, detail, and communication workflows because consistent donor, status, image, and identifier values connect each function.

The interface reduces complexity by presenting the task in small, manageable steps. Users can focus on the current action without needing to understand the underlying database or controller logic. This separation between technical implementation and user-facing simplicity improves the overall experience.

5.5. REST ItemController and Data Operations

The ItemController exposes item operations through /api/items. It includes methods for retrieving all items, creating a new item, deleting an item by id, and resetting all items. Although the main user-facing pages use server-rendered Thymeleaf templates, the REST controller demonstrates that item data can also be managed through API endpoints.

The GET method returns all items from the repository. The POST method accepts an Item object in the request body and saves it. The @Valid annotation indicates that validation should be applied before persistence. The DELETE method removes one item by identifier, while the reset endpoint deletes all items. The reset route is likely useful during development or testing.

This controller shows separation between data operations and visual page rendering. The DonateController manages the donation form and image upload, while ItemController offers generic item operations. In larger applications, this kind of separation can be useful if the frontend later becomes a single-page application or if a mobile client needs access to the same data.

From a thesis perspective, the controller demonstrates basic REST principles: resources are represented by URLs, HTTP methods represent actions, and repositories handle persistence. It also shows that the item model is central to multiple layers of the system.

Through the REST controller, item information can be returned as data rather than as a rendered page. This means the same stored fields used by the card interface can also be served to another client if the application is extended beyond Thymeleaf pages.
The endpoint therefore exposes the item resource in a reusable form.

Status remains relevant in the API because it is part of the Item resource returned or saved through these operations. Any client using the endpoint must still understand whether an item is available or unavailable, so the API preserves the same lifecycle information used by the web interface.

The controller also demonstrates how item data can support different actions through standard HTTP methods. Retrieving, creating, deleting, or resetting item records are data operations, while user-facing communication remains handled by the server-rendered workflow. This separation keeps the API role clear.

The API operations complement rendered pages by exposing item records in a reusable form for other clients. This illustrates extensibility at the data-access boundary rather than only in appearance.

A practical usability strength is that the page avoids overloading the user with options. The most relevant actions are visible, while supporting controls such as Back buttons help users recover their position in the application. This supports confidence during navigation.

6. Donation Feature Overview

The donation feature is the main entry point for donors. From the home dashboard, the donor selects the left icon and reaches the Donate Item page. The form requires item name, description, category, and image upload. After submission, the item becomes visible to receivers in the selected category.

Key points:

- The donor adds item information, category, and image in one focused form.
- The backend automatically assigns status and donor ownership from the active session.
- The saved item becomes available for browsing and communication.

The feature is designed to be direct. The user does not need to go through multiple pages. The donation form collects all required information in one screen. The category select menu ensures that the item is classified correctly, and the required image upload improves the quality of item listings.

The documented workflow describes a realistic example where a donor fills in an item, uploads a photo, clicks Donate, and then sees the item appear in the corresponding category. This immediate result is important because it confirms the action. The donor can visually verify that the item has been published correctly.

The donation process also connects with later workflows. After an item is saved, it can be discovered through Find What You Need, opened in item details, requested through Contact Donor, tracked in My Items, and eventually marked unavailable. Therefore, donation is the beginning of the item lifecycle.

The donation workflow demonstrates how frontend and backend cooperate. The form collects the visible information, but the backend adds system-controlled values such as available status and donor username. This ensures that item ownership and item state are not manipulated manually by the user.

The image upload is important for trust. A receiver is more likely to request an item when they can see it. The application saves the file in an uploads folder and stores only the generated URL in the database, which is a common and practical design for web applications.

The redirect after donation improves feedback. When the donor adds biscuits in the Food category, the application redirects to the Food listing and shows the newly added item. This confirms success without requiring a separate confirmation page.

A donation form benefits from immediate confirmation because publishing an item begins every later interaction. Once stored, the donation becomes available for discovery, contact, and status management by authorised users.

The page design favors clarity and consistency. Labels, buttons, and navigation elements are used in a predictable manner, allowing users to understand the function of each area quickly. This is especially useful in a system that contains many different routes.

6.1. DonateController Backend Processing

DonateController manages both displaying the donation form and saving submitted items. The GET /donate method creates a new Item object and adds it to the model. If the Principal is available, it also adds the username to the model so the top-right menu can display the logged-in user.

The POST /donate method receives the submitted Item object, the uploaded MultipartFile, and the Principal. Before saving, the controller sets the item status to available. This is important because status is a system-controlled field. The donor does not choose it manually, which avoids inconsistent values.

The controller also assigns the donor username from principal.getName(). This is a key security and data integrity decision. The donor username is not taken from a form input, so the user cannot submit an item under another account. The backend always connects the item with the authenticated session.

After processing the image and saving the item with itemRepository.save(item), the controller redirects to /find-items/category. This creates a smooth workflow because the donor immediately sees the published item inside the category they selected. The redirect also confirms that the data was stored successfully.

In this controller, the backend contribution is the assignment of values that should not be chosen freely from the browser. The authenticated Principal supplies donor ownership and the controller sets the initial availability, so the saved item reflects the active user session and a valid starting state.

Image handling is also part of controller processing because the uploaded file must be checked, renamed, saved, and connected with the Item record. This creates the link between the submitted form file and the image path that later appears in item listings. The controller therefore prepares both data and presentation resources.

The redirect is useful here because it confirms the effect of the POST request. After persistence is complete, the user is not left on the submission form; the browser moves to the relevant category page where the new record can be viewed. This completes the backend processing with visible feedback.

The visual structure is repeated in a controlled way throughout the application. This gives users a stable sense of place while they move between registration, donation, browsing, messaging, and administration pages.

User interaction is kept simple by focusing each page on one main purpose. When users donate, browse, request, or manage information, the interface presents the fields and controls needed for that specific purpose. This helps reduce mistakes and keeps the workflow understandable.

6.2. Image Upload and File Handling

The donation form uses enctype="multipart/form-data" because it sends both text fields and an image file. The file input accepts images and is marked as required. This ensures that each donated item has a visual representation.

In the backend, the controller checks whether the file exists and is not empty. It creates a unique filename by combining the current timestamp with the original filename. This reduces the chance that two uploaded files with the same name overwrite each other. The file is then saved in an uploads directory.

The code creates the uploads folder if it does not already exist. This improves reliability because the application can run on a fresh environment without manually creating the

directory. After saving the file, the controller stores the URL path /uploads/filename in the item imageUrl field. The frontend later uses this value in image tags.

This design separates binary files from database records. The database stores only the path, while the file system stores the actual image. This is a common approach in web applications because it keeps database rows lighter and allows the web server or application to serve image files efficiently.

In the file-handling subsection, the important point is how text data and image data travel together through the multipart request. The controller receives both parts, stores the uploaded file separately, and links the resulting path with the saved Item record. This keeps the donation data complete without storing the binary file in the database.

The uploaded image also improves practical evaluation of an item because it gives receivers visual evidence before they send a request. For implementation, the important detail is that the path stored in imageUrl later becomes the source used by the rendered image element. This connects file storage with the visible item card.

After the file is saved, the following redirect confirms that the complete item was created with both descriptive fields and a usable image reference. This links the upload step with the receiver-facing category page and makes the result of file handling immediately visible.

Consistency is an important part of the interface design. The recurring layout and color scheme make the application easier to recognize and help connect the separate workflows into one complete system.

The page supports non-technical users by translating system operations into familiar interface actions. Instead of exposing backend details, it presents readable labels, direct buttons, and clear navigation paths. This makes the application more approachable.

6.3. Donation Form Usability and Validation

The Donate Item page has a simple and focused layout. The form fields appear in a logical order: item name, description, category, image file, and submit button. This order matches the way a donor thinks about describing an object. First they identify it, then explain it, then classify it, then upload a photo.

