CSIS 4495 - 002 - Applied Research Project

Final Report

NATIVESPARK INTEGRATION SOLUTIONS CORP.

Valeriia Nikitina (300374609)

Contents

Introduction

The origin of the idea

Originally developed by Valeriia Nikitina and Valeriya Saltykova, NativeSpark Integration Solutions Corp. emerged from their unique expertise and shared vision. Although the platform was initially a collaborative effort, Valeriia Nikitina will now continue to develop the idea and platform independently.

This transition marks a new chapter for NativeSpark, as Valeriia takes the lead in refining the platform's features, expanding its capabilities, and advancing its mission. By leveraging her background in data analysis and her commitment to empowering Indigenous communities, Valeriia aims to ensure that NativeSpark evolves into a robust, scalable solution that meets the current needs of its users while adapting to future challenges.

The project will maintain its original vision of fostering Indigenous entrepreneurship, with a strong focus on enhancing user experience, integrating advanced technologies, and exploring growth opportunities in the United States market. Valeriia is dedicated to preserving the collaborative spirit of the initiative while elevating the platform to new heights.

Problem framing

NativeSpark is dedicated to helping Indigenous communities overcome barriers to participation in industry. Many Indigenous entrepreneurs lack access to essential resources, mentorship, and training necessary for successful competition [1]. Additionally, Indigenous products often struggle to reach mainstream markets, and cultural and regulatory obstacles further complicate participation.

Indigenous entrepreneurs face several systemic barriers that impede their ability to engage effectively in industrial and commercial opportunities, including:

- Many Indigenous communities are situated in remote areas, making it challenging for entrepreneurs to access the funding, tools, and infrastructure needed to grow their businesses.
- Entrepreneurs frequently lack mentors who understand their unique challenges, as well as training programs tailored to their cultural and business contexts.

- Indigenous products and services often encounter difficulties navigating regulatory frameworks and facing cultural biases that prevent them from accessing mainstream markets.
- Ongoing economic exclusion has perpetuated cycles of poverty, limiting opportunities for Indigenous entrepreneurs to thrive.

These challenges are further exacerbated by the impact of social determinants of health, as highlighted in research by Richmond et al. (2017) in Social Science & Medicine. The study underscores that structural inequities, including limited access to education, employment, and economic opportunities, disproportionately affect Indigenous communities. These inequities diminish quality of life and hinder participation in economic systems, perpetuating cycles of poverty in marginalized communities [2, 3].

Competitive advantage among others

Existing platforms like the Indigenous Industrial and Contracting Network (IMCN) and Indigenous Business Development Services (IBDS) have made significant strides in supporting Indigenous entrepreneurs. However, they lack advanced technological features that could greatly enhance their effectiveness.

IMCN offers resources and networking opportunities but does not utilize AI-driven tools to optimize connections between businesses and entrepreneurs. On the other hand, IBDS emphasizes business development services but does not make use of predictive modeling or natural language processing (NLP) to improve decision-making and communication [4, 5].

These limitations hinder these platforms from effectively meeting the complex needs of Indigenous entrepreneurs, such as identifying market opportunities, facilitating collaboration, and offering personalized recommendations. NativeSpark aims to bridge these gaps by integrating:

- AI-Powered Matchmaking: Automatically connecting businesses with the most relevant Indigenous entrepreneurs based on their skills, location, and project requirements.
- NLP Capabilities: Enabling smooth communication across diverse linguistic and cultural backgrounds.

By addressing these gaps, NativeSpark strives to create a more comprehensive and inclusive platform for Indigenous entrepreneurs.

Overview of the platform

The platform will serve as a bridge connecting three distinct types of users—Indigenous Peoples, businesses, and regular consumers—while also integrating an administrative role for monitoring and oversight. Its core mission is to empower Indigenous communities by providing a space to showcase their craftsmanship, connect with opportunities, and facilitate transactions in an engaging and supportive environment.

Indigenous Peoples will be central to the platform, showcasing their unique skills and products through detailed profiles that include craftsmanship information and photos. They can sell items individually or in bulk and browse job postings and bulk order requests from businesses. A social networking feature will promote visibility through likes and comments.

Businesses can place large-scale orders or post projects, specifying requirements like quantity and materials, and will pay a commission on transactions. Communication with sellers for customizations is encouraged.

Regular consumers can purchase unique, handmade items, also paying a commission. They can communicate with sellers for details or customizations.

The administrator role is vital for platform integrity, overseeing user activity, moderating content, and resolving disputes to ensure a positive experience.

Social Networking for Indigenous Profiles:

Users will create portfolio-like profiles with photos, descriptions, and reviews, promoting community engagement and talent recognition.

Custom Requests:

Business users have the option to request specific quantities of products using a custom order form linked from the product details page.

Recommendation Engine:

A Python-based recommendation system suggests similar products by utilizing a cosine similarity algorithm implemented with scikit-learn.

Entrepreneur Insights:

Entrepreneurs can access information about buyers who have purchased their products, helping to foster trust and facilitate communication.

AI-Powered Matchmaking:

AI will connect businesses with Indigenous individuals based on skills, ratings, location, and availability, while consumers receive personalized product recommendations using historical data.

Integrated Chatbot with Escalation to Admins:

A chatbot will address common inquiries and guide users, escalating unresolved issues to admins for further assistance.

Admin Dashboard:

Administrators will monitor transactions, moderate content, and manage chatbot escalations to ensure a safe and effective platform.

The architecture of the system

In the development of NativeSpark's platform, we will adopt the Model-View-Controller (MVC) architecture model coupled with MySQL integration. This architectural approach is widely recognized and utilized in web application development due to its several advantages in terms of organization, scalability, and maintainability.

The MVC architecture separates an application into three interconnected components: Model, View, and Controller. Each component has specific responsibilities and interacts with the others to ensure efficient functionality and seamless user experience.

The hardware and software configuration of the system

Hardware Requirements

- Processor: Intel Core i5 or higher processors are recommended for smooth performance during development tasks.
- Memory (RAM): A minimum of 8GB RAM is recommended to handle the resource intensive tasks of running servers, databases, and development tools simultaneously.
- Network Connectivity: Stable internet connectivity is essential for accessing external resources, libraries, and version control systems during development.

Software Configuration

1) Development Environment:

- Integrated Development Environment (IDE): IntelliJ IDEA will be used as the primary IDE for Java development. It provides robust features for Java programming, Spring framework support, and seamless integration with version control systems. (Douglas College Credentials)
- Version Control: Git will be utilized for version control management, allowing collaborative development, code review, and version tracking.
- Build and Dependency Management: Maven will be used for managing dependencies and building the project.

2) Programming Languages and Frameworks:

- Java: The backend logic and business rules of the application will be developed using Java programming language.
- Spring Framework: Specifically, the Spring Boot framework will be used for rapid application development, dependency injection, and MVC architecture implementation.
- Thymeleaf: Thymeleaf will serve as the templating engine for generating dynamic HTML content in the View layer.
- Bootstrap: Bootstrap will be used for front-end styling, responsiveness, and user interface design.
- Python: Python is utilized for data seeding, database interactions, and developing a product recommendation engine.
- SQLAlchemy: This Python library is used for Object-Relational Mapping (ORM) access to MySQL database tables and models.
- Faker: Faker is a Python library that generates mock data (such as users, products, transactions, etc.) for testing and development purposes.
- scikit-learn: This is Python's machine learning library, which is employed to implement the product recommendation engine based on similarity metrics.
- JavaScript: JavaScript allows for interactive client-side features, such as save/unsave toggle buttons and dynamic updates to the user interface.
- HTML & CSS: These technologies are used in conjunction with Thymeleaf and Bootstrap to create the markup structure and apply custom styling.

3) Database Management System (DBMS):

- MySQL: The MySQL relational database management system will be used to store and manage application data. It provides a robust and scalable solution for structured data storage.

Changes made to the original proposal

Features

Original	Final
AI-Powered Matchmaking	Not implemented
Integrated Chatbot	Not implemented
Admin Dashboard	Not implemented
Social networking features	Not implemented
Custom product requests	Added
Saved listings for users	Added
Search, filter functionality	Added
Product recommendation system	Added
Registration of 3 user types	Done
Business (add/edit/delete job/project)	Done
Indigenous Entrepreneur (add/edit/delete product)	Done
Full product lifecycle (add to cart different items, checkout)	Done

Technology

Original	Final
AI libraries: OpenAI API	Not used
Chatbot integration (Bootstrap widgets)	Not used

Python, Python-based tools	Added
Spring Boot, Thymeleaf, Bootstrap	Used
Programming languages (Java, JavaScript)	Used
MySQL	Used

Reasons for change

1. **AI Matchmaking and Chatbot**

Reason: The complexity of implementing these features, combined with limited time, made it challenging. They required natural language processing and third-party integrations (such as OpenAI), which would have significantly expanded the project's scope.

Justification: The focus shifted to completing the core platform functionality and product flows first before exploring advanced AI features.

2. **Admin Dashboard**

Reason: Priority was given to user-facing features to ensure that at list one end-user functionality was fully developed.

Justification: Given the scope and timeline of the semester project, the development of admin functionality was deferred to a later phase after project submission.

3. **Addition of a Python-Based Recommendation**

Reason: To provide meaningful user personalization without depending on OpenAI.

Justification: A lightweight, local solution utilizing cosine similarity and category/price metrics was implemented to offer relevant product suggestions, which was more feasible within the available timeframe.

4. **New Simple Features (Search, Filters, Saved Listings)**

Reason: These features were identified during development as critical for enhancing user experience.

Justification: Based on typical e-commerce platforms and feedback from peers and instructors, these features were prioritized to make the platform more functional and realistic.

5. **Technology Usage Shift**

Reason: Some proposed tools were deemed unnecessary or overly complex for the final project scope.

Justification: Simpler and more manageable solutions were adopted, such as writing the recommendation engine in Python instead of using external APIs.

6. **Custom Product Order Requests Feature**

Reason: Businesses have expressed a need to request specific quantities or variations of products directly from Indigenous sellers. Additionally, when stock runs out, basic users and other Indigenous entrepreneurs should have the option to place orders for custom quantities.

Justification: This feature enhances support for bulk orders and B2B interactions, aligning with the platform's mission to empower Indigenous entrepreneurship by fostering direct and customizable engagement between businesses.

Project Planning and Timeline

Planning and Proposal

Milestone: Proposal Approval (Jan 26)

Deliverables: Approved proposal document, initial project plan, and timeline allocation.

System Design

Milestone: Finalize System Architecture (Feb 9)

Deliverables: Database schema and Interface design.

Development Phase

Milestone: Initial Working Demo (Feb 24)

Deliverables: Partially functional prototype demonstrating core features and database integration, Video showing a demo of the implementation, midterm report.

Milestone: Advanced Feature Implementation (Mar 30)

Deliverables:

1. **User Authentication and Role Management:**
    - Registration and login for users;
    - Role-based access control: Entrepreneur, Buyer, Business;
    - Dynamic navbar and page visibility based on user role.
2. **Product Listing & Details:**
    - Products shown on the home page;
    - Each product links to a detail page.
3. **Saved Listings Feature:**
    - Save/unsave functionality for:
        - Products;
        - Jobs;
        - Projects;

- Saved items visible in a "Saved" section per user.

4. **Job and Project Postings:**
    - Jobs and Projects posted by businesses visible to Entrepreneurs.
5. **Search Functionality:**
    - Keyword-based search filtering.
6. **Filtering:**
    - Products filterable by dynamically loaded categories from the database;
    - Jobs filterable by employment type (Full-time, Part-time, Internship).
7. **Custom Order Request Form:**
    - Businesses and other users can submit requests for custom product quantities.
8. **Recommendation Of The Product;**
9. **Full Product Lifecycle: From Browsing to Purchase:**
    - **Browsing Products:**
    - **Adding to Cart:**
    - **Cart Management:**
    - **Checkout Process:**
    - **Order History & Details.**

Testing Phase

Milestone: Refinement, testing and design (April 6)

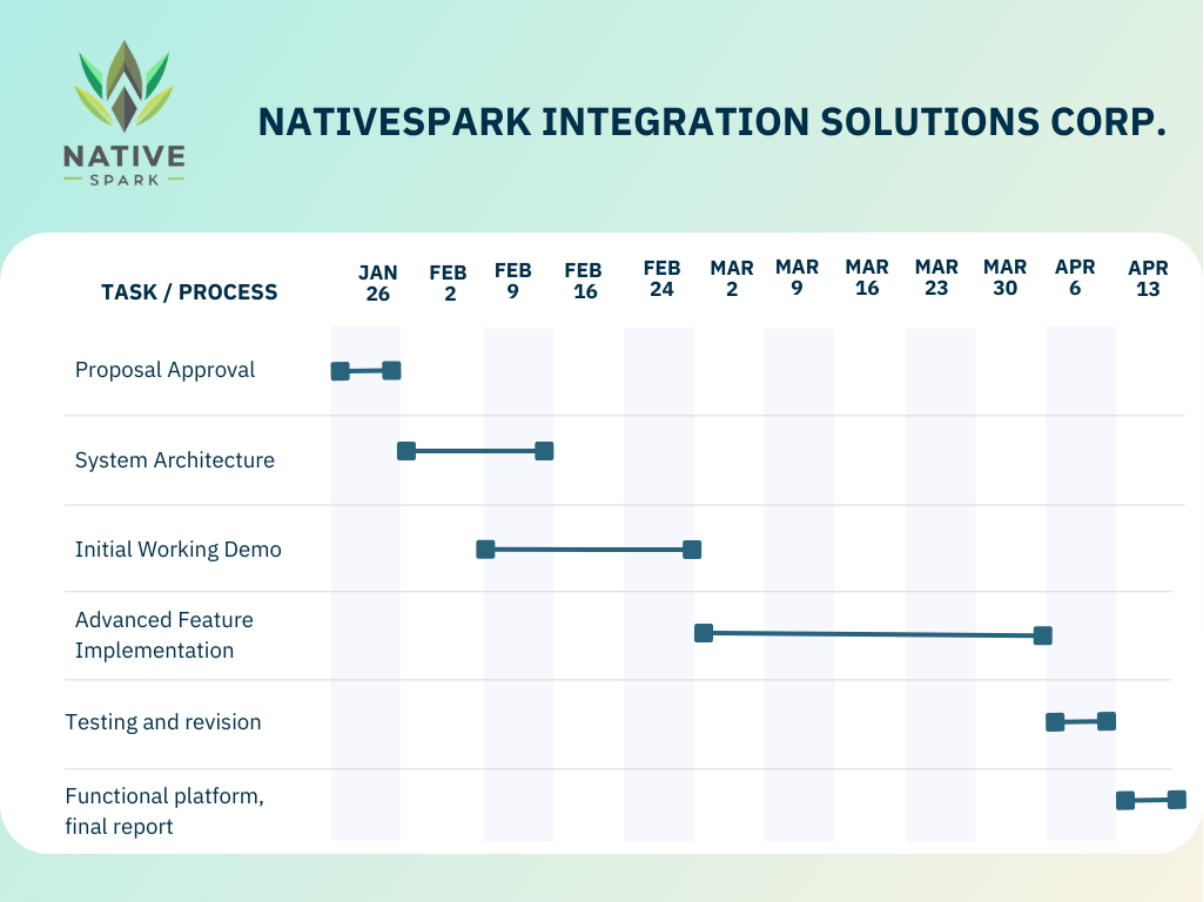
Deliverables: Usability testing and revision.

Final Submission and Presentation

Milestone: Final Submission and Presentation (Apr 13)

Deliverables: Functional platform, final report.