The required attributes on the name, category, and image fields prevent incomplete items from being submitted. The description field gives flexibility because some items need short text, while others need more details about quantity, condition, or pickup expectations.

The category dropdown limits values to the categories supported by the application. This is better than allowing free text because it prevents spelling variations such as food, Food, foods, or FOOD from creating inconsistent categories. The backend and category pages can then rely on predictable values.

After donation, automatic redirection to the selected category is a strong usability decision. The user does not need to wonder whether the item was added. They see it immediately in the same view that receivers will use. This reinforces trust in the system and connects the donor workflow with the receiver browsing workflow.

The donation workflow demonstrates how frontend and backend cooperate. The form collects the visible information, but the backend adds system-controlled values such as available status and donor username. This ensures that item ownership and item state are not manipulated manually by the user.

The image upload is important for trust. A receiver is more likely to request an item when they can see it. The application saves the file in an uploads folder and stores only the generated URL in the database, which is a common and practical design for web applications.

From a usability perspective, the final response after submission should make the result of the form obvious. Sending the user to the selected category provides confirmation that the form values were accepted and that the donated item entered the receiver-facing browsing

The completed submission path supports donor confidence by confirming that the item was accepted and placed in the expected category. This matters because later stages depend on a correctly published donation.

The interface remains usable because it does not require users to make unnecessary decisions. Important actions are clearly identified, and navigation aids help users return or continue when needed. This creates a more reliable interaction flow.

7. Community Feature - Share With Others

The Community feature is accessed through the third main icon on the home dashboard. Its purpose is different from item donation and item searching. Instead of managing objects, it allows users to share impressions, stories, and feedback about their experience with Give & Share.

The page includes a header, the authenticated user menu, a short introductory message, a grid of community posts, pagination controls, and a Share Your Story form. The form asks for name, role, and message. The role can be Donor, Receiver, or Both, matching the general role structure of the application.

This feature gives the platform a social dimension. Users can read how others used the application, which can increase trust and encourage participation. For example, a receiver may describe how an item helped them, while a donor may describe a positive experience giving something away.

Although the community feature does not directly complete an item exchange, it supports the overall purpose of the project. A donation platform benefits from visible community engagement because users are more likely to participate when they see that others have had meaningful interactions. The feature therefore strengthens the human side of the application.

The community feature supports emotional engagement with the system. Users can explain how the platform helped them, and these stories can encourage others to participate. This is useful because trust is important in a donation platform where people interact outside the screen as well.

Pagination is a practical implementation detail. The controller retrieves a fixed number of posts per page and sorts them by newest first. This keeps the page readable and prevents performance or layout problems when the number of stories increases.

The form for submitting a story is intentionally short. It asks for name, role, and message. This encourages participation because users do not need to complete a long process just to share their experience.

The community page supports engagement by allowing experiences to be read and contributed in one place. Its layout therefore balances browsing existing stories with submitting a new personal experience.

The page layout contributes to the overall accessibility of the system. By using concise fields, recognizable controls, and predictable navigation, it helps users complete tasks without confusion. These choices make the application easier to use in everyday situations.

7.1. Community Page Layout and Post Cards

The Community page uses a grid layout to display posts as cards. Each card shows the poster’s name, the role badge, the date, and the message. This layout is clean and readable, especially when there are many posts. The cards allow users to browse stories quickly without reading a long unstructured list.

The header keeps the same gradient style as other pages, and the user menu remains available in the top-right corner. This shows that the community page is part of the authenticated application and not a separate external page. The Back button allows users to return to the previous screen without losing orientation.

The Share Your Story section is placed after the existing posts. This order encourages users to read community content first and then contribute their own message. The form is simple, requiring only the information needed to create a post. This reduces barriers to participation.

The page also includes a footer with application copyright text. While simple, this gives the page a more complete website structure. Overall, the design supports both browsing and contribution, which are the two main actions of the community feature.

In this layout section, the post cards turn user experiences into short, readable records. The poster name, role badge, date, and message are separated clearly, so visitors can understand the context of each story without reading a long unstructured page.

Pagination also affects the layout because it prevents the community screen from becoming too long when more stories are added. By limiting the number of visible posts, the page remains readable while older content stays reachable through navigation controls.

The story form is placed as a simple contribution area rather than as a complex publishing tool. This supports participation because users can add a short experience quickly while the card layout keeps submitted content consistent.

The post-card layout gives this feature a clear structure by applying the same presentation to every contribution. Users can scan comparable stories, then decide whether to add their own experience.

The interface is designed to keep each task direct and understandable. Users are normally shown only the information and input fields that are relevant to the action they are performing. Clear button labels and return options support orientation and help reduce common mistakes, especially for users who are not technically experienced.

7.2. CommunityController and Pagination

CommunityController manages the community page. The GET /community method receives a page parameter with default value 0. If the page value is negative, it is corrected to 0. This prevents invalid negative pagination requests.

The controller retrieves posts using repository.findAll with PageRequest.of(page, 12, Sort.by("createdAt").descending()). This means each page displays up to 12 posts and the newest posts appear first. Sorting by createdAt descending is appropriate because users usually expect recent stories to be visible at the top.

The Page object is added to the model together with currentPage and totalPages. The Thymeleaf template uses these values to render pagination links. The user can move between previous and next pages when the number of posts grows.

The POST /community method receives name, role, and message from the form. It creates a new CommunityPost and saves it through the repository. Then it redirects back to /community. This redirect prevents duplicate form resubmission if the user refreshes the page and also displays the new post in the list. The implementation is simple but effective.

At controller level, the community feature is handled as a combination of retrieval and creation. The GET method prepares sorted posts for reading, while the POST method records a new story and returns the user to the same area.

Pagination is especially relevant in this subsection because it shows how the controller limits the data sent to the template. The page number, total pages, and sorted results support stable browsing as the number of posts increases.

From the controller perspective, story submission remains intentionally short. It receives name, role, and message, saves one post, and returns the user to the community page without an unnecessary process.

At controller level, usability depends on a short submission cycle: a post is received, saved, and returned to the community view for confirmation.

In terms of usability, the page keeps the interaction focused. The user is not required to process unnecessary options, and the available controls guide the next step clearly. This makes navigation easier and supports a smoother experience for non-technical users.

8. Full Donor and Receiver Interaction Overview

The most complete demonstration of the application is the interaction between a donor and a receiver. This scenario connects almost every major feature: login, category browsing, item details, contact donor, My Items, contact receiver, My Requests, status update, and rating.

Key points:

- The receiver discovers the item and starts communication with the donor.
- The donor replies, manages item availability, and coordinates the exchange.
- Both users can complete the interaction with ratings and continued accountability.

The example used in the documentation describes a food item called biscuits. The donor has already uploaded it with an image and status available. The receiver logs in, navigates to Food through Find What You Need, sees the biscuits item, opens its details, and chooses Contact Donor.

This scenario is valuable because it proves that the application is not only a collection of separate pages. The features work together as a complete donation workflow. An item can be published, discovered, requested, discussed, accepted, marked unavailable, and evaluated.

In a thesis, this use case is important because it shows the system from the perspective of real users. It explains how the software supports a practical social action. The user interface guides the participants, while the backend stores the conversation and controls access to the appropriate pages.

In this overview, the donor-receiver scenario connects the main workflow steps. The receiver finds the item and begins contact, the donor reviews the request and responds, and both participants can proceed through status management, completion of the exchange, and final feedback.

At overview level, the communication records connect the separate screens used by both participants. DonorMessage identifies the request, item, donor, and receiver, while ChatMessage stores each individual reply in chronological order for the detailed conversation history.

Read status is also part of this complete interaction. New-message indicators show when a user needs to respond, and opening the conversation updates the stored state without requiring live chat or continuous online communication.

At this overview stage, unity comes from workflow rather than appearance: a published item becomes the subject of a request, a discussion, a status decision, eventual feedback, and continuing administrative oversight.

The page follows a simple interaction model. Each form or action area includes only the details needed for that part of the workflow, while buttons and back-navigation links make the available choices easy to understand. This reduces confusion and supports practical use of the platform.

8.1. Receiver Starts the Conversation

The receiver begins by opening the item details page for the biscuits item. Because the receiver is not the donor, the page shows the Contact Donor button. This button redirects to /contact-donor/itemId. The contact page displays item, donor and rating information, earlier messages, and a text area.

The receiver writes an initial message asking to arrange a pickup. When the form is submitted, ContactDonorController retrieves the item by id. It checks whether a DonorMessage already exists for this item, donor, and receiver. If not, it creates a new DonorMessage record containing itemId, itemName, donorUsername, receiverUsername, imageUrl, message, and readByDonor false.

The controller also creates a ChatMessage record. This is important because DonorMessage stores the conversation summary, while ChatMessage stores individual messages in chronological order. The first receiver message is therefore both the first message of the conversation and the first chat entry.

After saving, the receiver is redirected back to the contact donor page. The conversation appears on the screen, confirming that the message was sent. This creates a clear starting point for the donation request.

For this subsection, the focus is the receiver's first action after selecting an item. The receiver's request is linked to the chosen item and its donor, so the interaction begins with the correct context, participants, and stored message information that will support later donor response and receiver follow-up.

The two stored message forms serve the opening step of the request. DonorMessage creates a summary visible to the donor, while ChatMessage records the receiver's initial message and preserves the chronological conversation when further replies are exchanged about the selected item.

The unread state supports the start of communication. Because the donor has not yet opened the request, the record can be marked as new and used by My Items to direct attention to the receiver's message when the donor next reviews donated items.

The contact page keeps the receiver's first action focused on one donated item and one message. This makes the beginning of the request clear before the conversation develops further.

Usability is supported through a clean structure and clear navigation. Instead of presenting many unrelated choices, the page guides the user toward the next useful action. This helps users complete tasks with fewer errors and without needing technical knowledge.

8.2. Donor Sees Notification in My Items

After the receiver sends a message, the donor logs in and opens My Items. The biscuits item now displays a notification indicating that there is a new message in the conversation for this item. This is one of the most useful interaction details in the application, because it tells the donor where attention is needed.

The donor does not need to search through all conversations manually. The notification is attached to the item that received interest. This is logical because the donor thinks in terms of their donated items. If they donated several objects, they can immediately see which one has a receiver message.

When the donor opens the item details page, the receiver message is shown in the receiver messages section. The donor can then click Contact Receiver to open the conversation. This path connects My Items with the messaging system.

The notification relies on read-state fields. DonorMessage has readByDonor, and the system can mark it as read when the donor opens the conversation. This prevents the same message from remaining marked as new forever. The mechanism is asynchronous but practical for this application.

In this donor-side subsection, the request becomes visible to the owner of the item. The item card operates as both a donation record and a notification that a receiver has expressed interest, allowing the donor to recognise which item requires attention, open its related discussion, and decide on a response.

The conversation summary is useful in My Items because the donor does not need the full discussion immediately. A compact request indication identifies the interested receiver and selected item, while the contact page remains available for reviewing and answering the detailed exchange.

The read-status field supports the notification behavior of My Items. When the donor opens the request, it can be marked as viewed, preventing an old message from remaining displayed as new while preserving the conversation for a later reply.

The visual structure is repeated in a controlled way throughout the application. This gives users a stable sense of place while they move between registration, donation, browsing, messaging, and administration pages.

The layout is intentionally straightforward. Input fields are limited to the needs of the current task, and action buttons use clear wording. As a result, the page remains accessible and helps users move through the donation workflow with confidence.

8.3. Donor Replies and Accepts the Request

The donor replies from the Contact Receiver page. This page is only accessible to the donor associated with the DonorMessage. The ContactReceiverController checks whether donorMessage.getDonorUsername() equals principal.getName(). If not, the user is redirected away. This protects conversations from unauthorized access.

The page displays the item name, receiver username, receiver rating, and chat history. The donor writes a reply, for example confirming a meeting time and place. When the form is submitted, the controller creates a ChatMessage with the donor as sender and the receiver as recipient. The message is linked to the same donorMessageId.

The donor can also accept the exchange by changing the item status from available to unavailable. This is done from the item details page using Change Status. The status update is important because it communicates to other users that the item is no longer available.

This step shows how the donor has control over their donated item. The donor can respond to requests, decide whether to proceed, and update the public item state. The system supports the donor’s ownership while still enabling receiver communication.

In this reply subsection, the exchange moves from receiver interest to donor decision. The donor answers through the same conversation, decides whether to proceed with the request, and updates the public item status when the donation is reserved, accepted, or no longer available.

The stored message history supports the donor's answer by preserving the receiver's request and every following reply. This allows both users to continue one organised discussion associated with the same donated item, request context, and developing exchange process without fragmenting communication into unrelated records.

After the donor replies, read status assists the receiver. A new reply can remain marked as unread until the receiver opens the request, so My Requests clearly shows that the discussion has changed and requires the receiver's attention.

The receiver-facing request area is organised around follow-up: locating a discussion, noticing a reply, and continuing the arrangement. This keeps attention on an interaction already initiated.

The page also supports ease of use by avoiding unnecessary form elements and unclear controls. Users can identify what they need to do, complete the action, and return to the previous area when necessary. This approach improves clarity and reduces the chance of user error.

8.4. Receiver Follows the Conversation in My Requests

After the donor replies, the receiver can see the new message in My Requests. This page gathers all requests initiated by the receiver. The request card includes the item image, item name, donor username, first message, and a notification when there is a new reply.

The receiver opens the conversation through Open Conversation. The contact donor page now shows the full chat history, including the original receiver message and the donor’s reply. The receiver can then answer and confirm the meeting. In the provided scenario, the receiver confirms attendance and thanks the donor.

My Requests is important because it preserves continuity. The receiver does not need to remember the category or search for the item again. All active conversations are accessible from one personal page. This improves the usability of the receiver side of the platform.

The page also supports read-state logic. When the receiver opens the conversation, messages addressed to them can be marked as read. This keeps notifications accurate and prevents old messages from repeatedly appearing as new.

In this subsection, My Requests concerns receiver follow-up rather than profile editing. It gives the receiver a personal area for reviewing requests already made, returning to active conversations, checking donor responses, and continuing communication about selected items until the exchange is completed.

The receiver-side page complements My Items without repeating its donor purpose. Instead of managing uploaded objects, it organises contacts initiated by the receiver and keeps the item, donor, initial message, later replies, request status, and continuing exchange context linked to every interaction.

Rating information becomes relevant in this area after an exchange is completed. Showing reputation with receiver activity can support trust in continuing communication and connect completed requests with the feedback recorded for the user involved in the completed donation experience.

The receiver workflow is therefore a continued path rather than a single contact action. The user discovers an item, sends a request, receives a donor reply, returns through My Requests to coordinate the final details, and can later provide feedback after the actual exchange has been completed.

The stored conversation records support this continuity by keeping every reply associated with the selected item and donor. The receiver can reopen the same discussion, inspect its history, continue the exchange without recreating a request, and avoid searching for the item again.

8.5. Rating After the Exchange

After the donor and receiver complete the arrangement, both users can rate each other. The receiver can rate the donor from the contact donor page, and the donor can rate the receiver from the contact receiver page. The interface uses stars, which is a familiar rating method for users.

The RatingRepository retrieves ratings by ratedUsername. The system calculates the number of ratings and the average score. The average is then formatted, for example as one decimal, and displayed together with star symbols. This appears on pages such as contact donor, contact receiver, My Items, and My Requests.