Gantt Chart


NATIVE
SPARK
NATIVESPARK INTEGRATION SOLUTIONS CORP.
TASK / PROCESS
JAN 26
FEB 2
FEB 9
FEB 16
FEB 24
MAR 2
MAR 9
MAR 16
MAR 23
MAR 30
APR 6
APR 13
Proposal Approval
System Architecture
Initial Working Demo
Advanced Feature Implementation
Testing and revision
Functional platform, final report

Implemented Features

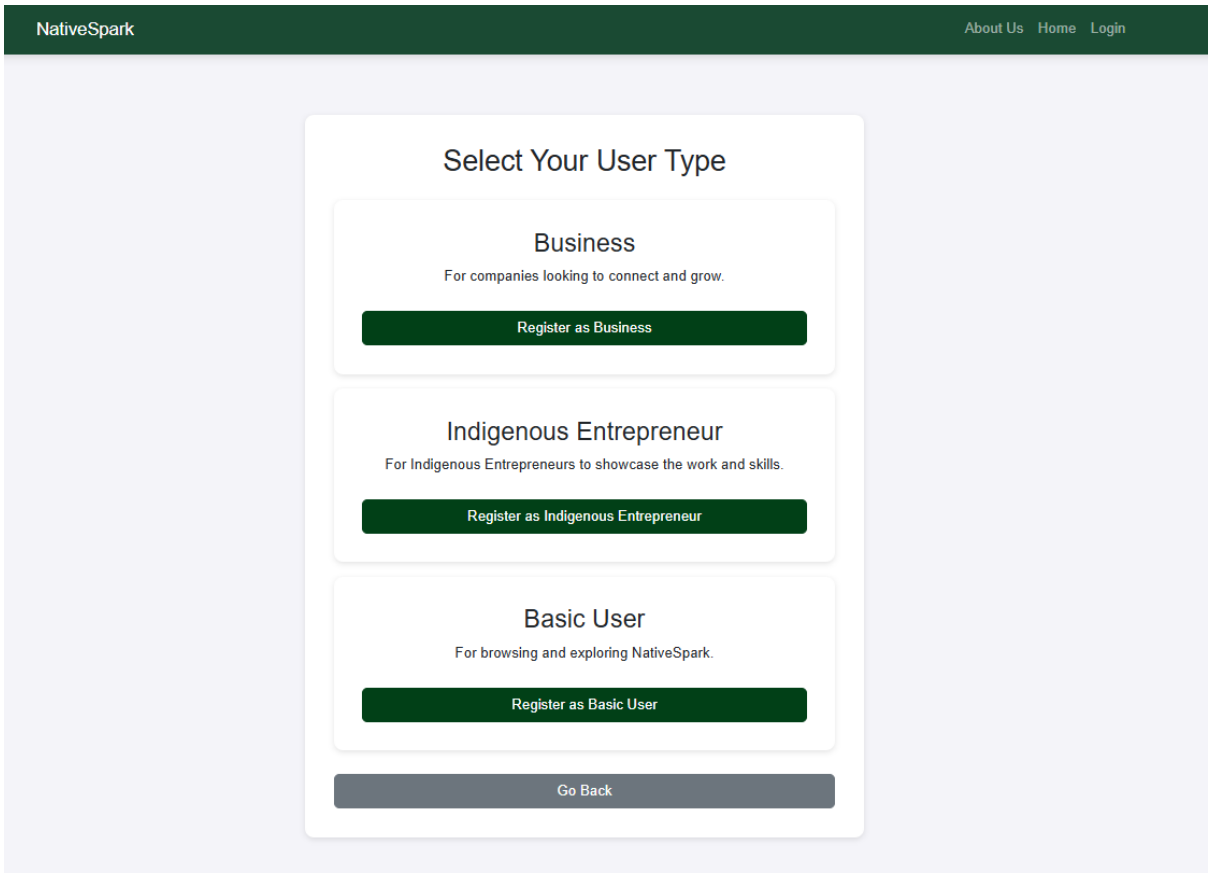
User Registration for All Three User Types

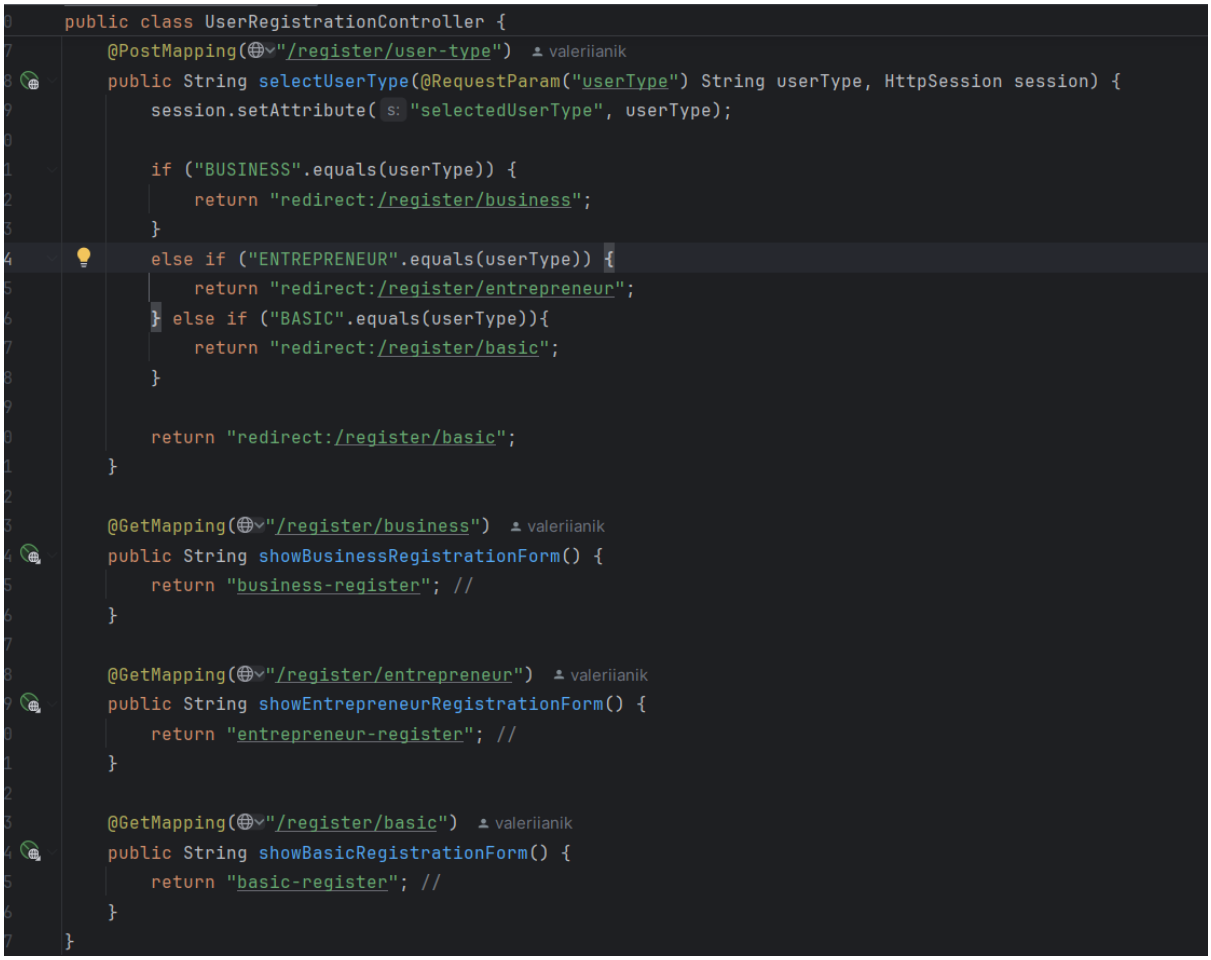
The platform supports registration for three different user roles:

- **Basic User**
- **Business User**
- **Entrepreneur User**

The registration process consists of **two steps**:

- **Step 1:** The user provides basic details such as email and password.
- **Step 2:** Additional details are collected based on the user type.



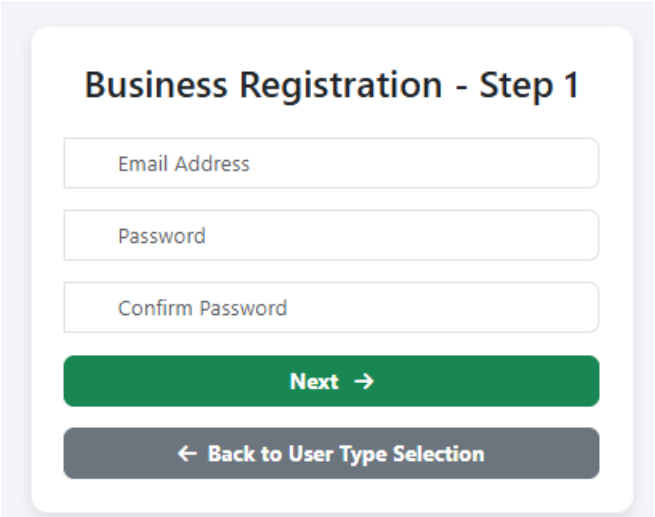


```java
public class UserRegistrationController {
    @PostMapping("/register/user-type")  valeriianik
    public String selectUserType(@RequestParam("userType") String userType, HttpSession session) {
        session.setAttribute( s: "selectedUserType", userType);

        if ("BUSINESS".equals(userType)) {
            return "redirect:/register/business";
        }
        else if ("ENTREPRENEUR".equals(userType)) {
            return "redirect:/register/entrepreneur";
        } else if ("BASIC".equals(userType)){
            return "redirect:/register/basic";
        }

        return "redirect:/register/basic";
    }

    @GetMapping("/register/business")  valeriianik
    public String showBusinessRegistrationForm() {
        return "business-register"; //
    }

    @GetMapping("/register/entrepreneur")  valeriianik
    public String showEntrepreneurRegistrationForm() {
        return "entrepreneur-register"; //
    }

    @GetMapping("/register/basic")  valeriianik
    public String showBasicRegistrationForm() {
        return "basic-register"; //
    }
}
```

Step 1: Basic Registration Form

Each user registers with a unified form, where they select their role and provide credentials.

Example of Business Registration:

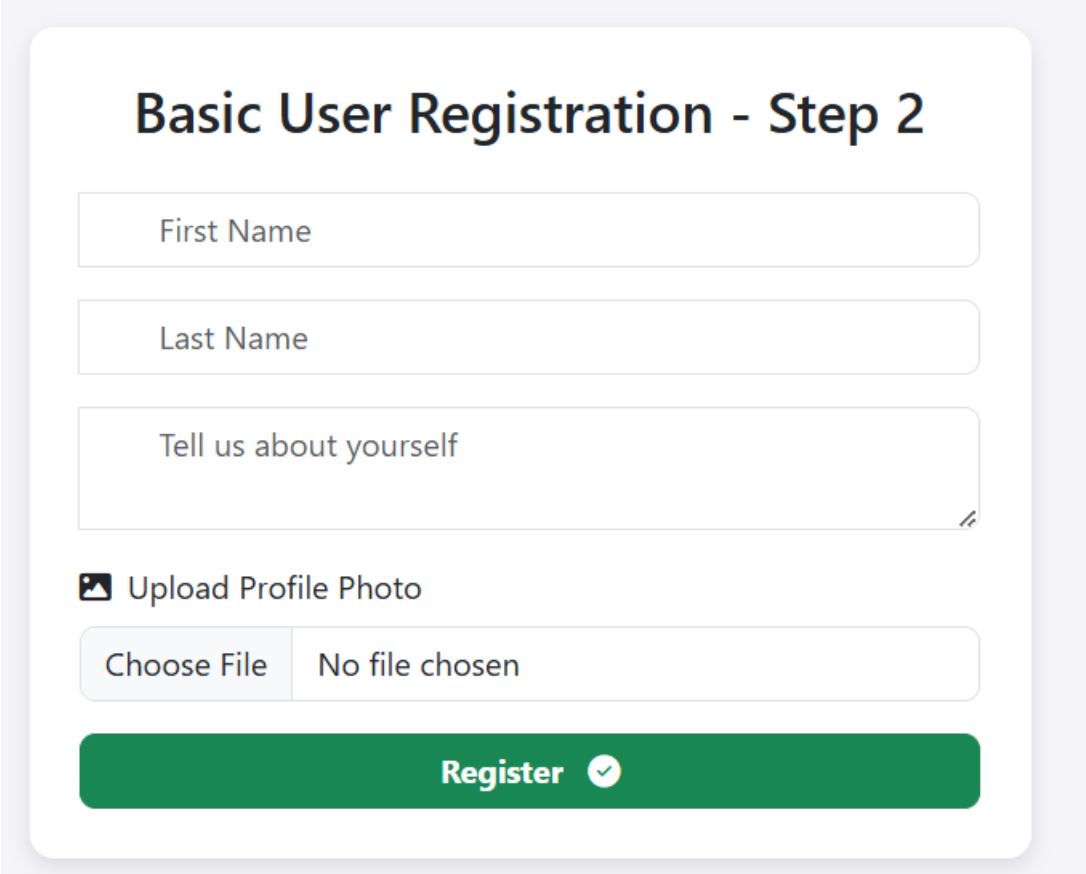



```java
@PostMapping("/step1")  valeriianik *
public String registerBusinessUserStep1(
        @RequestParam String email,
        @RequestParam String password,
        @RequestParam String confirmPassword,
        HttpSession session) {

    if (!password.equals(confirmPassword)) {
        return "redirect:/register/business?error=password_mismatch";
    }

    User user = userService.createUser(email, password, userType: "BUSINESS");
    session.setAttribute( s: "registeredUser", user);

    return "redirect:/register/business/info";
}
```

Step 2: Role-Specific Registration

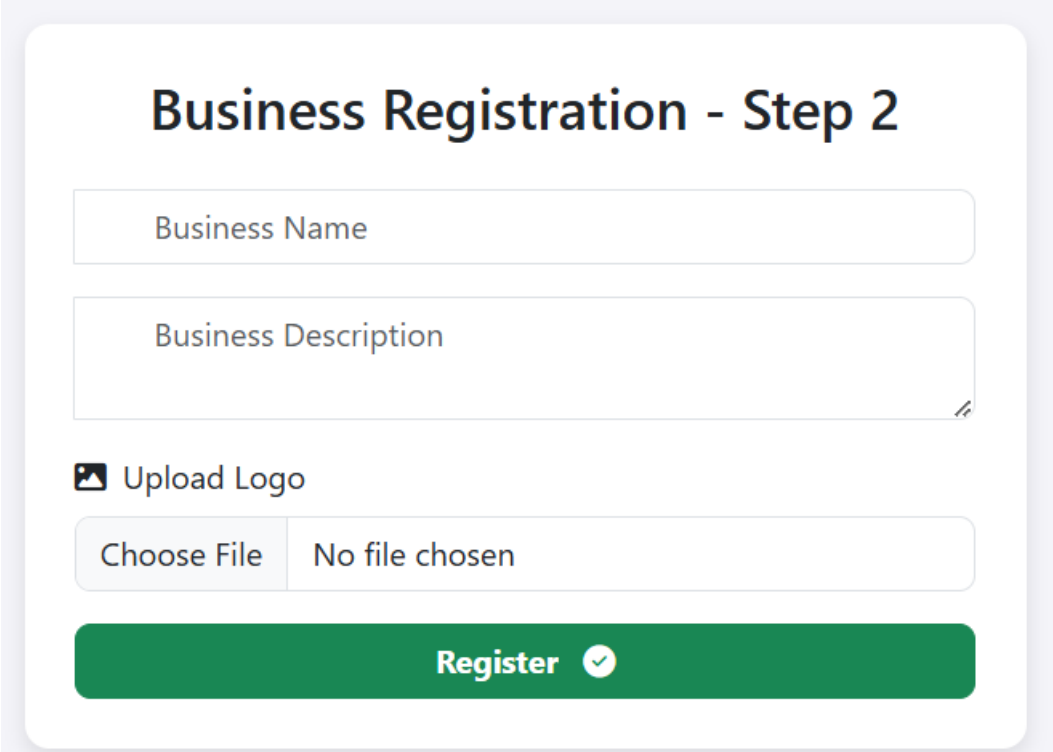
After completing Step 1, users fill in additional details.

- **Basic User** – Provides a name and profile picture.