Ratings support trust. A donor with positive ratings may appear reliable to receivers. A receiver with positive ratings may appear respectful and punctual to donors. Since the application involves real interaction between people, feedback helps future users make decisions.

The rating system is also connected to community quality. If users know they can be evaluated, they may behave more responsibly. From a technical perspective, ratings are stored as separate records, which allows averages to be recalculated dynamically as new feedback is added.

In this rating-focused subsection, the outcome of the donor-receiver exchange is the important element. After communication, agreement, and handover are completed, both participants can provide feedback that contributes to the reputation displayed in later interactions.

Ratings relate to the completed exchange through the users involved rather than through a repeated account of chat storage. This keeps the subsection focused on feedback while allowing reputation information to appear when later donors or receivers consider communication.

Read status is less central in the rating stage, but it supports the path leading to completion. Clear notification of replies helps users finalise arrangements, complete the handover, and then submit meaningful feedback about their interaction.

A rating interface should make final feedback easy to understand because users complete it after the practical exchange. Clear star input translates their experience into recorded reputation data.

From the user experience side, the page is organized around clarity rather than complexity. Required actions are presented in a visible order, and navigation controls help users understand where they are in the system. This is important for a donation platform intended for a wide range of users.

8.6. Message Entities and Read Status

The communication system uses two main message-related entities: DonorMessage and ChatMessage. DonorMessage represents the overall conversation about an item between one donor and one receiver. It stores item id, item name, donor username, receiver username, image URL, first message, readByDonor, and creation date.

ChatMessage represents each individual message inside the conversation. It stores the donorMessageId, sender username, receiver username, message content, creation date, and read state. This separation is useful because the system can display conversation summaries in My Items and My Requests while still preserving detailed chat history.

Read status improves the interface. For donors, readByDonor helps show whether a receiver's first message is new. For receivers, unread chat ids help identify new replies. When a user opens the relevant conversation, the controller marks messages addressed to that user as read.

This design provides message notifications without requiring real-time WebSocket technology. The application behaves asynchronously: users check their pages, see indicators, open conversations, and continue communication. For the scope of the project, this is a suitable and understandable implementation.

In this entity subsection, the donor-receiver interaction explains why two message models are useful. One record identifies the overall conversation concerning an item, donor, and receiver, while the other stores every individual message exchanged between the participants during the continuing request.

The message model therefore supports both summary and detailed conversation views. My Items and My Requests can display concise request information, while the contact pages retrieve the complete chronological discussion connected to the same conversation and item record.

Here, read-status fields are a technical feature of the message entities. They let the same stored conversation produce different visual notifications according to whether the current participant has already reviewed the latest message.

For message management, continuity is created by stored identifiers and read-state fields. These allow participants to reopen the correct discussion and recognise activity that still requires attention.

The design keeps the user journey practical. Each visible control has a clear role, whether it is submitting information, opening details, returning to a previous page, or confirming an action. This makes the interface easier to learn and use.

8.7. Complete Donation Lifecycle

The full donation lifecycle begins when a donor uploads an item. The item is saved with status available and appears in the selected category. A receiver discovers the item through category browsing and opens the details page. If interested, the receiver contacts the donor.

The system creates a conversation and stores the first message. The donor receives a notification in My Items and opens the conversation. The donor replies, accepts the request, and may change the item status to unavailable. The receiver sees the reply in My Requests and confirms the meeting.

After the exchange, both users can rate each other. The rating results then appear in relevant pages. The administrator can also inspect users, items, and conversations from the admin dashboard. This means the system supports both user interaction and oversight.

This lifecycle demonstrates the practical value of the application. It is not only an item database. It includes discovery, communication, state changes, feedback, and management. These parts together form a complete web application appropriate for a master-level Informatics project.

Within the lifecycle, donation is only the starting stage. An uploaded item becomes useful to the platform when it can be discovered by a receiver, requested, discussed through messages, updated in status, exchanged successfully, and connected with later user feedback and administrative visibility.

Across the complete lifecycle, the item image remains valuable because it helps a receiver evaluate an object before requesting it. At this point, the image supports item discovery and an informed decision rather than explaining the file-upload procedure.

The redirect after donation also begins the lifecycle by making the new item immediately visible in its selected category. From that listing, the item can proceed into receiver contact, donor agreement, status change, completion, and rating.

In this final lifecycle view, responsibility passes between the participants. The donor provides an item, the receiver expresses interest, both users coordinate the arrangement and handover through communication, and each participant can finish by leaving feedback that informs later platform interactions.

The message entities keep the lifecycle connected to the correct item and users. This allows both personal dashboards to show the relevant conversation, preserves the history of communication, and prevents the exchange process from becoming detached from the donation record.

Read status completes the communication stage by showing whether a participant still needs to review a message. This keeps the interaction manageable and understandable even when the donor and receiver are not online together at the same time.

9. Admin Account and Role Assignment

The admin feature adds a management layer to the application. For development and testing, a dedicated administrator account was prepared and then updated in the database so that the stored user record received ROLE_ADMIN authority and ADMIN functional role values. This describes the setup without exposing unnecessary login credentials in the thesis text and keeps the focus on the role assignment process.

- The admin role protects management functions from normal users.
- The dashboard provides oversight of users, items, and conversations.
- Delete actions are checked on the backend before changing the database.

This approach demonstrates how stored user roles can control access. The role field is used for security authority, while userRole also allows the application to recognize the account as an administrator. The AdminController checks both possibilities through the isAdmin method.

The admin page is available at /admin. A normal user cannot access it. If a user who is not admin enters the URL, the controller redirects them away. The documented error case shows that an unauthorized user cannot reach the admin dashboard. Although a custom access denied page would be more polished, the protection itself is visible.

The administrator can return to the normal dashboard through Back to Home. This is useful because the admin account can both manage the system and inspect the result of actions from the normal application pages. The admin feature therefore completes the system with supervision functionality.

The admin feature demonstrates role-based management. A normal account was converted into an administrator by updating the users table so that the role became ROLE_ADMIN and userRole became ADMIN. The AdminController checks these values before allowing access to /admin.

The dashboard gives the administrator control over users, items, and conversations. Each table has a clear purpose: users can be removed, items can be viewed or deleted, and conversations can be deleted together with their chat messages. These actions help maintain the quality and safety of the platform.

The unauthorized access case is also important. It shows that a regular user cannot simply type /admin and access the dashboard. The current redirect could be improved with a custom access- denied page, but the main security requirement is already implemented.

The administrative area requires familiar navigation because it includes higher-impact actions. Clear record grouping and authorised access help the administrator review content before making decisions.

The page is structured so that users can complete the intended action without unnecessary distraction. Form fields, buttons, and navigation links are placed in a predictable way, which improves readability and makes the workflow easier to follow.

9.1. Admin Dashboard Structure

The admin dashboard uses a large container with three management sections: All Users, All Items, and All Conversations. Each section is displayed as a table. This design is suitable for administration because structured records are easier to inspect in rows and columns.

The header shows Admin Dashboard and displays the admin username in the top-right area. This confirms that the current session belongs to the administrator. The Back to Home button provides navigation back to the standard dashboard.

The All Users table includes id, username, email, role, user role, full name, and a delete action. The All Items table includes id, item name, category, status, donor, view, and delete action. The All Conversations table includes id, item, donor, receiver, first message, and delete conversation action.

The design follows the same visual language as the rest of the application. It uses the gradient header, white background, rounded container, and clear buttons. Even though the admin page has more dense information, it remains consistent with the user interface style.

In the dashboard structure subsection, role checking explains why this page is shown only after administrator validation. The controller prepares the management view only for an authorised session, so table data is not exposed to ordinary users or mixed with the normal donation screens.

The three dashboard tables organise supervision into separate areas. User records, donated items, and conversations are placed in different sections, allowing the administrator to inspect each type of platform content without mixing unrelated actions or losing the purpose of each table.

Access protection remains in the background of this page. Its practical effect is that the dashboard layout is prepared only when the current account has the expected management role and authority values.

In the dashboard, clarity comes from dividing records into tables for users, items, and conversations, allowing the administrator to review each content type before acting.

The interface reduces complexity by presenting the task in small, manageable steps. Users can focus on the current action without needing to understand the underlying database or controller logic. This separation between technical implementation and user-facing simplicity improves the overall experience.

9.2. User Management in Admin Panel

User management gives the administrator visibility over all registered accounts. This is important because user data is the foundation of the application. Donors, receivers, and admin accounts all exist in the same users table, but their roles determine what they can do.

The admin can delete a user by clicking the Delete button in the row. The button sends a POST request to /admin/delete-user/□id□. Before the deletion, the interface asks for confirmation. This confirmation reduces the risk of accidental deletion.

The backend checks admin rights before performing the delete operation. This is essential because deleting users is a sensitive action. If the Principal is null or the username does not belong to an admin, the method redirects the user away instead of deleting the account.

This part of the admin panel could be extended in the future with role editing, account disabling, detailed user profiles, or activity logs. However, even in its current form, it shows the ability to supervise registered accounts and remove problematic records.

In user management, role-based control is important because account deletion affects every later workflow connected to that user. The controller therefore checks administrator status before allowing the delete request to continue and before modifying the stored user table or related account data.

This subsection focuses on registered accounts rather than all dashboard sections. The user table lets the administrator review identity values, roles, and profile information before deciding whether a record should remain in the system or be removed from active use during supervision.

The protection is especially relevant for this action because a non-admin user must not be able to remove accounts by manually calling a delete URL or changing request parameters outside the interface.

For user supervision, the page makes the account record and available action easy to inspect before deletion. This supports responsible handling of user data and role-based control within the administrative process.

A practical usability strength is that the page avoids overloading the user with options. The most relevant actions are visible, while supporting controls such as Back buttons help users recover their position in the application. This supports confidence during navigation.

9.3. Item Management in Admin Panel

The All Items section allows the administrator to inspect every donated item in the system. Each row shows the id, name, category, status, and donor username. The View button opens the normal item details page, allowing the admin to see the same content users see.

The Delete button allows the administrator to remove an item directly. This is useful if an item is inappropriate, duplicated, incorrectly uploaded, or no longer valid. The action is handled by /admin/delete-item/□id□, and the backend uses itemRepository.deleteById(id).

Item management is important because a donation platform depends on the quality and accuracy of item listings. If donors upload invalid content, the administrator needs a way to correct the system without relying only on the donor.

The admin item management page also shows the benefit of using repositories. The same ItemRepository supports user-facing features such as category browsing and admin features such as deletion. This reuse keeps the data access layer organized and consistent.

The admin feature demonstrates role-based management. A normal account was converted into an administrator by updating the users table so that the role became ROLE_ADMIN and userRole became ADMIN. The AdminController checks these values before allowing access to /admin.

The dashboard gives the administrator control over users, items, and conversations. Each table has a clear purpose: users can be removed, items can be viewed or deleted, and conversations can be deleted together with their chat messages. These actions help maintain the quality and safety of the platform.

This protection belongs to the broader admin design: ordinary users should not reach management routes or inspect conversation records outside their own exchanges and permitted pages.

In moderation, clear presentation is necessary because communication records may require removal. Showing participants, item, and first message helps the administrator act with appropriate context.

The page design favors clarity and consistency. Labels, buttons, and navigation elements are used in a predictable manner, allowing users to understand the function of each area quickly. This is especially useful in a system that contains many different routes.

9.4. Conversation Management in Admin Panel

The All Conversations section displays the conversations created between donors and receivers. Each record shows the conversation id, item name, donor username, receiver username, and first message. This gives the administrator a summary of communication activity inside the platform.

The admin can delete an entire conversation. The backend implementation is careful because a DonorMessage may have many ChatMessage records. Before deleting the DonorMessage, the controller deletes all chat messages associated with the same donorMessageId. This prevents orphaned messages from remaining in the database.

Conversation management is useful for moderation. If a conversation is inappropriate, irrelevant, or created by mistake, the administrator can remove it. This gives the system owner control over communication records.

The method is annotated with @Transactional. This is important because deleting related records should happen as one database operation. If part of the deletion fails, transactional behavior can help keep the database consistent. This is a good technical detail to mention in the thesis because it shows awareness of persistence integrity.

For conversation management, role checking protects private communication records and deletion actions. Only an authorised administrator should be able to inspect the summary of exchanges and remove records when moderation or correction is required by the platform and its users.

The conversation table has a moderation purpose. It shows the item, donor, receiver, and first message, giving the administrator enough context to identify irrelevant, mistaken, or inappropriate exchanges before deletion or further investigation inside the panel when needed for system supervision.

The unauthorized access case is also important. It shows that a regular user cannot simply type /admin and access the dashboard. The current redirect could be improved with a custom access- denied page, but the main security requirement is already implemented.

The visual structure is repeated in a controlled way throughout the application. This gives users a stable sense of place while they move between registration, donation, browsing, messaging, and administration pages.

User interaction is kept simple by focusing each page on one main purpose. When users donate, browse, request, or manage information, the interface presents the fields and controls needed for that specific purpose. This helps reduce mistakes and keeps the workflow understandable.

9.5. Admin Security and Unauthorized Access

Admin security is implemented inside AdminController using the isAdmin method. This method searches the user by username and checks whether the role equals ROLE_ADMIN or the userRole equals ADMIN. Only then can the user access the admin dashboard or perform delete actions.

Every admin action repeats the same validation. The dashboard route checks admin status before loading data. The delete user, delete item, and delete message routes also check admin status before performing changes. This prevents a normal user from bypassing the dashboard and calling a delete URL directly.

The documented case where a non-admin user tries to access /admin and receives an error demonstrates that ordinary accounts do not reach the admin panel. A future improvement would be to redirect to a custom access-denied page or home page instead of producing a generic Whitelabel Error Page. However, the important functional requirement is satisfied: access is exclusive to the admin account.

This feature shows role-based access control at application level. It is especially important because the admin dashboard contains destructive operations. Without this validation, data could be modified by unauthorized users.

In this security subsection, the main focus is the isAdmin validation used before every sensitive route. The same check is applied when loading the dashboard and before deleting users, items, or conversation records from the database layer during administration of protected content.

Because the admin panel contains destructive actions, the dashboard is treated as a protected management area. Its tables are useful only when combined with backend checks that confirm the user has administrative authority before changes occur in storage or records are removed.

The unauthorized-access case is kept here as the direct security example. A regular account cannot rely on the URL alone to reach /admin, because the controller verifies the current account first.

A protected administrative page must prioritise controlled action rather than ordinary navigation. Management controls are useful only after backend validation confirms an authorised session.

The page supports non-technical users by translating system operations into familiar interface actions. Instead of exposing backend details, it presents readable labels, direct buttons, and clear navigation paths. This makes the application more approachable.

10. Database Design and Entity Relationships

The application uses several entities that represent the core data. User stores account and profile information. Item stores donated object information. DonorMessage stores the summary of a request conversation. ChatMessage stores individual chat entries. Rating stores user feedback. CommunityPost stores public stories shared by users.

Key points:

- Users, items, messages, ratings, and community posts form the core data model.
- Identifiers and usernames connect records across the donation lifecycle.
- The design is simple enough to explain clearly while supporting the required features.