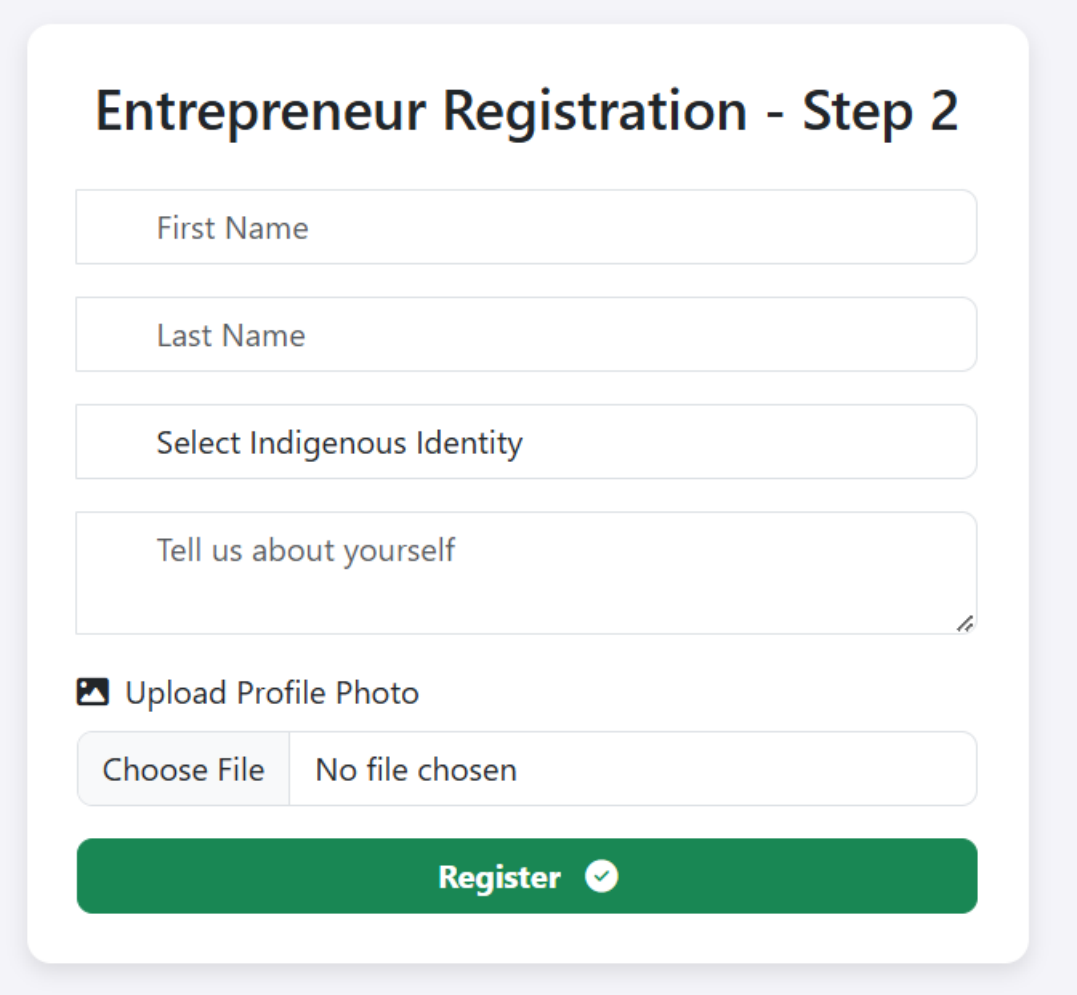

```java
@PostMapping("/step2") valeriianik
public String registerBasicUserStep2(
        @RequestParam String firstName,
        @RequestParam String lastName,
        @RequestParam String about,
        @RequestParam("photo") MultipartFile photo,
        HttpSession session) throws IOException {

    User registeredUser = (User) session.getAttribute( s: "registeredUser");
    if (registeredUser == null) {
        return "redirect:/register/basic";
    }

    BasicUser basicUser = basicUserService.registerBasicUser(registeredUser, firstName, lastName, about, photo);

    System.out.println("✅ Entrepreneur User Registered: " + basicUser);

    session.removeAttribute( s: "registeredUser");
    return "redirect:/login?success=basic_registered";
}
```

- **Business User** – Adds business name, description, and logo.




```java
@PostMapping("/step2")  valeriianik
public String registerBusinessUserStep2(
        @RequestParam String businessName,
        @RequestParam String description,
        @RequestParam("logo") MultipartFile logo,
        HttpSession session) throws IOException {

    User registeredUser = (User) session.getAttribute( s: "registeredUser");

    if (registeredUser == null) {
        System.err.println("🚨 ERROR: No registered user found in session!");
        return "redirect:/register/business";
    }

    System.out.println("✅ User Found in Session: " + registeredUser.getEmail());

    BusinessUser businessUser = businessUserService.registerBusinessUser(registeredUser, businessName, description, logo);

    System.out.println("✅ Business User Registered: " + businessUser);

    session.removeAttribute( s: "registeredUser");
    return "redirect:/login?success=business_registered";
}
```

- **Entrepreneur User** – Provides personal and professional details.




```
@PostMapping("/step2")  valeriianik
public String registerEntrepreneurUserStep2(
        @RequestParam String firstName,
        @RequestParam String lastName,
        @RequestParam String about,
        @RequestParam String identityType,
        @RequestParam("photo") MultipartFile photo,
        HttpSession session) throws IOException {

    User registeredUser = (User) session.getAttribute( s: "registeredUser");
    if (registeredUser == null) {
        return "redirect:/register/entrepreneur";
    }

    EntrepreneurUser entrepreneurUser = entrepreneurUserService.registerEntrepreneurUser(registeredUser, firstName, lastName, about, identityType, photo);

    System.out.println("✅ Entrepreneur User Registered: " + entrepreneurUser);

    session.removeAttribute( s: "registeredUser");
    return "redirect:/login?success=entrepreneur_registered";
}
```

Password Encryption

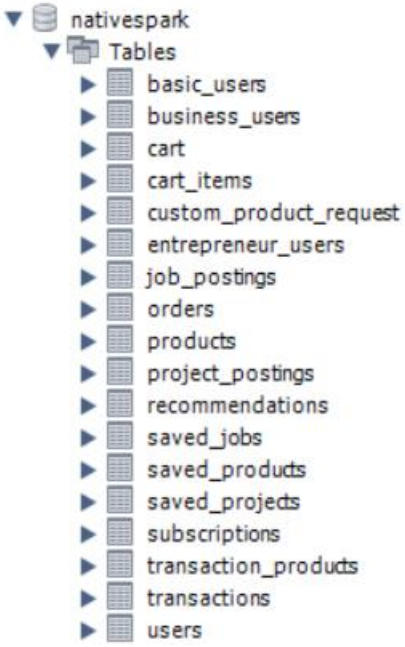
To enhance security, all passwords are hashed using BCrypt prior to being stored in the database. This method prevents the storage of plaintext passwords, thereby reducing the risk of data breaches.

```
public User createUser(String email, String password, String userType) {  3 usages  valeriianik *
    if (userRepository.findByEmail(email).isPresent()) {
        throw new IllegalArgumentException("Email is already in use.");
    }

    User user = new User(email, passwordEncoder.encode(password), userType);
    return userRepository.save(user);
}
```

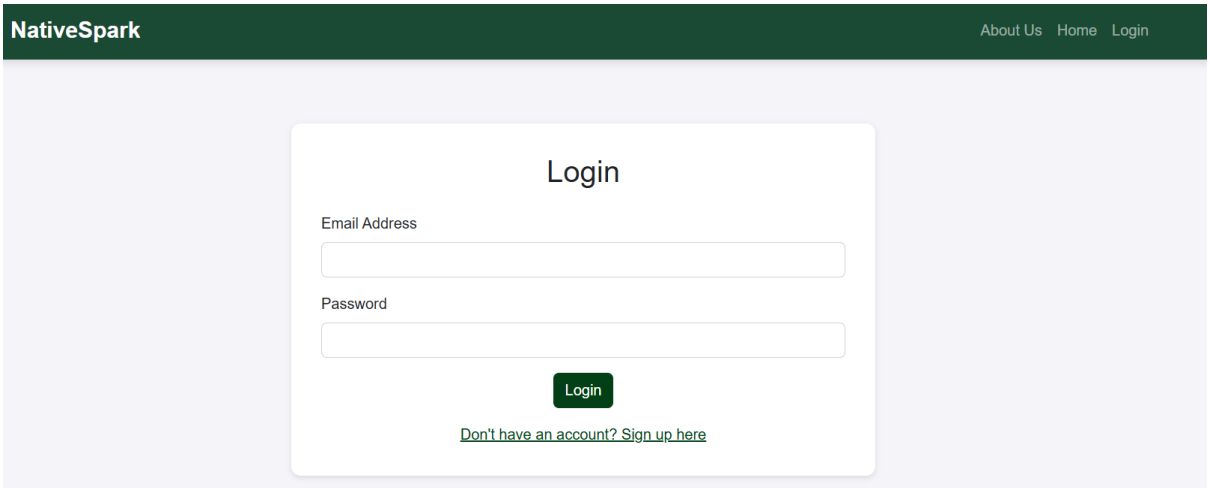
password
$2a$10$pOqLlmvl470Aob.IZ/0jvufhv5CTAaxDZczG.RFU8kd...
$2a$10$8a5QTTRffYtw5lgikl6wvOVxN06FeOmuUzK5u5tKCkQ...
$2a$10$earbZ2goQnN1SAQamLqg6u8OClRA4D6vgfUz2LfSg7h...

Existing Database Schema

The platform follows database normalization (3NF), ensuring that user data is efficiently stored without redundancy.



Login Handling




```java
@PostMapping("/login")  new *
public String loginUser(@RequestParam("username") String email,
                        @RequestParam("password") String password,
                        Model model) {

    System.out.println("🔄 Attempting login for email: " + email);

    Optional<User> userOptional = userService.findByEmail(email);
    if (userOptional.isEmpty()) {
        System.out.println("❌ User not found: " + email);
        model.addAttribute( attributeName: "error",  attributeValue: "Invalid email or password.");
        return "login";
    }

    User user = userOptional.get();
    if (!userService.verifyPassword(password, user.getPassword())) {
        System.out.println("❌ Incorrect password for: " + email);
        model.addAttribute( attributeName: "error",  attributeValue: "Invalid email or password.");
        return "login";
    }

    System.out.println("✅ Authentication successful. Redirecting to account...");
    return "redirect:/account";
}
```

Session-Based Authentication

- **If a user is logged in**, they can browse restricted pages.
- **If not logged in**, they are redirected to the login page.

```
@GetMapping  new *
public String showAccountPage(Model model) {
    org.springframework.security.core.Authentication authentication =
            SecurityContextHolder.getContext().getAuthentication();

    if (authentication == null || !authentication.isAuthenticated()) {
        System.out.println("❌ No authenticated user found, redirecting to login.");
        return "redirect:/login";
    }

    String email = authentication.getName();
    Optional<User> userOptional = userRepository.findByEmail(email);

    if (userOptional.isEmpty()) {
        System.out.println("❌ User not found in database, redirecting to login.");
        return "redirect:/login";
    }

    User user = userOptional.get();
    model.addAttribute( attributeName: "user", user);
    System.out.println("✅ User session loaded successfully: " + user.getEmail());

    return "account";
}
```

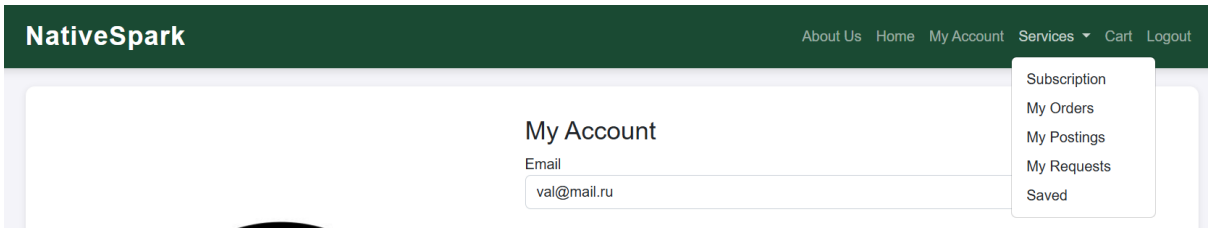
Welcome page

Dynamic Navbar

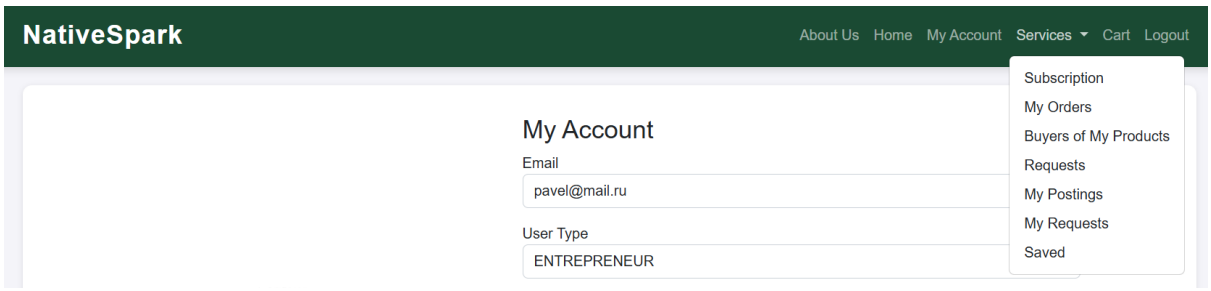
If the user is not logged in, the "My Account" option is hidden in the navigation bar.

When the user is logged in more options are available:

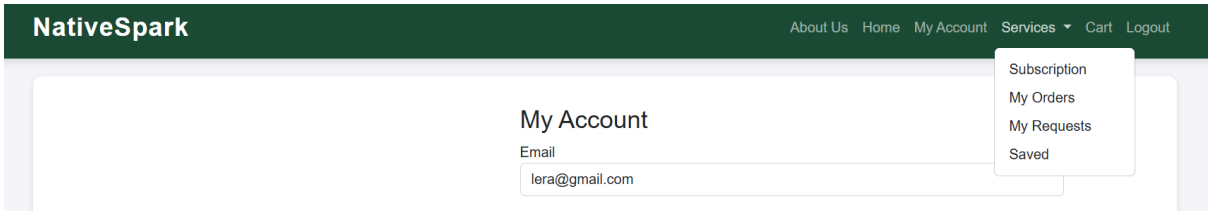
- Business



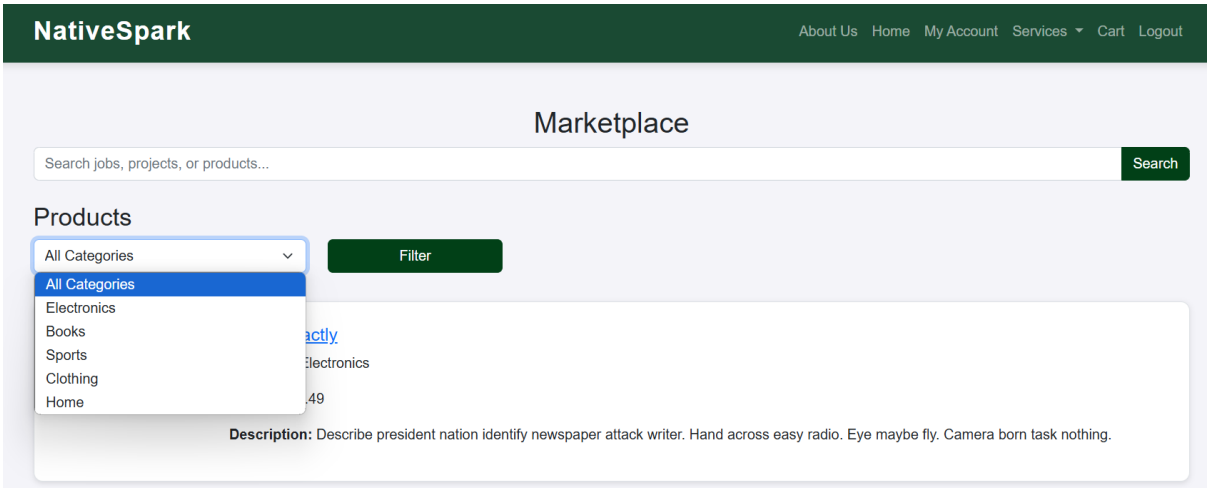
- Entrepreneur



- Basic

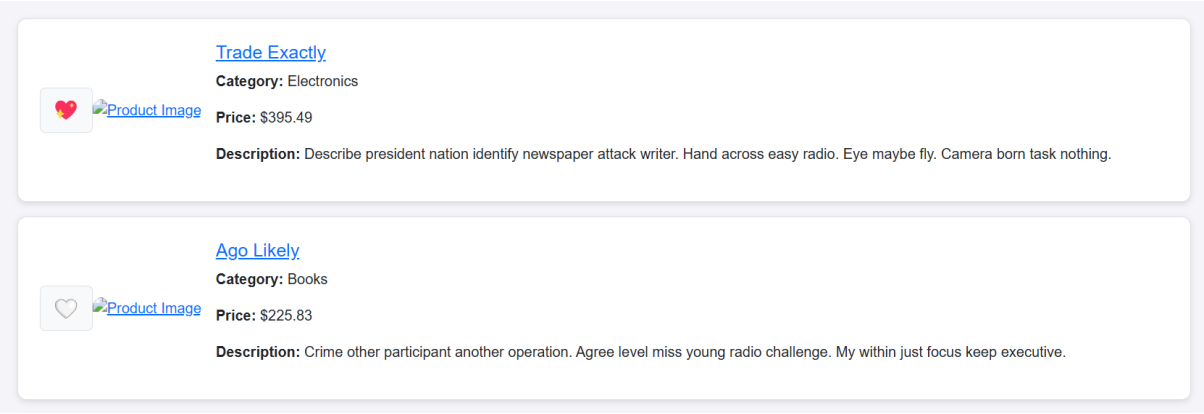


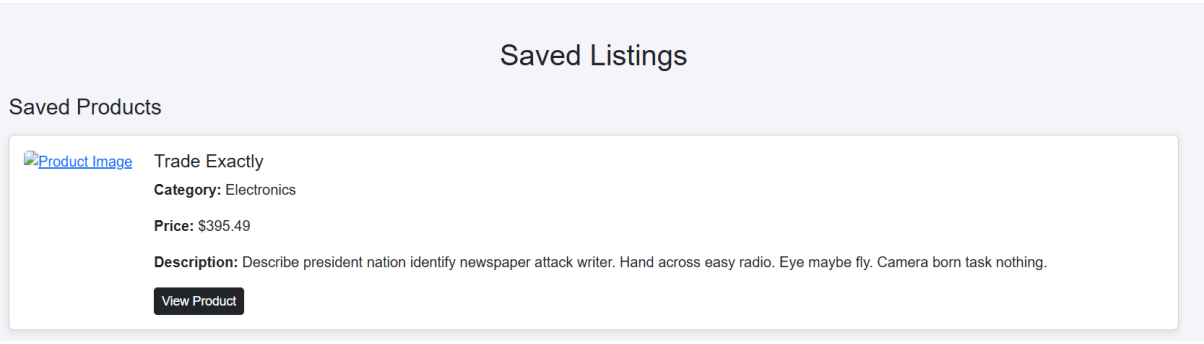
Filter and search on Home page



The user can filter products by the relevant category. Additionally, the user can search for the desired product in the search bar using keywords that will match the product name and description.

Save favourites

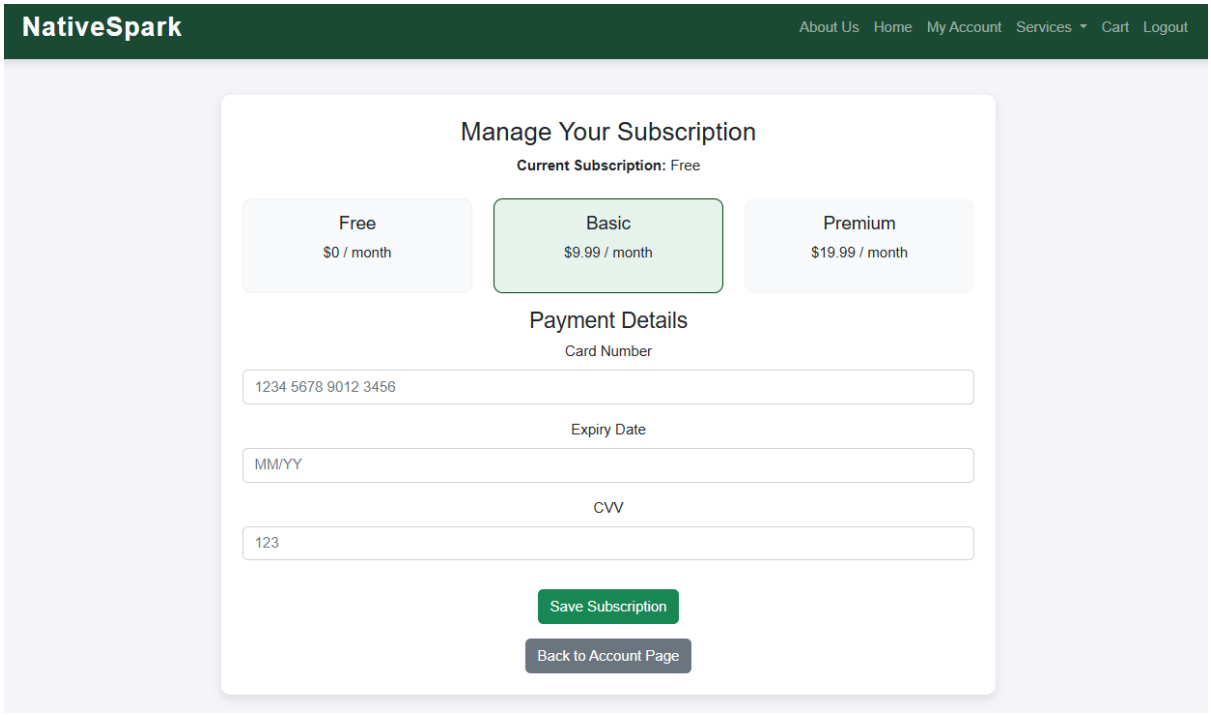




```java
@PostMapping("/product/{id}/toggle-save")  valeriianik
@ResponseBody
public ResponseEntity<String> toggleSave(@PathVariable Long id, Authentication auth) {
    if (auth == null || !auth.isAuthenticated()) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Not logged in");
    }

    String email = auth.getName();
    User user = userRepo.findByEmail(email).orElse( other: null);
    if (user == null) return ResponseEntity.badRequest().body("User not found");

    Product product = productRepo.findById(id).orElse( other: null);
    if (product == null) return ResponseEntity.notFound().build();

    Set<User> savedUsers = product.getSavedByUsers();
    String responseStatus;
    if (savedUsers.contains(user)) {
        savedUsers.remove(user);
        responseStatus = "unsaved";
    } else {
        savedUsers.add(user);
        responseStatus = "saved";
    }

    productRepo.save(product);
    return ResponseEntity.ok(responseStatus);
}
```

Subscription

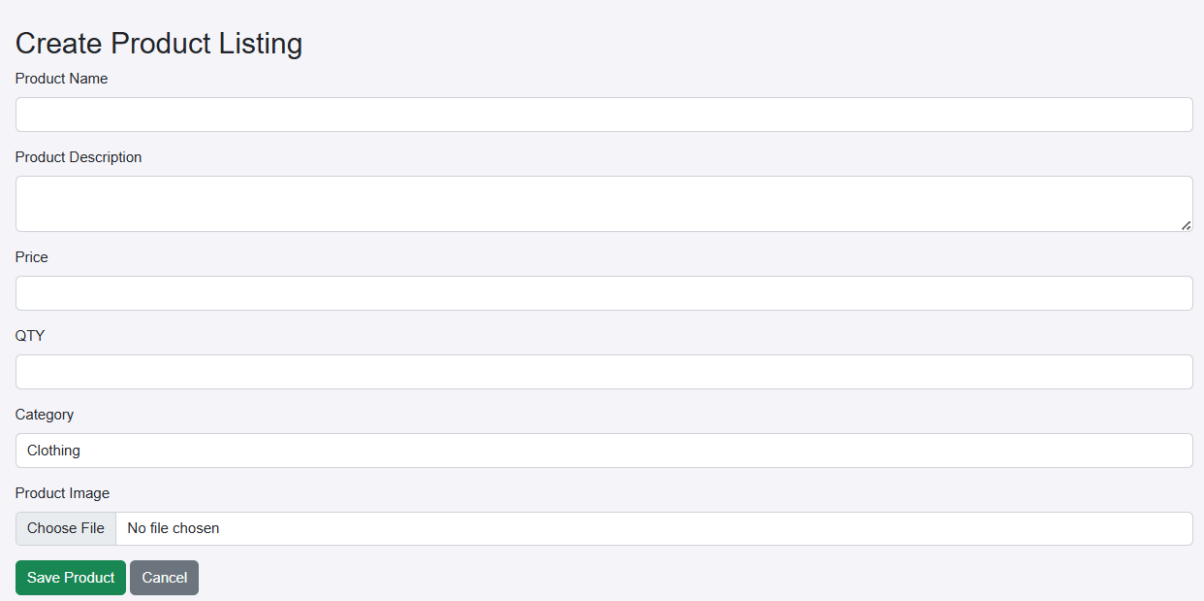
Wen the user creates the account the functionality assigns him a free subscription, later the user can change it.




```
// Determine cost based on subscription type
double cost = 0.0;
if ("Basic".equalsIgnoreCase(subscriptionType)) {
    cost = 9.99;
} else if ("Premium".equalsIgnoreCase(subscriptionType)) {
    cost = 19.99;
}

if (subscriptionOptional.isPresent()) {
    Subscription subscription = subscriptionOptional.get();
    logger.info("Updating subscription for user: " + user.getEmail() + " to " + subscriptionType);
    subscription.setSubscriptionType(subscriptionType);
    subscriptionRepository.save(subscription);
} else {
    logger.info("Creating new subscription for user: " + user.getEmail());
    Subscription newSubscription = new Subscription(user, subscriptionType);
    subscriptionRepository.save(newSubscription);
}
```

Product functionality (add, delete and update)

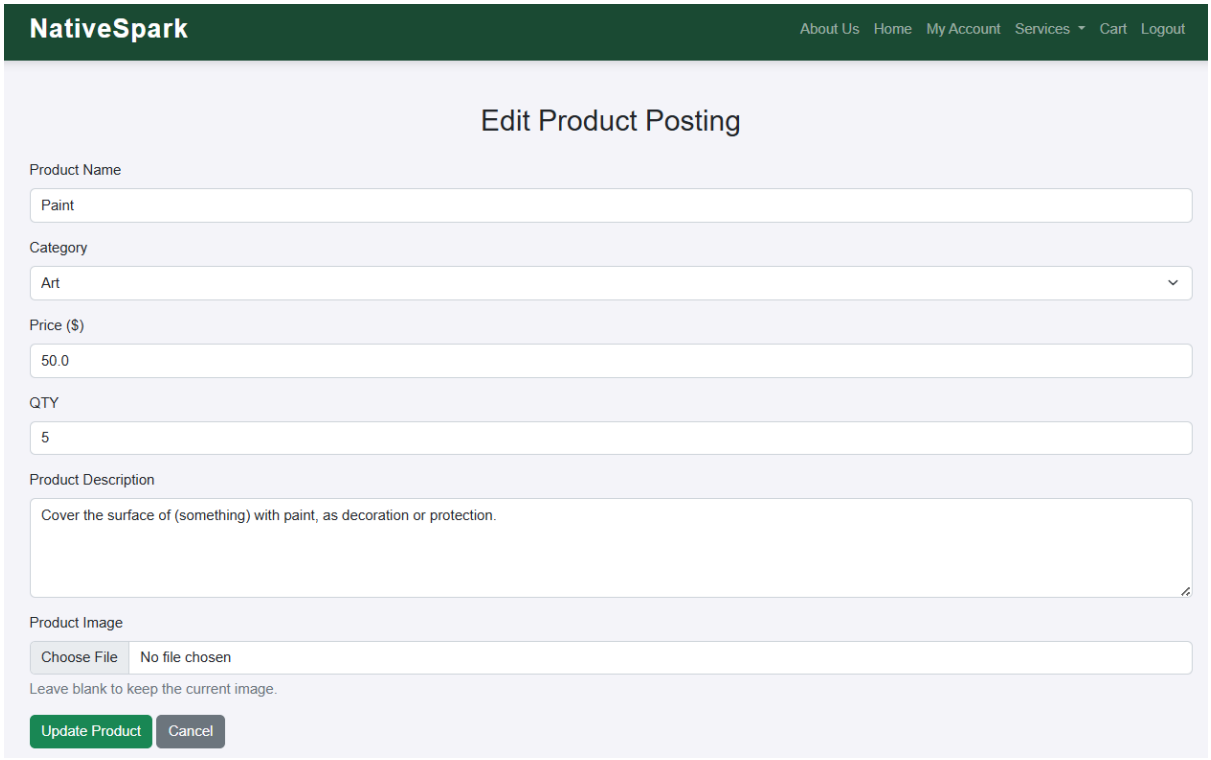
Add product




```java
@PostMapping("/save")  valeriianik
public String saveProductPosting(@ModelAttribute Product product,
                                 @RequestParam("imageFile") MultipartFile imageFile,
                                 Principal principal) throws IOException {
    User user = getAuthenticatedUser(principal);
    if (user == null) return "redirect:/login";

    Optional<EntrepreneurUser> entrepreneurUser = entrepreneurUserRepository.findByUser(user);
    if (entrepreneurUser.isPresent()) {
        product.setEntrepreneur(entrepreneurUser.get());
        product.setPostedDate(LocalDateTime.now());

        // ✅ Handle Image Upload
        if (!imageFile.isEmpty()) {
            File directory = new File(UPLOAD_DIR);
            if (!directory.exists()) {
                boolean created = directory.mkdirs();
                if (!created) {
                    throw new IOException("Could not create upload directory: " + UPLOAD_DIR);
                }
            }
```

```
            // ✅ Generate unique filename
            String fileName = System.currentTimeMillis() + "_" + imageFile.getOriginalFilename();
            Path filePath = Paths.get(UPLOAD_DIR, fileName);

            Files.copy(imageFile.getInputStream(), filePath);

            // ✅ Save relative image path for Thymeleaf
            product.setImagePath("/" + UPLOAD_DIR + fileName);
        }

        productRepository.save(product);
    }

    return "redirect:/my_postings"; // ✅ Redirect back to My Postings page
}
```