The relationships are implemented mainly through ids and usernames. Item contains donorUsername, linking an item to the user who donated it. DonorMessage contains itemId, donorUsername, and receiverUsername, linking a request to a specific item and two users. ChatMessage contains donorMessageId, linking each chat entry to a conversation. Rating contains the rated username, allowing the system to calculate average ratings for each user.

This design is simple and readable. It avoids complex object graphs while still supporting the required features. For a thesis project, this is acceptable because the relationships are easy to explain and directly tied to user workflows.

A possible future improvement would be to use explicit JPA relationships such as @ManyToOne between Item and User or between ChatMessage and DonorMessage. However, the current username/id approach works correctly for the implemented pages and makes repository queries straightforward.

A future version could add more advanced functionality such as real-time chat, email delivery for password reset links, better upload validation, search filters, item sorting, location-based matching, and stronger admin reporting. These improvements would build on the current architecture rather than replacing it.

The existing implementation is a good foundation because the main entities and workflows already exist. Future features can reuse users, items, messages, categories, ratings, and admin management. This shows that the project is extensible.

The value of this data model lies in its support for the donation lifecycle. Stored links between users, items, messages, ratings, and posts make each workflow retrievable and explainable.

The interface remains usable because it does not require users to make unnecessary decisions. Important actions are clearly identified, and navigation aids help users return or continue when needed. This creates a more reliable interaction flow.

At database level, Thymeleaf is not the main focus; the important point is that entity fields provide the values later rendered by templates. Usernames, ids, status values, ratings, timestamps, and message references are stored consistently so pages can present reliable data.

10.1. UI and UX Consistency

The user interface is consistent across the application. Most pages use the same font, the same blue-orange gradient header, white content cards, rounded corners, and subtle shadows. This visual consistency makes the system feel unified even though it contains many different features.

Forms are centered and simple. Registration, login, password reset, profile editing, donation, and community story submission all follow a clear layout. Buttons use strong colors and hover effects, making interactive elements easy to identify. Error and success messages are styled differently, helping users understand the result of their actions.

Navigation is also consistent. The top-right username menu is present on many authenticated pages, and Back buttons help users return to previous screens. The three dashboard icons simplify the main decision points. Empty states such as no items yet, no requests, or no ratings yet prevent confusion when a new account has no data.

From a usability perspective, the system does not overload users with technical details. Instead, it translates backend actions into understandable pages: donate, find, contact, request, rate, and manage. This is one of the strengths of the application.

In the UI and UX section, the main issue is not adding new functions but keeping existing interactions predictable. Consistent cards, forms, menus, feedback messages, and return buttons help users understand how to move through different workflows without unnecessary learning effort.

This consistency makes the current interface easier to learn because donation, browsing, profile, and admin screens share recognisable patterns. Users do not need to relearn the layout for every feature.

Consistency here is not decorative only: repeated navigation positions, form structures, and feedback states allow users to transfer what they learned on one feature to later actions with less confusion.

The page layout contributes to the overall accessibility of the system. By using concise fields, recognizable controls, and predictable navigation, it helps users complete tasks without confusion. These choices make the application easier to use in everyday situations.

Thymeleaf supports this consistency by allowing templates to reuse the same visual structures while filling them with different model values. The result is a dynamic interface that still feels unified across pages and remains connected to backend data during everyday use of the platform.

10.2. Technical Strengths and Limitations

The main technical strength of the application is that it implements a complete workflow using standard web development practices. Controllers manage requests, repositories manage data access, entities represent tables, and Thymeleaf templates render dynamic content. The system also includes file upload, pagination, authentication, password reset tokens, role checking, message read status, and rating calculation.

Another strength is the separation of donor and receiver workflows. The donor uses Donate Item and My Items, while the receiver uses Find What You Need and My Requests. The communication system connects the two workflows without mixing responsibilities.

There are also limitations that could be addressed in future work. Password reset links are shown in the console instead of being emailed. Unauthorized admin access produces a generic error rather than a custom page. Messaging is asynchronous and requires users to refresh or open pages to see updates. File upload validation could be stronger by checking file size and type on the server.

These limitations do not reduce the value of the project. Instead, they show realistic areas for extension. A master thesis can present both what has been implemented and what could be improved in a production version.

These limitations identify concrete production concerns rather than weaknesses in the implemented workflow. Email delivery, stronger upload checks, clearer access-denied handling, and more immediate messaging would improve robustness and deployment readiness while preserving the current architecture.

The technical structure can support such improvements because the current controllers, repositories, and entities already separate responsibilities clearly enough for gradual extension and maintenance.

These limitations are presented with the implemented strengths so the evaluation remains balanced. They identify deployment improvements without implying that the core donation workflow is incomplete.

The interface is designed to keep each task direct and understandable. Users are normally shown only the information and input fields that are relevant to the action they are performing. Clear button labels and return options support orientation and help reduce common mistakes, especially for users who are not technically experienced.

In this subsection, Thymeleaf is relevant as part of the implemented stack. It renders dynamic content from backend models, while the technical limits mainly concern deployment, validation, messaging immediacy, security feedback, and error handling in production use of the system.

10.3. Future Improvements

The Give & Share application could be improved in several ways. First, real-time notifications could be added using WebSocket technology. This would allow donors and receivers to receive messages instantly without manually checking My Items or My Requests.

Second, the password recovery system could be connected to an email service. Instead of printing the reset link in the console, the application could send a secure email to the registered address. This would make the system more suitable for deployment.

Third, the item browsing feature could include search and advanced filters. Users could filter by city, condition, newest items, or availability. Since the Item entity already stores category, status, donor username, and creation date, the system has a foundation for more advanced queries.

Fourth, the admin panel could be expanded with editing actions, user suspension, logs, and statistics. For example, the admin could see how many items are available, how many conversations exist, or which categories are most active.

Finally, the user interface could be adapted for mobile devices. Since donation and receiving may happen in everyday situations, mobile accessibility would make the platform more practical.

For future work, these enhancements can be grouped into communication, security, browsing, administration, and interface accessibility. Organising them in this way shows how each improvement extends a specific part of the platform rather than repeating existing functions.

The proposed extensions can reuse the existing foundation. Users, items, messages, categories, ratings, and admin records already provide the data needed for search, reports, notifications, and richer moderation.

Future interface work should extend practical tasks such as filtering, notification, reporting, and mobile access rather than focusing on appearance alone.

In terms of usability, the page keeps the interaction focused. The user is not required to process unnecessary options, and the available controls guide the next step clearly. This makes navigation easier and supports a smoother experience for non-technical users.

Future interface improvements would still depend on dynamic rendering. New filters, status indicators, mobile layouts, or admin statistics could use model values in templates while preserving the existing visual language and user flow of the platform during expansion and maintenance.

11. Conclusions - Summary

The Give & Share application demonstrates a complete community-based donation system. It allows users to register, log in, manage profiles, donate items, browse categories, view item details, contact donors, manage requests, communicate through messages, update item status, rate each other, share community stories, and perform administrative management.

The documentation shows that the application is not only a collection of web pages. It is a connected system where each feature supports the donation lifecycle. Registration and login create secure access. The dashboard provides navigation. Donation creates new resources. Category browsing helps receivers find items. Messaging enables coordination. Status updates keep availability accurate. Ratings build trust. Community posts encourage engagement. Admin tools provide supervision.

The implementation uses Spring Boot, Thymeleaf, repositories, entities, controllers, and database persistence. The code follows a clear structure where each controller has a specific responsibility. The frontend remains consistent across pages, and the backend stores the data required for each user flow.

Overall, the project satisfies the goals of a practical master-level web application. It combines social purpose with technical implementation and provides a foundation that could be extended into a larger platform. The application promotes sustainability, reuse, and cooperation, while also demonstrating important software engineering concepts such as MVC architecture, authentication, file upload, persistence, role-based access, and user-centered design.