Update Product

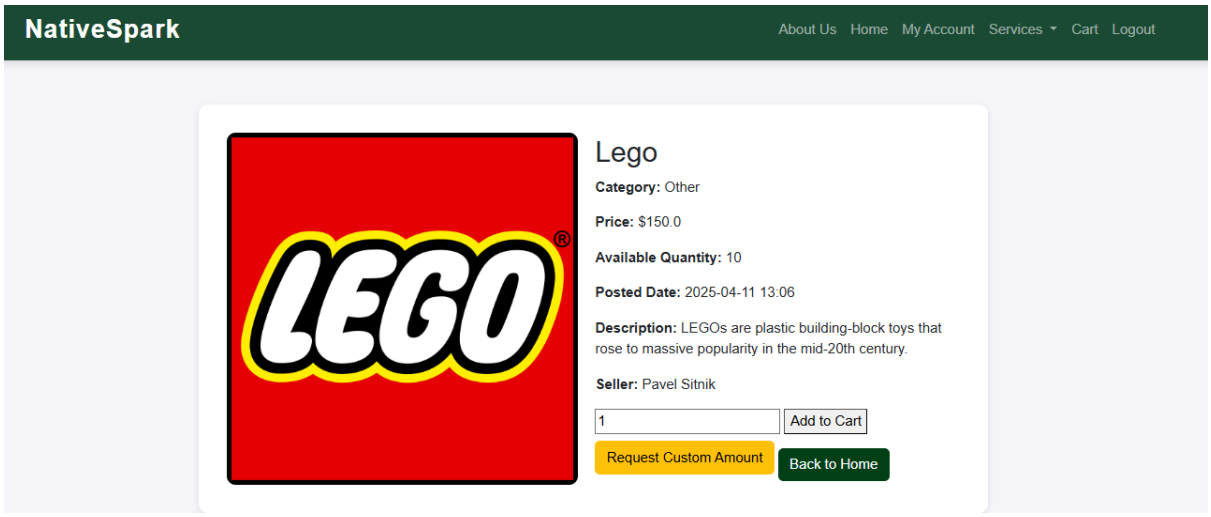


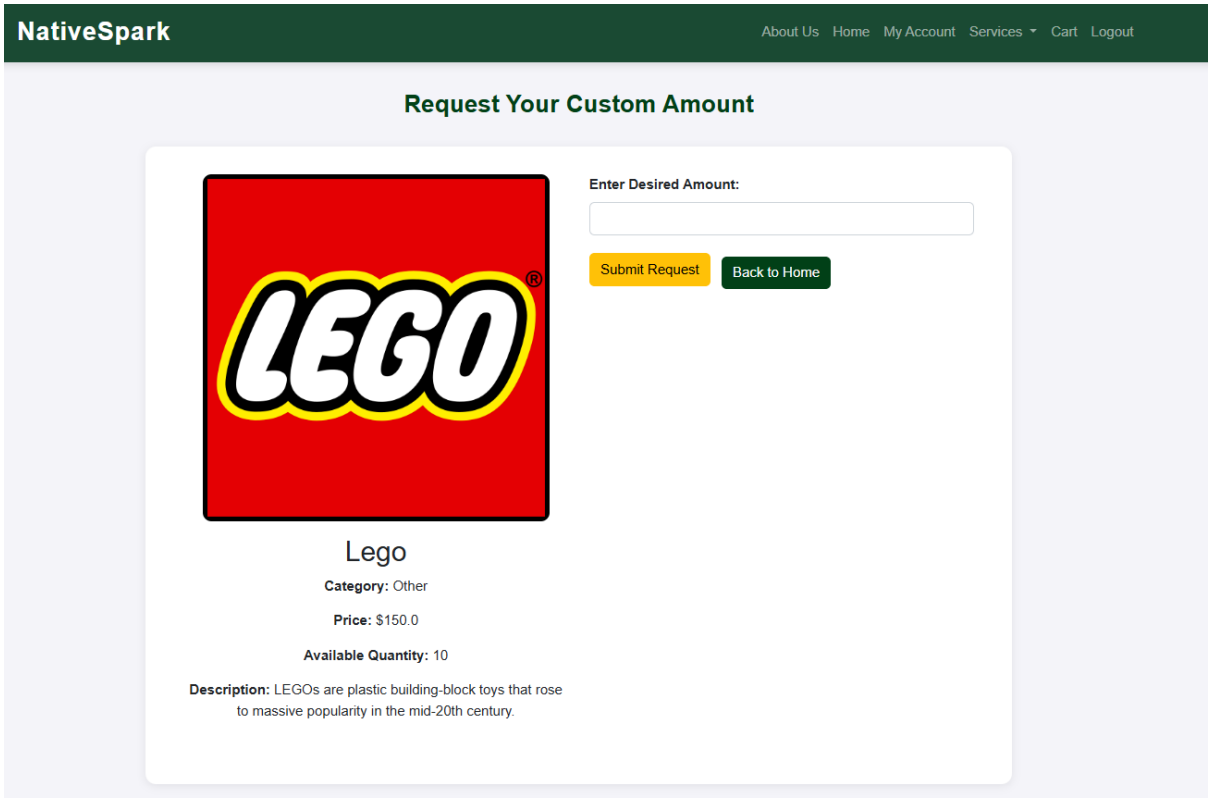
```
@PostMapping("/update/{id}")  valeriianik
public String updateProductPosting(@PathVariable Long id, @ModelAttribute Product product,
                                   @RequestParam(value = "imageFile", required = false) MultipartFile imageFile) throws I
    Optional<Product> existingProduct = productRepository.findById(id);

    if (existingProduct.isPresent()) {
        Product updatedProduct = existingProduct.get();
        updatedProduct.setProductName(product.getProductName());
        updatedProduct.setCategory(product.getCategory());
        updatedProduct.setPrice(product.getPrice());
        updatedProduct.setProductDescription(product.getProductDescription());
        updatedProduct.setQuantity(product.getQuantity());

        // ✅ Handle Image Upload
        if (imageFile != null && !imageFile.isEmpty()) {
            String fileName = System.currentTimeMillis() + "_" + imageFile.getOriginalFilename();
            Path filePath = Paths.get( first: "uploads/products/", fileName);
            Files.copy(imageFile.getInputStream(), filePath);
            updatedProduct.setImagePath("/uploads/products/" + fileName);
        }

        productRepository.save(updatedProduct);
    }

    return "redirect:/my_postings";
}
```

Delete Product

```
@PostMapping("/delete/{id}")  valeriianik
public String deleteProductPosting(@PathVariable Long id) {
    productRepository.deleteById(id);
    return "redirect:/my_postings";
}
```

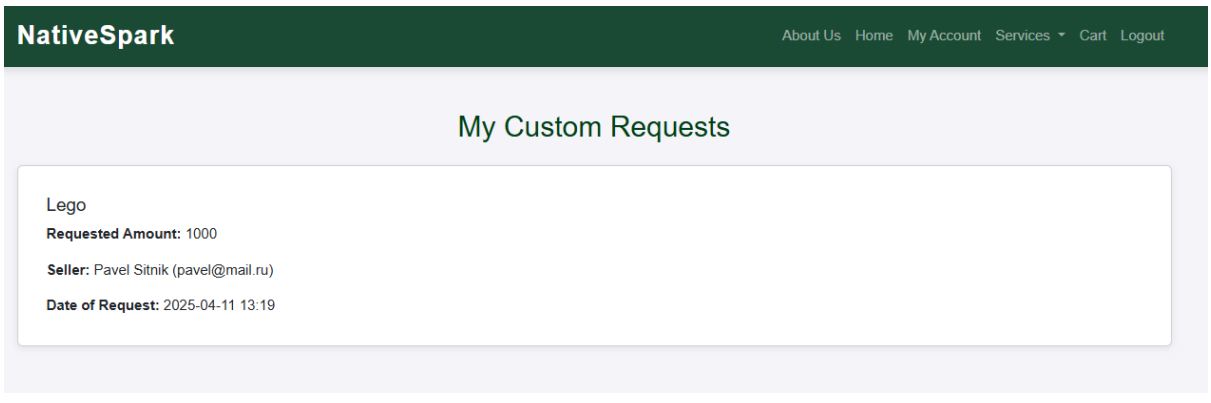
Request custom amount





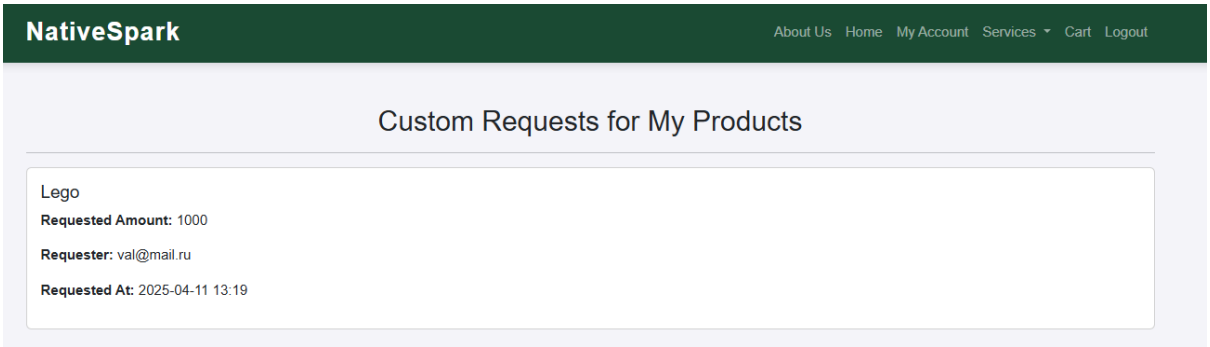

```
@PostMapping("/{id}")  valeriianik
public String submitCustomRequest(@PathVariable Long id,
                                  @ModelAttribute("request") CustomProductRequest request,
                                  Authentication auth) {
    Product product = productRepository.findById(id).orElseThrow();
    User user = userRepository.findByEmail(auth.getName()).orElseThrow();

    request.setProduct(product);
    request.setUser(user);
    customProductRequestRepository.save(request);

    return "redirect:/request-custom/" + id + "?success=true";
}
```

Custom requests are shown in account of the user.



```java
@GetMapping("/my_requests")  valeriianik
public String viewMyCustomRequests(Model model, Authentication authentication) {
    User user = userRepository.findByEmail(authentication.getName()).orElseThrow();
    List<CustomProductRequest> customRequests = customProductRequestRepository.findByUser(user);
    model.addAttribute( attributeName: "customRequests", customRequests);
    return "my_requests";
}
```

The user will also see the request to make it



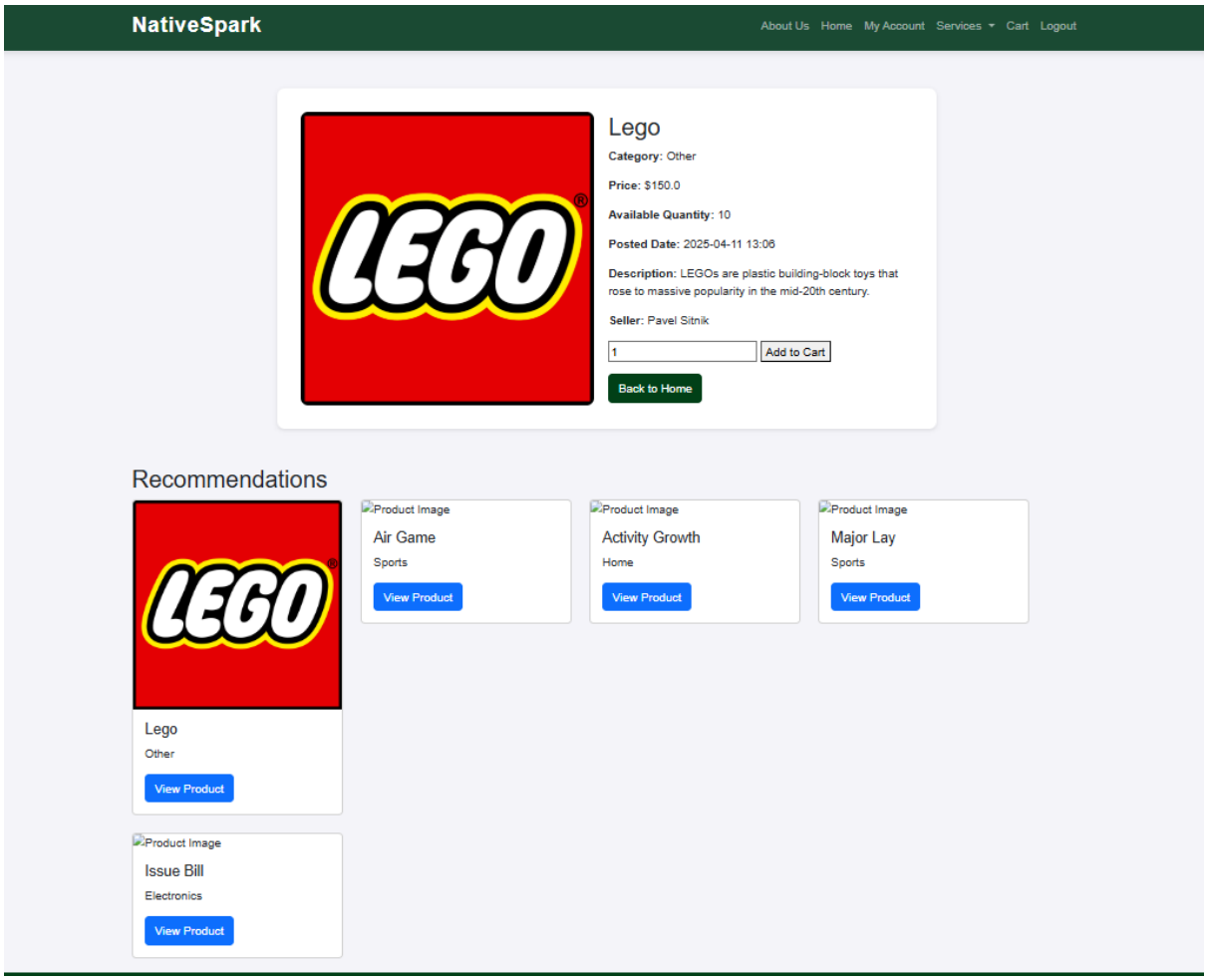

```java
@GetMapping("/requests")  valeriianik
@PreAuthorize("hasAuthority('ROLE_ENTREPRENEUR')")
public String viewRequestsForMyProducts(Model model, Authentication authentication) {
    User entrepreneurUser = userRepository.findByEmail(authentication.getName()).orElseThrow();
    EntrepreneurUser entrepreneur = entrepreneurUserRepository.findByUser(entrepreneurUser).orElseThrow();

    List<CustomProductRequest> allRequests = customProductRequestRepository.findAllByProduct_Entrepreneur(entrepreneur);
    model.addAttribute( attributeName: "requests", allRequests);

    return "requests"; // ✅ This must match the name of your Thymeleaf template (requests.html)
}

}
```

Python-based recommendation of products

The program shows the recommendations of the product based on the category and the price for the product.

SQLAlchemy: This Python library is used for Object-Relational Mapping (ORM) access to MySQL database tables and models.

scikit-learn: This is Python's machine learning library, which is employed to implement the product recommendation engine based on similarity metrics.



Script for recommendations

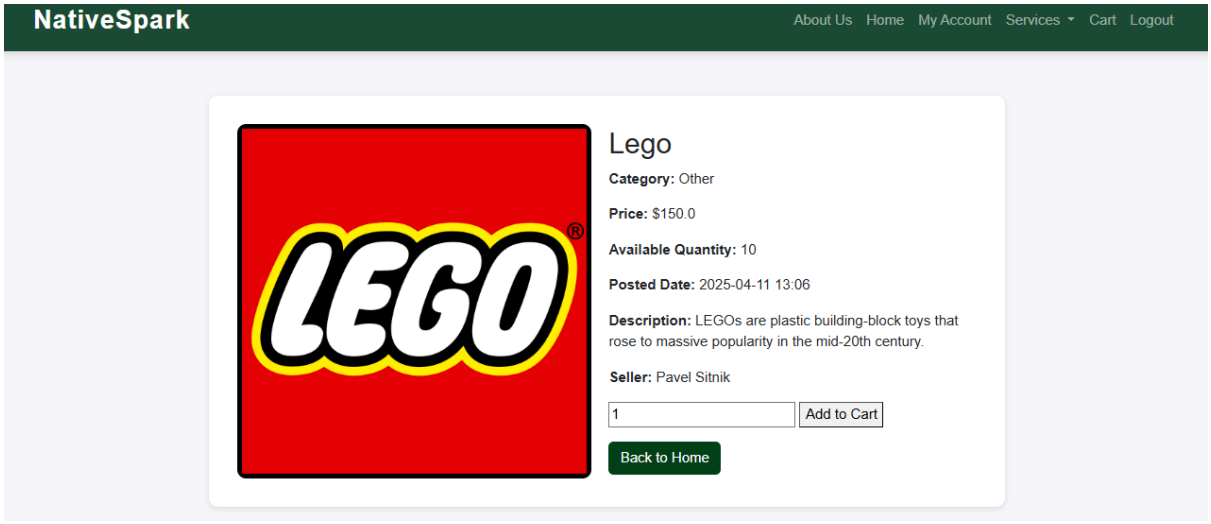
```
import pandas as pd
import numpy as np
from db.alchemy import (
    ProductRecommendation,
    session,
    products_table,
    mysqlengine,
    reccomended_products_table,
```

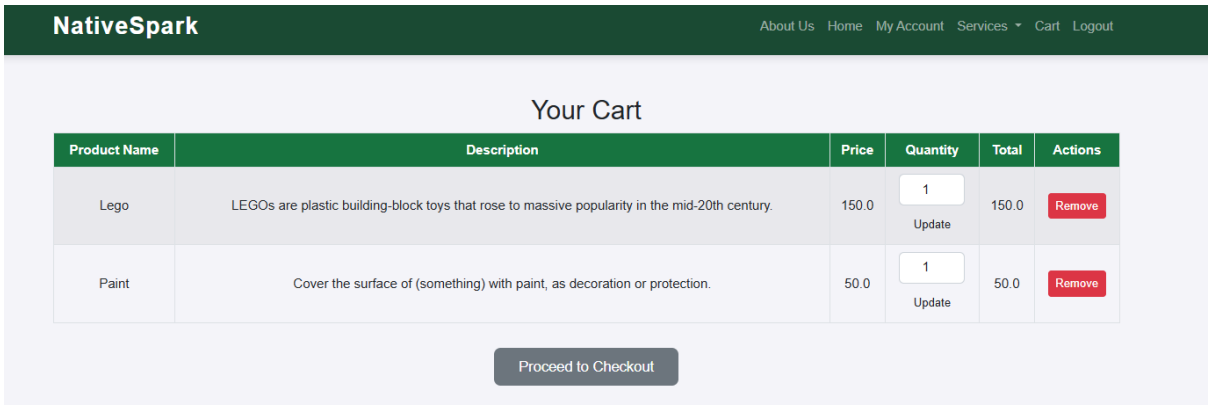
```
)
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import MinMaxScaler


# Load products
def load_products_df():
    with mysqlengine.connect() as connection:
        query = products_table.select()
        df = pd.read_sql(query, connection)
        return df


def get_recommendations_for_product(products_df, similarity_matrix, product_id, n=2):
    product_idx = products_df.index[products_df["id"] == product_id].tolist()[0]

    similarity_scores = similarity_matrix[product_idx]

    similar_indices = similarity_scores.argsort()[::-1][1 : n + 1]

    recommendations = products_df.iloc[similar_indices][["id", "category", "price"]]
    return recommendations


def create_reccomendations(products_df):
    category_encoded = pd.get_dummies(products_df["category"])

    scaler = MinMaxScaler()
    price_normalized = scaler.fit_transform(products_df[["price"]])

    feature_matrix = np.hstack((category_encoded.values, price_normalized))
    similarity_matrix = cosine_similarity(feature_matrix)

    insert_data = []
    for product_idx, product_id in enumerate(products_df["id"]):
        similarity_scores = similarity_matrix[product_idx]
        similar_indices = similarity_scores.argsort()[::-1][1:6]
        for idx in similar_indices:
            insert_data.append(
                {
                    "product_id": product_id,
                    "recommended_product_id": int(products_df.iloc[idx]["id"]),
                }
            )
    with mysqlengine.connect() as connection:
        connection.execute(reccomended_products_table.insert(), insert_data)
```

```
        connection.commit()


def clear_reccomendations():
    session.query(ProductRecommendation).delete()
    session.commit()


def main():
    clear_reccomendations()
    products_df = load_products_df()
    create_reccomendations(products_df)


if __name__ == "__main__":
    main()
```

Add to cart

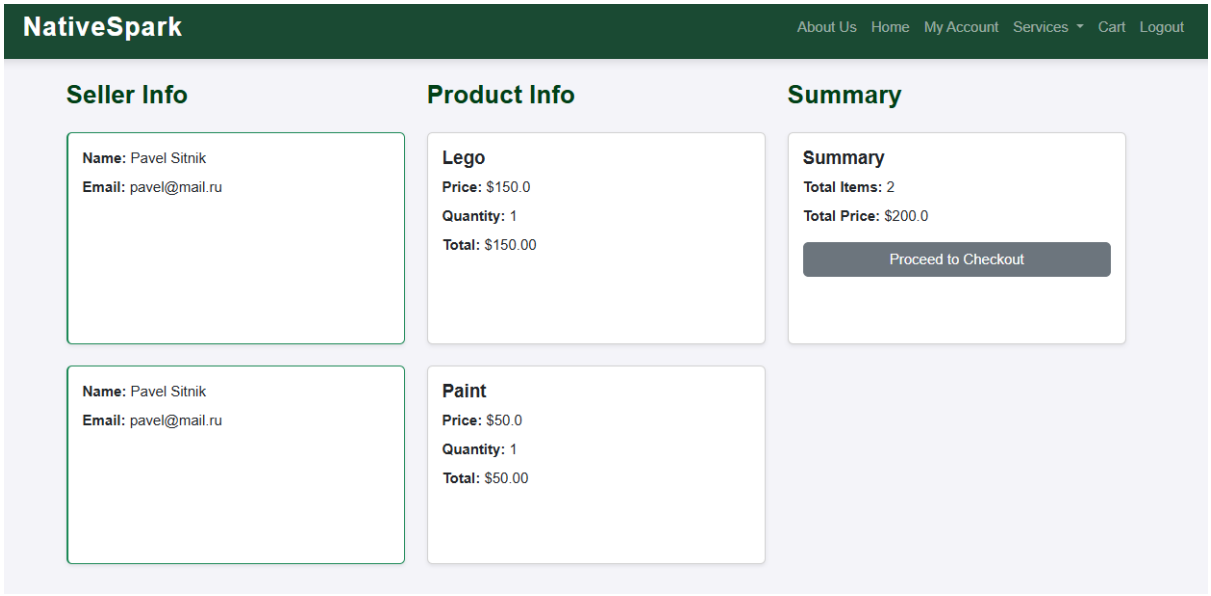
The user can add the product to cart and update the qty or delete the product from cart.

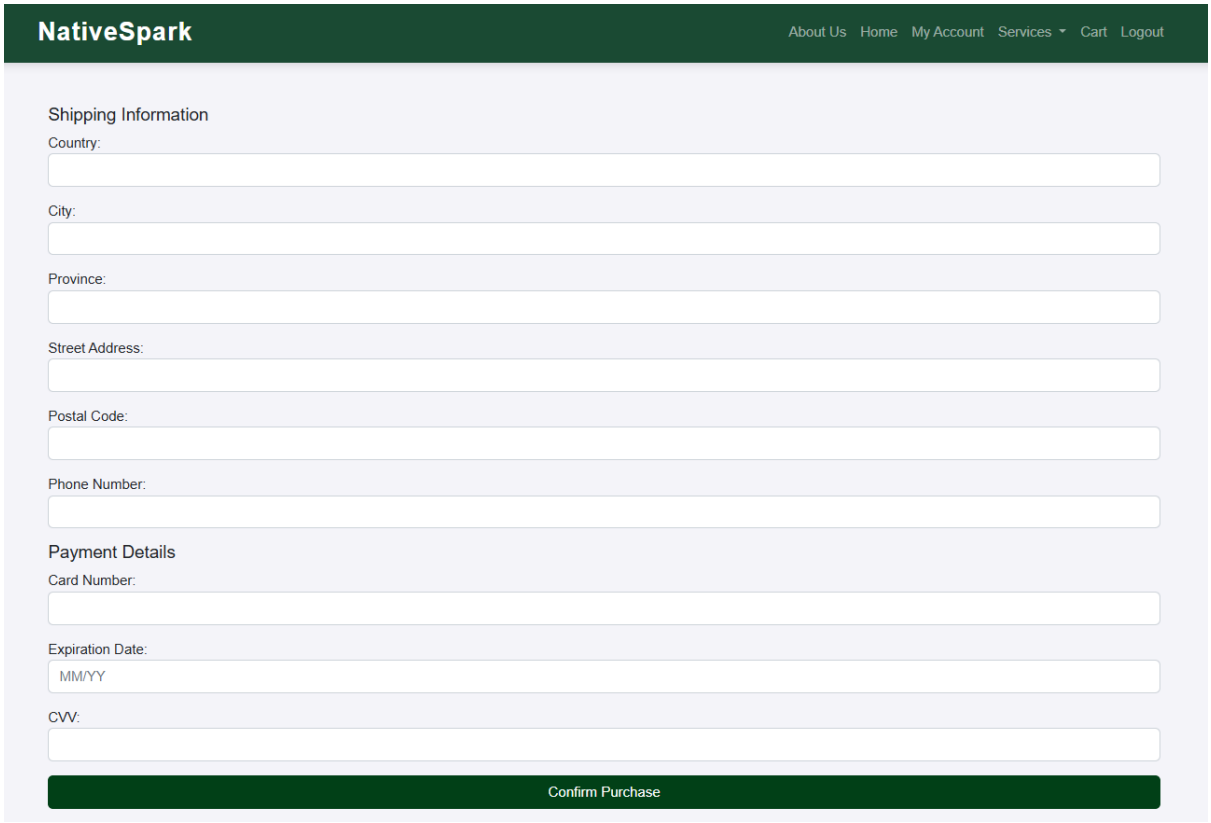




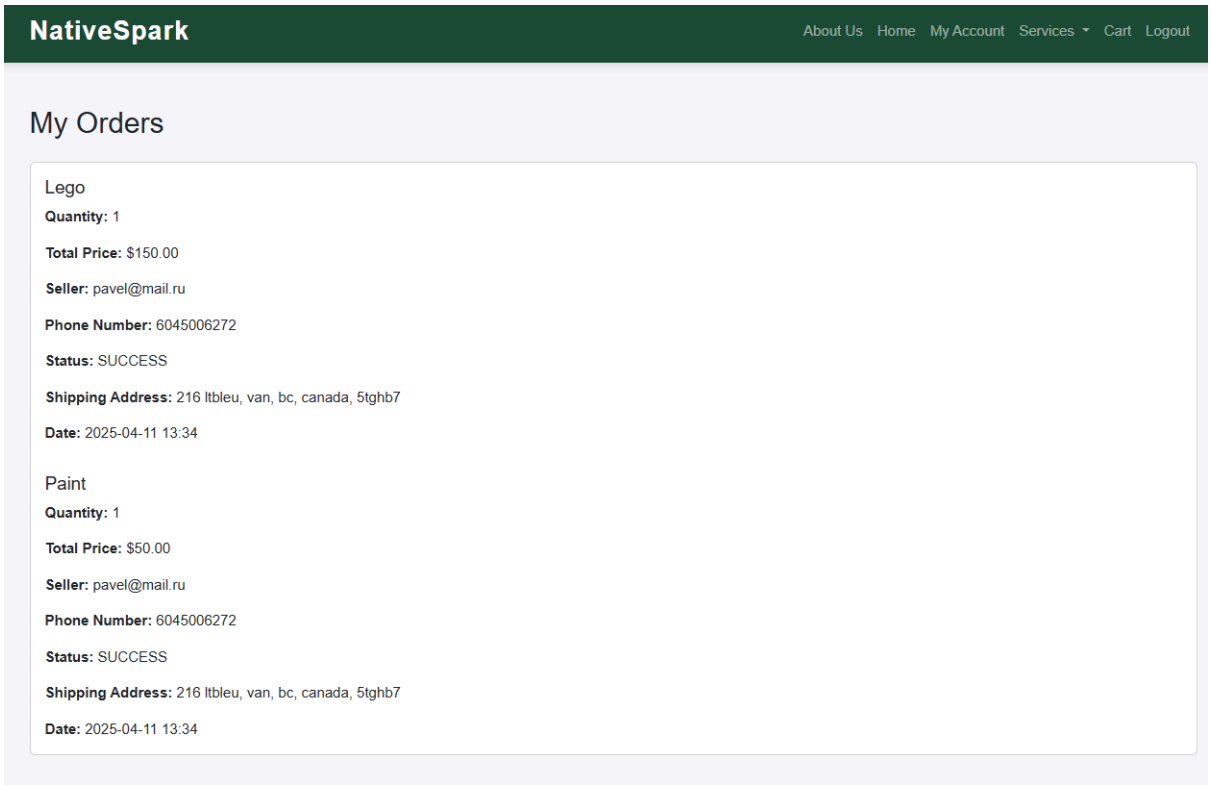
```java
@PostMapping("/cart/updateQuantity")  valeriianik
public String updateQuantity(@RequestParam Long cartItemId, @RequestParam int quantity) {
    CartItem cartItem = cartItemRepository.findById(cartItemId)
            .orElseThrow(() -> new RuntimeException("CartItem not found"));
    cartItem.setQuantity(quantity);
    cartItemRepository.save(cartItem);
    return "redirect:/cart";
}

@PostMapping("/cart/removeItem")  valeriianik
public String removeItem(@RequestParam Long cartItemId) {
    CartItem cartItem = cartItemRepository.findById(cartItemId)
            .orElseThrow(() -> new RuntimeException("CartItem not found"));
    cartItemRepository.delete(cartItem);
    return "redirect:/cart";
}
```

Checkout Process



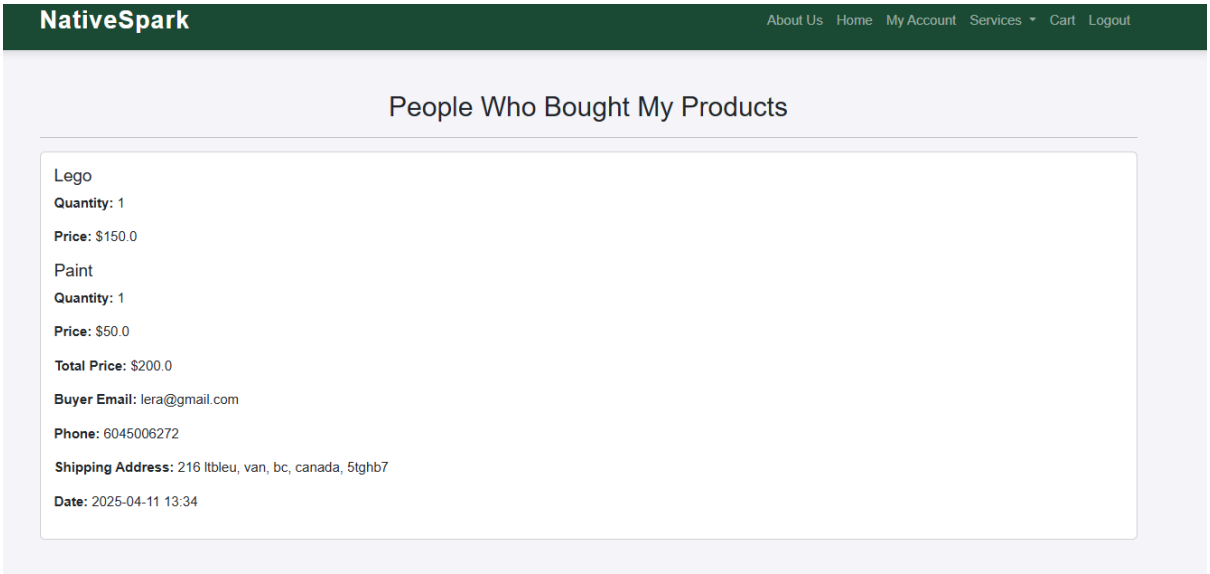


My orders summary



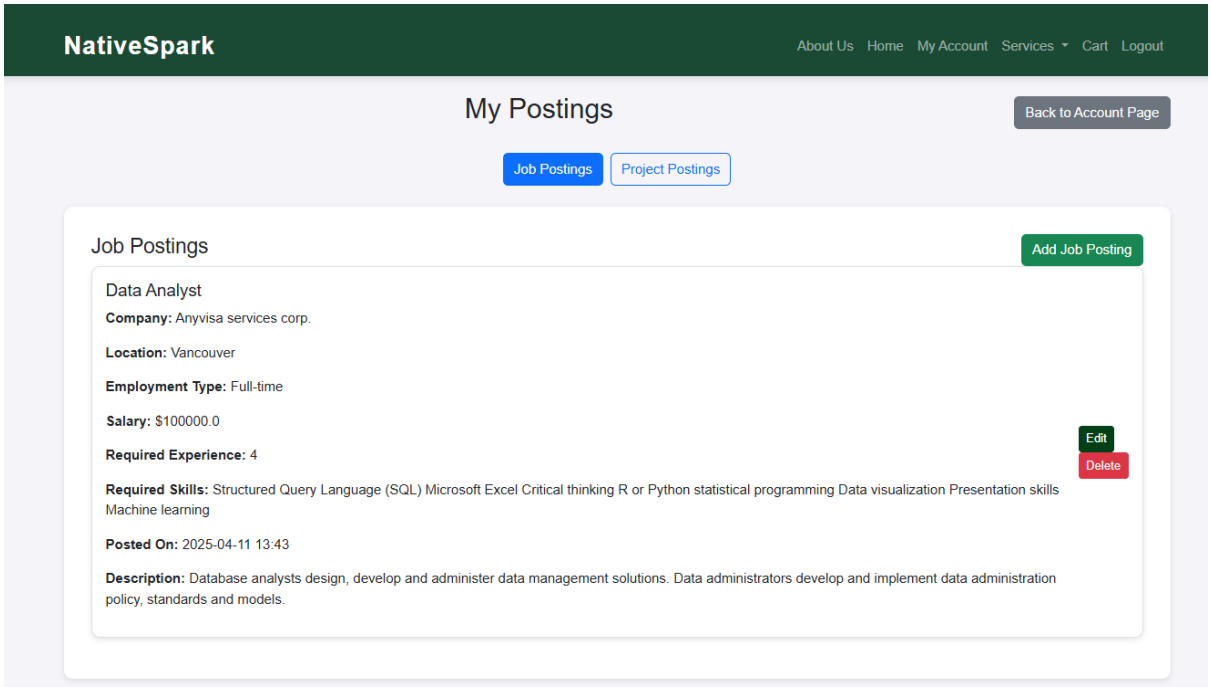
```java
@GetMapping("/my_orders")  valeriianik
public String viewOrders(Model model, Authentication authentication) {
    String email = authentication.getName();
    Optional<User> userOptional = userRepository.findByEmail(email);

    if (userOptional.isPresent()) {
        User user = userOptional.get();
        List<Transaction> orders = transactionRepository.findByBuyer(user);
        model.addAttribute( attributeName: "orders", orders);
    }

    return "my_orders";
}
```