These improvements are mentioned in the conclusion as possible continuation rather than as missing core functionality. Real-time chat, email reset delivery, stronger filters, and richer reporting would extend the platform after the completed thesis version while respecting the current structure.

The current implementation remains a solid base because the principal workflows already operate together. Future additions can build on the existing entities, controllers, repositories, and user-facing pages instead of replacing the system.

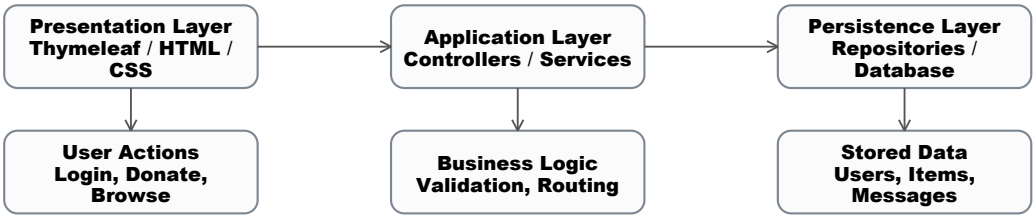
The final contribution is the completed interaction supported by the software: reusable goods can be published, requested, discussed, transferred, evaluated, and supervised through one application.

The page follows a simple interaction model. Each form or action area includes only the details needed for that part of the workflow, while buttons and back-navigation links make the available choices easy to understand. This reduces confusion and supports practical use of the platform.

12. Appendix - Summary Diagrams

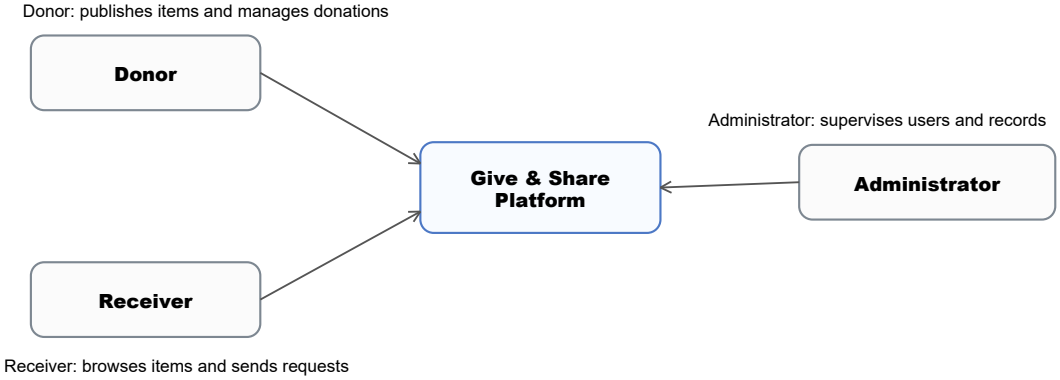
The following diagrams summarize the structure, roles, and workflows of the Give & Share application.

Figure 12.1. System Architecture Overview



This diagram shows the layered structure that connects the user interface, backend logic, and database storage.

Figure 12.2. User Roles and Functional Areas

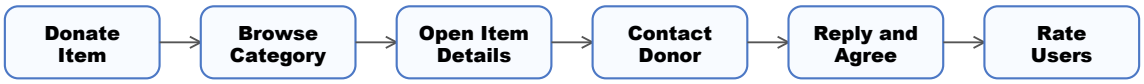


The platform supports donor, receiver, and admin roles, each linked to different functional responsibilities.

12.1. Workflow and Data Diagrams

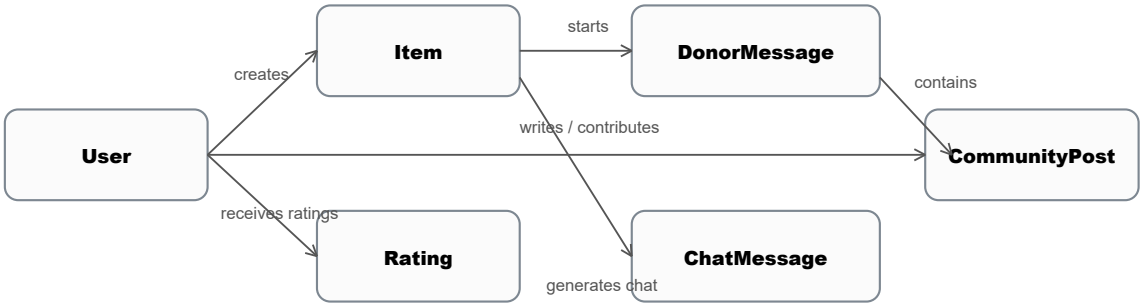
These diagrams summarize the operational flow of donations and the relationships between the core data entities.

Figure 12.3. Donation and Communication Workflow



The workflow begins when a donor uploads an item and ends when both participants complete the exchange and provide ratings.

Figure 12.4. Core Data Entity Relationships



- User stores profile and authentication data.
- Item stores donated object information and donor ownership.
- DonorMessage summarizes a request conversation for one item.
- ChatMessage stores the chronological discussion between users.
- Rating stores user feedback and average-score inputs.

13. Bibliography

The following official documentation sources and academic articles support the technologies, concepts, and research background used in the Give & Share web application, including Spring Boot, secure web development, persistence, frontend implementation, accessibility, artificial intelligence, smart applications, mobile systems, machine learning, and recommendation-related research.

[1] Spring Boot Reference Documentation.
Available at: https://docs.spring.io/spring-boot/index.html

[2] Spring Security Reference Documentation.
Available at: https://docs.spring.io/spring-security/reference/index.html

[3] Spring Data JPA Reference Documentation.
Available at: https://docs.spring.io/spring-data/jpa/reference/

[4] Thymeleaf Template Documentation.
Available at: https://www.thymeleaf.org/documentation.html

[5] Oracle. Java Platform, Standard Edition Documentation.
Available at: https://docs.oracle.com/en/java/

[6] Mozilla. MDN Web Docs: HTML.
Available at: https://developer.mozilla.org/en-US/docs/Web/HTML

[7] Mozilla. MDN Web Docs: CSS.
Available at: https://developer.mozilla.org/en-US/docs/Web/CSS

[8] Mozilla. MDN Web Docs: JavaScript.
Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[9] Oracle. MySQL Documentation.
Available at: https://dev.mysql.com/doc/

[10] W3C. Web Content Accessibility Guidelines (WCAG) 2.2.
Available at: https://www.w3.org/TR/WCAG22/

[11] Tselepatiotis Michail, Efthimios Alepis - QUESTA: A Hybrid IRT-Thompson Sampling Algorithm for Adaptive Quizzes Applied to Career Recommendation. IISA 2025: 1-9.

[12] Maria Virvou, George A. Tsihrintzis, Efthimios Alepis - Emotional Advice Compared Between an Affective Intelligent Authoring Tool and Generative AI: A Cognitive Walkthrough in Medical Mobile Tutor and ChatGPT. IISA 2025: 1-8.

[13] Aristea Kontogianni, Efthimios Alepis - AI in Smart Tourism: LLMs & GPTs Leading the Way. IISA 2024: 1-8.

[14] Tselepatiotis Michail, Efthimios Alepis - Adaptive Learning in Mobile Serious Games: A Personalized Approach Using AI for General Knowledge Quizzes. IISA 2024: 1-8.

[15] Nancy Alonistioti, Evangelia-Aikaterini Tsichrintzi, Konstantina Chrysafiadi, Efthimios Alepis - Requirements for Fuzzy Logic in Personalisation of Fire Emergency Alerts. IISA 2023: 1-8.

[16] Anargyros Douladiris, Efthimios Alepis - Covid-19 New Cases Correlation Analysis: Weather Conditions, Citizen Traffic and Vaccination Statistics Impact in NARX Estimated Regressions in Attica, Greece. IISA 2023: 1-7.