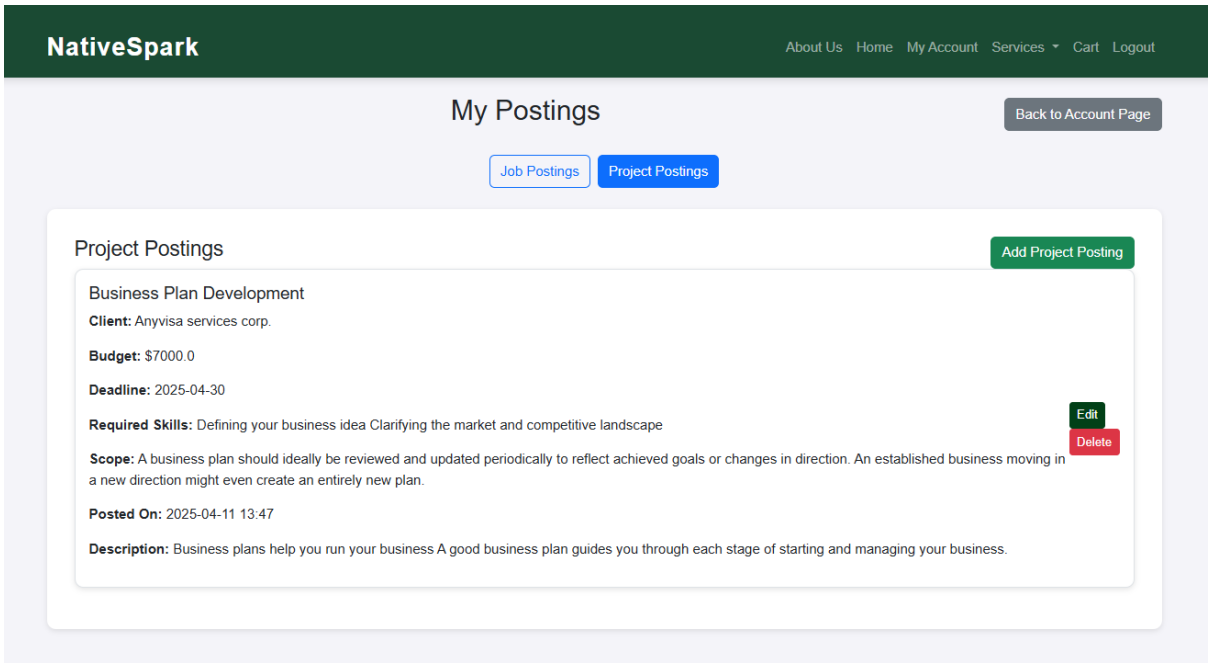
Buyers of the product




```java
@GetMapping("/buyers")  valeriianik
public String viewBuyersOfMyProducts(Model model, Authentication authentication) {
    String email = authentication.getName();
    User user = userRepository.findByEmail(email).orElse( other: null);

    if (user == null) {
        return "redirect:/login";
    }

    List<Transaction> sales = transactionRepository.findBySeller(user);  // ✅ correct lookup
    model.addAttribute( attributeName: "sales", sales);

    return "buyers";
}

}
```

Job and Project functionality (add, delete and update)





Evaluation Techniques

The mixed-method evaluation was made to identify strengths in the current implementation and uncover areas for future enhancement, ensuring that the platform aligns closely with the expectations and needs of its target audience.

Basic user

Also, besides the survey the discussion was made afterwards, I was asked to help with the development of other platform (due to my knowledge of how to make the platform work).

1.How easy was it to navigate the platform and find what you were looking for?

2. Did the product listings (images, descriptions, pricing) provide enough information for you to make a decision?

3. How likely are you to purchase Indigenous-made products through NativeSpark?

4. What features do you find most valuable as a basic user?

5. If a product is out of stock, how useful is the 'Request Custom Amount' feature for you?

6. What additional features would make you more likely to use this platform regularly?

7. How comfortable were you with the design and layout of the site?

8. Did you experience any issues while browsing, searching, or saving items?

9. Based on what you've seen and the idea presented, how likely are you to recommend NativeSpark to others?

Most users reported that the platform is easy to navigate and visually appealing, demonstrating success in creating an intuitive user interface. Key features such as product search, filters, the "Save" (heart) button, and the "Request Custom Amount" function were particularly highlighted as useful, contributing to a positive user experience. Additionally, the personalized recommendation engine was appreciated for helping users discover relevant products.

However, aside from satisfaction with current functionalities, users expressed a strong desire for more advanced tools to enhance decision-making and shopping confidence. Suggestions included implementing a user-driven review and rating system to evaluate products and sellers based on community feedback, as well as expanding the filtering system to allow for more precise sorting by item characteristics (e.g., size, color, price range, location). This feedback

reflects a growing expectation for customization and trust-building features, which are becoming standard in modern e-commerce platforms.

Business user

1. Is there anything missing from NativeSpark that would make your work easier?

2. If you could add one feature to the platform, what would it be?

3. What do you wish was different or better about the current interface or process?

4. How easy is it to post your jobs or projects on the platform?

5. Is the “Save” feature (heart icon) useful for you? Why or why not?

6. How do you feel about the "My Orders" page - is it giving you enough detail?

Overall, user appreciated the platform's functionality and user interface, finding it generally intuitive and effective for posting jobs or projects. While the onboarding experience for returning users was considered smooth, there were suggestions for enhancing the onboarding process for first-time users. Suggestions included adding templates or guided prompts during job or project creation.

A consistent theme in the feedback was the need for improved communication and management tools. Business users found the "Save" (heart) feature useful for tracking profiles, but expressed interest in expanding its capabilities with features such as tagging or folder creation. Additionally, they recommended integrating a live chat option to facilitate real-time communication with Indigenous creators or support staff, which would enhance responsiveness and reduce friction in transactions.

Feedback regarding the “My Orders” page indicated a desire for more detailed information and tracking. Business users requested clearer timelines, a visual progress tracker, and the ability to leave private notes on each transaction to enhance internal project management. Another important feature requested was a way to view and manage candidates who expressed interest in job or project postings, which would streamline hiring and collaboration efforts.

In conclusion, while business user was satisfied with the platform's current offerings, their suggestions highlight the need for more robust operational tools and process automation. Prioritizing these enhancements in future development phases will not only improve the efficiency of business users but also contribute to a more dynamic and scalable marketplace environment on NativeSpark.

Reflections

Throughout the development of NativeSpark, I faced several technical and conceptual challenges that greatly contributed to my learning. One of the major obstacles was related to setting up the backend environment. Initially, I planned to use XAMPP for MySQL database management, but it stopped responding midway through the development process. This issue led me to switch to a standalone MySQL installation, which turned out to be more stable and reliable for integrating with my Spring Boot application.

An important insight I gained during the project came from implementing the recommendation engine. Before this experience, I didn't realize that Python could be used in a modular, object-oriented way, similar to Java. I had previously thought that Python was only suitable for scripting or simple automation. However, as I worked on generating data using the Faker library, building a recommendation engine with scikit-learn, and managing the database with SQLAlchemy, I discovered a strong interest in using Python to work with datasets and derive insights from them—a field I would love to continue exploring.

The most satisfying part of the project was seeing the entire platform come together—from the frontend interface to database interactions and custom product requests. It was rewarding to witness a real, functional system where different users could interact seamlessly. Looking ahead, I'm excited about the idea of deploying the platform to a public domain. I would love to learn how to deploy it on cloud platforms like AWS or Heroku so that real users can access NativeSpark through a live domain and engage with it in a real-world setting.

Hours logs

Date	# Hours	Description of work
12.01.2025	1	I spent time brainstorming the main idea for my project by reviewing my past work to analyze its strengths and identify areas for improvement. I focused on finding a completely new scope of work that could add unique value and elevate the project's impact.
17.01.2025	2	I wrote a project proposal and conducted research to establish the domain, define the problem. Created project contract.
20.01.2025	3	Explore existing literature to identify knowledge gaps. This included developing hypotheses, outlining the research design, methodology, technologies, and expected results, emphasizing how the findings could address practical challenges and contribute to the field. Created Repo on GitHub.
21.01.2025	2	Reading different articles, watching YouTube videos about regarding AI and chatbot implementation.
23.01.2025	1,5	I wrote a project proposal (section project planning and timeline). Creating Gantt Chart.
03.02.2025	2	I started by creating my Spring Boot project and setting up the pom.xml file, ensuring the correct dependencies were included for the initial setup. I added a welcome.html page as the main landing page and configured a HomeController to serve it properly. Along the way, I encountered SQL database connection issues, which I resolved by correctly configuring application.properties with the necessary MySQL settingt. I implemented SecurityConfig.java, allowing public access to specific pages so that I wouldn't need to log in every time. Additionally, I had issues displaying images, which I resolved by placing them in the static/images/ directory and updating the image paths in the HTML.
04.02.2025	1	I worked on integrating my IntelliJ IDEA project with an existing GitHub repository through the desctop version, ensuring proper Git initialization, remote repository linking, and successfully committing and pushing my code.

05.02.2025	1.5	I worked on refining the user registration process for the login page in my application. After users click "Register" on the login page, they are redirected to a user type selection page. I configured Spring Security to allow access to the /select-user-type endpoint, ensuring that users can view this page without needing to authenticate. Additionally, I updated the SecurityConfig to manage redirections correctly and resolved an HTTP 403 (Forbidden) error by adjusting the necessary permissions. I also verified that the "Go Back" button on the selection page accurately redirects users back to the login page.
07.02.2025	2.5	I worked on setting up the user registration flow for the NativeSpark project. Currently, I am implementing a multi-step registration process where users first select their type: Business, Indigenous Entrepreneur, or Basic User. For business users, I have designed a two-step process. In Step 1, users provide their email, password, and user type, which are stored in the users table. In Step 2, users enter their business details and upload a logo, which is saved in the business_users table. I have set up the necessary entities (User and BusinessUser), as well as repositories, services, and controllers to manage data persistence and form submission. Additionally, I created Thymeleaf templates for the registration pages. I also spent time debugging issues related to database mapping. Despite these efforts, saving BusinessUser data in the database is still not functioning correctly, and I will need to continue troubleshooting this issue in the next session. I also plan to implement a similar process for the other two user types.
09.02.2025	3	I focused on debugging the issue where user data wasn't saved in the database after Step 2 of registration. Initially, there were errors with the @JoinColumn reference in the BusinessUser entity. After fixing this, I encountered another problem: uploaded logo files were not being saved correctly, leading to an error. I updated the BusinessUserService to ensure the upload directory exists before saving files. The application was trying to save files in a non-existent directory within the Tomcat temporary workspace. I modified the code to create the directory if it didn't exist and changed the file-saving logic to use `Files.copy()`, ensuring proper storage of uploaded files. I implemented the Entrepreneur User Registration process, following the same structure as the Business User Registration. Similar to the Business Registration, the entrepreneur registration consists of two steps.

		In Step 1, the user enters their email and password, which are stored in the users table with the user_type set to "ENTREPRENEUR." In Step 2, the user provides personal details, including their first name, last name, a brief description about themselves, indigenous identity, and a profile photo. This information is saved in the entrepreneur_users table and is linked to the corresponding user. Writing logs and progress report 1.
11.02.2025	2.5	I implemented the Basic User Registration process, following the same structure as the Entrepreneur User Registration. Similar to the Entrepreneur Registration, the basic registration consists of two steps. In Step 1, the user enters their email and password, which are stored in the users table with the user_type set to "BASIC." In Step 2, the user provides personal details, including their first name, last name, a brief description about themselves, and a profile photo. This information is saved in the basic_users table and is linked to the corresponding user. This time without debugging issues. Initially, passwords were stored in plain text, but I needed to implement encryption using a PasswordEncoder. I updated the UserService class to include password encoding before saving user data to the database. However, I encountered an issue because UserService implemented UserDetailsService but did not override the `loadUserByUsername(String username)` method. This omission caused an error during the authentication setup in SecurityConfig.java. To resolve this, I implemented the missing `loadUserByUsername` method in UserService, ensuring that it retrieves user data from the database and returns a UserDetails object. This change enabled authentication to function properly with Spring Security. Additionally, I updated the authentication manager in SecurityConfig to utilize UserService for user authentication and to encode passwords before validation. Some modifications in htmls files.
12.02.2025	1	Today, I added a confirmation message for successful registration. However, I encountered an issue where the data from Step 2 of the business registration process was not being saved to the database. After debugging, I discovered that the problem was caused by the `event.preventDefault();` call in the form submission, which was preventing the form from sending data to the backend. To resolve this issue, I removed the `onsubmit` event. Once I verified that the business registration flow was working correctly, I

		implemented the same functionality for the Entrepreneur and Basic User registration processes to ensure consistency across all user types. New page subscription, need to connect it to all users.
13.02.2025	2.5	Styling home page. I was trying to implement subscription as step 3 of registration, but half way throug decided to leave for future account settings.
14.02.2025	2	Setting up the loging for users. Need to start again.
16.02.2025	3	Trial number two to set up login for users, unfortunately broke whatever had before.
23.02.2025	3	I have started from scratch developing the login access to the account for users, the programm logges in via users table. At this trial i also had some issues, my login wasn't working due to a conflict between the Authentication class from Apache Tomcat and the one from Spring Security. This conflict caused the authentication check to fail, preventing the user session from being recognized. Consequently, after logging in, the system redirected me back to the login page instead of taking me to the account page. To resolve this issue, I removed the incorrect import (org.apache.tomcat.util.net.openssl.ciphers.Authentication), ensured I was using org.springframework.security.core.Authentication, and updated my AccountController to properly verify if a user was authenticated using Spring Security.
23.02.2025	2	Writing Midterm Report, How to Run The Program document, recording Video and etc.
03.03.2025	2.75	After logging into my account, I focused on displaying user-specific details based on different user types: Basic, Business, and Entrepreneur. Initially, I could only see the email and user type, but additional details like names, descriptions, and logos were absent. The issue stemmed from my AccountController, which was retrieving user-specific entities (e.g., BusinessUser, EntrepreneurUser, etc.), but the data was not being correctly passed to the Thymeleaf template. After debugging, I ensured that the correct user details were loaded and properly mapped. As a result, I was able to fix the missing information. However, the logo for business users still wasn't showing because the uploaded images were stored in the uploads directory rather than in the static directory, which prevented Thymeleaf from accessing them directly. I

		adjusted the image path handling to serve the uploaded images properly, and after that, everything displayed correctly. Additionally, I implemented a logout feature for all users. While users are logged in, they can navigate within the platform, including the "My Account" page, but once they log out, they cannot access the "My Account" page anymore.
04.03.2025	2.67	Today, I worked on implementing the subscription change feature within the user accounts of my web application. When users register, they receive a Free subscription by default. I wanted to give them the option to upgrade or downgrade their plan from their account settings. Initially, I encountered an issue where the subscription wasn't updating in the database. I resolved this by ensuring the backend correctly retrieved the authenticated user and updated their subscription information in the database. Next, I faced a "bootstrap is not defined" error when trying to display the confirmation modal. This occurred because Bootstrap's JavaScript file wasn't loaded properly. I fixed this by adding the correct Bootstrap JS file just before the closing </body> tag. After that, I encountered another problem: the frontend failed with the error "Unexpected token 'S'" when saving the subscription. This happened because the backend was returning a plain text response instead of JSON. To fix this, I made the backend send a JSON response and updated the frontend to properly parse it.
05.03.2025	2.67	Today, I began by separating the controller logic for each user type: Business, Entrepreneur, and Basic. This allowed me to test and manually update the HTML form, ensuring that user data was being correctly updated. Initially, I entered the update paths manually in the HTML form for each user type to verify the data submission process. After confirming that the updates worked for each type individually, I combined all three user types into a single controller. This new setup would handle updates dynamically based on the logged-in user's type. However, I encountered an issue where the "Save" button did not appear when clicking "Edit," and the file input for logos/photos was not enabling for Entrepreneur and Basic users. I resolved this by adjusting the JavaScript to check the user type before enabling the appropriate fields, ensuring that only the relevant inputs were editable for each user type.