[17] Aristea Kontogianni, Efthimios Alepis - Social Network Data Enabling Smart Tourism. IISA 2023: 1-6.

[18] Tselepatiotis Michail, Efthimios Alepis - Design of Real-Time Multiplayer Word Game for the Android Platform Using Firebase and Fuzzy Logic. IISA 2023.

[19] Vasileios Argyropoulos, Efthimios Alepis, Constantinos Patsakis - Semi-Decentralized File Sharing as a Service. IISA 2022: 1-8.

[20] Aristea Kontogianni, Efthimios Alepis - AI, Blockchain & Cyber tourism joining the Smart Tourism realm. IISA 2022: 1-6.

[21] Athanasios Giannikis, Efthimios Alepis, Maria Virvou - Crowdsourcing Recognized Image Objects In Mobile Devices Through Machine Learning. ICTAI 2021: 560-567.

[22] Efthimios Alepis, Maria Virvou, Polychronis Kontomaris - Covid-19 Mobile Tracking Application Utilizing Smart Sensors. IISA 2021: 1-8.

[23] Aristea Kontogianni, Efthimios Alepis - Moments of Interest: A novel cloud-based crowdsourcing application enhancing smart tourism recommendations. CEEC 2019: 144-149.

[24] Jose Torres, Sergio de los Santos, Efthimios Alepis, Constantinos Patsakis - Behavioral Biometric Authentication in Android Unlock Patterns through Machine Learning. ICISSP 2019: 146-154.

[25] Jose Torres, Sergio de los Santos, Efthimios Alepis, Constantinos Patsakis - User Behavioral Biometrics and Machine Learning Towards Improving User Authentication in Smartphones. ICISSP (Revised Selected Papers) 2019: 250-271.

[26] Charalampos Houlis, Constantinos Patsakis, Efthimios Alepis - Smart Android Application using Self-Destructive Identities against Cyber Harassment. IISA 2019: 1-7.

[27] Dimitrios P. Panagoulias, Persephone Papatheodosiou, Anastasios Bonakis, Dimitris G. Dikeos, Maria Virvou, George A. Tsihrintzis - MemoryBERT: A Systematic Framework for Memory Identification. BIBE 2025: 711-717.

[28] Maria Virvou, George A. Tsihrintzis, Vangelis Marinakis, Elissaios Sarmas, Dimitrios P. Panagoulias, Evangelia-Aikaterini Tsichrintzi - Trustworthy Integration of Generative AI and LLMs in the Energy Digital Spine: Epistemic Risk Models and Trust Performance Indicators. ICTAI 2025: 130-138.

[29] Dimitrios P. Panagoulias, Elissaios Sarmas, Vangelis Marinakis, Maria Virvou, George A. Tsihrintzis - A Simulation Framework for Battery Optimization Under Extreme Energy Imbalance Using Machine Learning Forecasting. ICTAI 2025: 1272-1278.

[30] Konstantina Chrysafiadi, Spyros Papadimitriou, Maria Virvou - Fuzzy Logic for Evaluating the Acquisition of Soft Skills Through Adaptive Scenarios in Game-Based Learning: The Case of EPATHLO. IISA 2025: 1-6.

[31] Panagiota Ismini Matthe, Maria Virvou - Trustworthiness of an LLM in the Classroom Through a Role-Playing Game: The Case of ChatGPT. IISA 2025: 1-8.

[32] Maria Virvou, George A. Tsihrintzis - Requirements for Fair and Trustworthy AI: Overcoming Clustering and Stereotype Bias Through Individualized User Modeling. IISA 2025: 1-7.

[33] Maria Virvou, George A. Tsihrintzis, Konstantina Chrysafiadi, Evangelos Sakkopoulos, Evangelia-Aikaterini Tsichrintzi - Bi-Directional 21st Century Skills for Generative AI in Educational Software: A Requirements Engineering Perspective. IISA 2025: 1-10.

[34] Maria Virvou, George A. Tsihrintzis, Evangelia-Aikaterini Tsichrintzi - Hallucinations in Generative AI: Epistemic Risks for Learners in Educational Applications. IISA 2025: 1-8.

[35] Dimitrios P. Panagoulias, Maria Virvou, George A. Tsihrintzis - Truth in Trees: A Multilayered Linguistic Framework for Automated Fact-Checking with AI-Driven Reasoning. KES 2025: 1032-1041.

[36] Dimitrios P. Panagoulias, Anastasios P. Palamidas, Maria Virvou, George A. Tsihrintzis - An OSCE-Inspired Framework for Fine-Tuning Multimodal LLMs in Medical Diagnosis. KES 2025: 1042-1050.

[37] Konstantina Chrysafiadi, Spyros Papadimitriou, Maria Virvou - A Fuzzy-Logic Based Cognitive Walkthrough to Assess the Degrees of Soft Skills Targeted in an Intelligent Educational Adventure Game. AI (2) 2025: 347-361.

[38] George A. Tsihrintzis, Elissaios Sarmas, Vangelis Marinakis, Dimitrios P. Panagoulias, Evangelia-Aikaterini Tsichrintzi, Maria Virvou - Which AI and How Trusted by Human Energy Stakeholders? Comparing Generative AI ChatGPT With Domain Specific AI-ENERGIA-SYS. SMC 2025: 7066-7073.

[39] Dimitrios Mathioudakis, Dionisios N. Sotiropoulos, George A. Tsihrintzis - A mathematical analysis of the Genetic-AIRS classification algorithm. EUSIPCO 2016: 26-30.

[40] Dionisios N. Sotiropoulos, Demitrios E. Pournarakis, George M. Giaglis - A genetic algorithm approach for topic clustering: A centroid-based encoding scheme. IISA 2016: 1-8.

[41] Christos Bilanakos, Dionisios N. Sotiropoulos, Ifigeneia Georgoula, George M. Giaglis - Optimal Influence Strategies in Social Networks. ASONAM 2015: 856-863.

[42] Dionisios N. Sotiropoulos, Christos Bilanakos, George M. Giaglis - A Time-Variant and Non-Linear Model of Opinion Formation in Social Networks. ASONAM 2015: 872-879.

[43] Dionisios N. Sotiropoulos, Demitrios E. Pournarakis, George M. Giaglis - Tracking the evolution of communities in co-authorship networks: A semantically aware approach. IISA 2015: 1-6.

[44] Ifigeneia Georgoula, Demitrios Pournarakis, Christos Bilanakos, Dionisios N. Sotiropoulos, George M. Giaglis - Using Time-Series and Sentiment Analysis to Detect the Determinants of Bitcoin Prices. MCIS 2015: 20.

[45] Dionisios N. Sotiropoulos, Demitrios E. Pournarakis, George M. Giaglis - Semantically aware time evolution tracking of communities in co-authorship networks. Panhellenic Conference on Informatics 2015: 97-102.

[46] Nikos Bouros, Dionisios N. Sotiropoulos, Demitrios Pournarakis, George M. Giaglis - Social Network Analysis Within The ICMB Community: Co-Authorship Networks. ICMB 2014: 8.

[47] Dionisios N. Sotiropoulos, Dimitrios Mathioudakis, George A. Tsihrintzis - Genetic-AIRS: A hybrid classification method based on Genetic Algorithms and Artificial Immune Systems. IISA 2014: 1-5.

[48] Dionisios N. Sotiropoulos, Chris D. Kounavis, Panos E. Kourouthanassis, George M. Giaglis - What drives social sentiment? An entropic measure-based clustering approach towards identifying factors that influence social sentiment polarity. IISA 2014: 361-373.

[49] Dionisios N. Sotiropoulos, Christos Giannoulis, George A. Tsihrintzis - A comparative study of one-class classifiers in machine learning problems with extreme class imbalance. IISA 2014: 362-364.