06.03.2025	1.88	I was working on enabling file uploads so that Business users could upload a logo, while Entrepreneur and Basic users could upload a profile photo. I also needed to ensure that these updates reflected on the account page. The main issue I encountered was that the file input fields were not activating correctly for different user types, and the uploaded images were not being saved to the database properly. I resolved this by adjusting the JavaScript `toggleEdit()` function to enable the correct input based on user type. Additionally, I updated the form to include both logo and photo inputs, and I modified the backend services and controller to correctly handle file uploads, saving them in the appropriate directories. I worked on implementing a transaction system to log subscription changes when users upgrade or downgrade their plans. Previously, changing a subscription only updated the Subscription table without retaining a record of past changes. To address this, I added a Transactions table to store the subscription type, cost, and timestamp whenever a user updates their plan. One challenge I faced was ensuring that transactions were recorded only for paid plans (Basic and Premium) while treating "Free" as a non-billable change. I resolved this by incorporating conditional logic in the backend to log only the relevant changes. Additionally, I tackled a UI issue where canceled profile edits still displayed modified (but unsaved) data. To fix this, I implemented a Cancel button that refreshes the page to reload the original database values. As a result, both subscription changes and profile edits now function smoothly, ensuring data accuracy and proper tracking.
10.03.2025	2.87	Renamed transations table to orders for future differentiation of transactions for the business and the customers. Also had an error when the table was named order. The error occurs because order is a reserved keyword in SQL. Since order is used for ordering results in queries, using it as a table name causes a syntax error. Today, I established the posting functionality for different user types. While the postings are not yet visible on the front end, I have successfully created the necessary tables in the database: job_postings, project_postings, and products. The controller is set up, allowing business users to manage job and project postings, while entrepreneur users can list products for sale. This

		functionality is role-specific, ensuring that businesses can add job and project postings, whereas entrepreneurs can add products along with images. I also implemented a toggle switch for business users, allowing them to easily switch between job and project postings while ensuring that the correct data is displayed. Additionally, I added a "Back to Account" button to enhance navigation. Access to "My Postings" has been restricted, so Basic users cannot see this option in the navigation menu. During this process, I encountered security role issues and repository mismatches, which I resolved by refining security configurations and adjusting database mappings.
11.03.2025	2.53	Today, I worked extensively on refining the functionalities for posting jobs, projects, and products in my Spring Boot application. I began by ensuring that business users can create job and project postings, while entrepreneurs can list their products for sale. I organized my Thymeleaf templates to properly display these listings, making sure that the images for product postings appear correctly and are aligned to the left. I encountered some issues during this process, so I improved the image upload logic for products. This ensures that uploaded images are stored correctly and displayed on the "My Postings" page.
12.03.2025	2.58	Today, I worked on implementing the delete functionality for job postings, project postings, and product postings. Initially, I encountered a "405 Method Not Allowed" error when attempting to delete a posting. This issue arose because the delete request was being blocked by CSRF protection in Spring Security. Additionally, using @DeleteMapping directly in the controller did not function properly with Thymeleaf forms, which default to sending POST requests. To resolve this, I modified the Thymeleaf delete buttons to include a hidden input field with _method=DELETE. This change allowed the form to properly send a DELETE request while still adhering to CSRF protection. I also updated the controllers to use @PostMapping for the delete endpoints instead of @DeleteMapping, ensuring that requests were correctly processed. I have implemented the Edit Posting functionality, making updates to both the frontend and backend of my application. On the frontend, I created an HTML page that maintains consistent styling with the job and project editing pages. Additionally, I added an image upload

		field, allowing users to update their product image while retaining the existing one if no new file is provided. On the backend, I made several important modifications to the ProductPostingController to support the editing feature. I introduced a GET mapping to load the existing product data into the form and a POST mapping to handle updates. The controller retrieves the product by its ID, ensuring that only authenticated entrepreneurs can modify their own postings. I also implemented logic to manage image file updates, so if a new image is uploaded, the old one is replaced; otherwise, the previous image remains unchanged.
13.03.2025	2.92	Today, I refined the home page functionality to ensure different types of users, including non-logged-in visitors, see the appropriate content. I solved an issue where unauthenticated users were wrongly redirected to the login page by adjusting the controller logic to keep the product list accessible to everyone. I also addressed a problem with product images not displaying for non-logged-in users due to incorrect image path references. I fixed this by updating the Thymeleaf syntax to properly serve images from the static resources directory. For entrepreneur users, I implemented a toggle bar to switch between products, job postings, and project postings. The challenge was keeping the clicked button visually active (highlighted in blue). I resolved this by using JavaScript to toggle Bootstrap classes, ensuring only the active button remained highlighted. Moreover, I fixed internal server errors (500) that occurred when entrepreneur users clicked "Home" in the navbar by updating the controllers to ensure the correct data was fetched for different user roles. Lastly, I improved the navbar behavior so that clicking "Home" directs users to the correct page based on their authentication status and role. Entrepreneurs can now see products, jobs, and projects, while basic users can only view products. These changes greatly enhanced the user experience and navigation consistency. I encountered an issue where removing some commented-out code caused the home button in the navbar to malfunction for unauthenticated users, redirecting them to the login page instead of displaying products. This was confusing since the home page should be accessible to everyone.

		Upon investigation, I found that the commented-out code contained crucial configurations that affected user authentication behavior. To fix the issue, I made sure products were visible to all users, allowing both authenticated and unauthenticated individuals to browse without needing to log in.
17.03.2025	3.067	I implemented a detailed view for all listings, allowing users to access details for products, jobs, and projects with a click. Now, when a logged-in user clicks a listing, it correctly opens the detailed page. In contrast, an unauthenticated user is redirected to the login page before they can access any listing details. I also improved the marketplace page for all users to ensure a smoother experience when navigating between products, jobs, and projects. Previously, clicking the toggle buttons redirected users to the product details page because of the stretched-link class, which made the entire card clickable, including the buttons. To resolve this, I modified the clickable areas so that only the title or image of each listing is clickable. I was attempting to implement a "like" functionality that would allow users to like a product, with the number of likes being stored and updated in the database. However, despite numerous attempts to debug and fix the code, the functionality is not working as expected. The likes are not being saved in the database, and the user interface does not update correctly when a user clicks the like button. Even after verifying that the repository, controller, and JavaScript functions were properly configured, the issue still persists. At this point, I have decided to start from scratch and rebuild the like feature from the ground up to ensure it functions correctly.
19.03.2025	2.75	Today, I worked on implementing the purchase functionality for my project. I created a purchase page where users can buy products from entrepreneurs. This page includes seller information, product details, and a purchase form for users to enter their shipping and payment details. I ensured that the form properly handles user input, including selecting the quantity, filling in address fields, and adding payment information. Additionally, I integrated the form with the backend so that when a user submits it, the data is sent correctly. On the product details page, I added a buy button for all users and a "request custom amount" option specifically for business users. If the quantity of a product is zero, the buy button becomes hidden. I also corrected an issue on the homepage where the user who posted a product could see their own listing, which should not be the case.

		On the backend, I set up the PurchaseController to handle form submissions and process orders. I ensured that the data is saved correctly in the database and that the appropriate associations are made between the user, product, and seller. During this process, I debugged several issues related to missing fields and incorrect mappings in my entities. At one point, I encountered a SQL error because a required field was missing a default value, but I resolved it by updating my entity and database schema.
20.03.2025	2.73	Today, I developed a page that allows users to view their product orders and enables sellers to see who purchased their products. I created separate views for buyers and sellers. Users can access a "My Orders" page that displays a list of all the products they've purchased, including details such as quantity, product information, and shipping data. Sellers can access a "Buyers of My Products" page, which shows who bought each of their listed products. On the backend, I implemented the necessary methods in the "OrderController" to fetch the relevant transaction data based on the logged-in user's role. I also worked on implementing and debugging the save/unsave feature for products, job postings, and project postings on the NativeSpark platform. I built the logic for the save button using a toggle mechanism that updates the database through POST requests to endpoints such as /product/{id}/toggle-save, /job/{id}/toggle-save, and /project/{id}/toggle-save. Each listing entity has a savedByUsers field that maps a many-to-many relationship with the User entity, creating new tables in the database. This setup allows multiple users to save the same listing and enables users to manage their saved items. On the frontend, I added heart icons next to each listing card, which change color when clicked—red when saved and white when unsaved. The initial implementation visually toggled the heart correctly on click; however, I noticed that after reloading the page or navigating away and back to the homepage, the heart icon did not accurately reflect whether an item was saved. To address this issue, I improved the toggleSave function in the JavaScript code to read the server's text response (saved or unsaved) and update the heart icon to display 💖 or ♡ accordingly. This enhancement significantly improved the visual feedback after saving or unsaving an item.
21.03.2025	1.55	Today, I added the CustomProductRequest entity, along with the CustomProductRequestController, CustomProductRequestRepository, and the corresponding request_custom_amount.html page. This addition allows business users to request a custom amount of a product, and these requests

		are now saved in the database. I also included a success message on the page and added a "Back to Home" button to enhance the user experience. In addition, I worked on feedback to better distinguish job postings from project postings. I removed the "Contract" option from job postings to prevent any overlap. The feedback also emphasized the importance of fully implementing individual features before expanding them. As a result, I plan to revise the product purchase flow. Instead of allowing users to buy products directly, they will now be able to add products to a cart and purchase from there, making the functionality more complete and user-friendly.
25.03.2025	3	I implemented full cart functionality for authenticated users. First, I created the `Cart` and `CartItem` entity classes. Each cart is linked one-to-one with a user, and a cart can contain multiple items. Each `CartItem` stores information about the product and its quantity. I added a cart field to the `User` entity and updated the `UserService` so that when a user registers, a new cart is automatically created and linked to them. Next, I created `CartRepository` and `CartItemRepository` to manage the carts and their items in the database. In the `CartController`, I added a `/cart` endpoint to load the user's cart and display it using Thymeleaf. Additionally, I implemented a `/cart/add` POST endpoint to handle adding products to the cart. This endpoint checks if the item already exists; if it does, it updates the quantity; if not, it creates a new cart item. I also added the necessary getters and setters for quantity in the `CartItem` class. On the frontend, I updated the `product-details.html` page to include a form that allows users to select a quantity and submit their choice. The cart page displays all cart items in a table, showing the product name, description, price, quantity, and total. It also manages scenarios where the cart is empty.
26.03.2025	2.5	In the CartController, I replaced the old `addToCart` method, which used form parameters, with a new version that accepts a `CartItemRequest` object in JSON format. This change facilitates smoother interactions on the frontend without requiring full-page reloads. I also implemented proper HTTP responses using `ResponseEntity` to return appropriate status codes and messages. To support this functionality, I created a new Data Transfer Object (DTO) class called `CartItemRequest`, which contains fields for `productId` and

		`quantity`. Additionally, I added the missing getter and setter methods in the `CartItem` entity for both the cart and product. On the frontend, I updated the product details page to handle cart addition using JavaScript's `fetch` method, sending data as JSON and displaying a Bootstrap toast notification to indicate success or error. Furthermore, I developed a `cart-checkout.html` page styled consistently with the rest of the app, which displays products added to the cart, including their name, price, quantity, and total. I added a reusable navbar and footer across some pages to create a more polished and cohesive user experience. I improved the cart functionality by implementing a complete checkout flow. In the CartController, I added a `/cart/summary` endpoint to show the cart summary with total quantity and price, along with a `cart-summary.html` view for a clean presentation. The Cart entity now features a `List<CartItem>` with appropriate JPA annotations, enabling access to all items in the user's cart. I also created endpoints for updating item quantities and removing items at `/cart/updateQuantity` and `/cart/removeItem`. On the frontend, I developed a styled `cart.html` page listing all items with forms for updates and removals, linked to a checkout summary via a "Checkout" button. The `cart-checkout.html` page collects shipping and payment information using a structured form while maintaining a consistent navbar and footer across all views. In the CartController, I added GET and POST handlers for the `/cart/checkout` route. The `getCheckout()` method returns the `cart-checkout.html` view, and the `completeCheckout()` method processes the form submission, capturing shipping and payment details after verifying user authentication. I saved a new Transaction post-purchase by building a Transaction object with shipping details and purchase information. After marking the status as "SUCCESS" and saving to the database, I passed the `transactionId` and `totalPrice` to the success page. Additionally, I updated entity relationships by linking Product to transactions with a `@ManyToMany` association and created a join table called `transaction_product`. The Transaction entity now supports multiple products, laying the groundwork for future multi-item checkouts.

27.03.2025	2	I implemented a basic product recommendation system and developed the data seeding infrastructure to support it. I created a `.env` file and utilized the `dotenv` package to securely load the database credentials into the application. In the `requirements.txt` file, I listed the necessary libraries, including `mysql-connector-python`, `pandas`, `SQLAlchemy`, and `scikit-learn`. I added `alchemy.py` to manage SQLAlchemy-based connections and to facilitate table reflection for easier Object-Relational Mapping (ORM) access to existing MySQL tables. Additionally, I created `basic.py` to implement the recommendation logic, which uses cosine similarity based on product categories and normalized prices, storing the results in a new recommendations table. Furthermore, I developed `seed.py` to generate realistic fake data using the `Faker` library for users, entrepreneur profiles, products, and thousands of transactions. This script ensures relational integrity, generates consistent random data, and populates the MySQL database cleanly through `cursor.execute()` and `commit()`. This setup lays the groundwork for building personalized recommendation features within the application. I implemented a system to display personalized product recommendations directly on the product details page. To achieve this, I created a Recommendation entity to define the relationships between a product and its recommended products. This involved using two relationships: one for the main product and another for the recommended product. In the Product entity, I added a relationship to connect each product with its list of recommendations. On the frontend, I updated the product details page to include a new "Recommendations" section located below the main product information. This section loops through the recommendations using product.getRecommendations() and displays each recommended product as a card, showcasing its image, name, category, and a link to its detail page. There is also more advanced file for recomedations but it reqires a lot of purchase history in transactions table. I will stick to basic recomendations but category and the price.
30.03.2025	2	The python script wasn’t working, and the random generation in the database user and products.

		I have no idea what happened, had to reboot computer few times and created a list of comands that made everything work: cd desktop cd W25_4495_S2_ValeriiaN cd recommendations Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser venv/Scripts/activate pip install -r requirements cd src python basic.py python -m db.seed python basic.py Writing progress report.
02.04.2025	2.5	Today, I spent time testing the program and improving the structure and functionality across the various pages. I separated the navbar and footer into individual Thymeleaf fragment files to reduce code repetition and successfully connected them to each page using `th:replace`. I also resolved some issues with Bootstrap by ensuring the correct scripts were properly loaded, which fixed the dropdown menu that was not functioning on certain pages. Debugging and testing this took quite a while across different views. As the program expands, it takes more time to test all of its functionality. I have been trying to separate it into different blocks to ensure that changes in one block do not affect the others, but it is not working that way al the time. During testing, I encountered a few issues. The "Buyers of My Products" page in the entrepreneur account is not displaying data due to recent changes made in the database schema. I attempted to fix this, but it is still not working as expected. Additionally, I discovered that when a user purchases a product, the product quantity is not updating in the database. This issue needs to be addressed to maintain accurate stock levels.
03.04.2025	3	Today, I fixed the buyers' page, which had been crashing due to a change in the database schema. The transaction entity no longer had a direct reference to the product, causing an error in the Thymeleaf template that attempted to access `sale.product.productName`, which no longer existed. To resolve this

		issue, I updated the page to loop through `transactionProducts` and accessed the product information from there. Additionally, I addressed some styling problems on the user registration page. Yesterday, I accidentally removed Bootstrap, but I reintroduced it today and cleaned up the layout. Fixed the product qty update after sucesfull purchase. Today, I encountered an issue while trying to delete a product. I received a foreign key constraint error because the product was still referenced in the recommendations table as a recommended product. Essentially, Hibernate couldn't delete it since other rows were still pointing to it. To resolve this, I realized that I had only mapped the recommendations list for the products that recommend others and not the inverse — the products that are being recommended. So, I added a new list called `recommendedBy` in the Product entity with `mappedBy = "recommendedProduct"` and used `cascade = CascadeType.ALL` along with `orphanRemoval = true`. After making these changes, deleting the product worked perfectly, and all related recommendations were removed automatically. Testing the entire thing....
04.04.2025	1.67	Today, I worked on the custom product request feature. I updated the product details page so that when the quantity is 0, there is now a "Request Custom Amount" button visible for all users, including both Entrepreneur and Basic users, not just Business users. Additionally, I built the "My Requests" page to list all custom requests a user has made. I fixed some issues with the controller mapping, which previously returned a 404 error, and added the "My Requests" link into the services dropdown in the navbar. To ensure it displayed correctly, I had to change its class to "dropdown-item." Also, i have updated the design of the request custom amount page.
04.04.2025	0.67	I worked on a feature that allows entrepreneurs to view a list of users who submitted custom product requests. I created a page that iterates through each CustomProductRequest and displays the product name, requested amount, request time, and the requester's information. Initially, I attempted to show the requester's full name using `req.user.firstName + ' ' + req.user.lastName`, but the application crashed with a 500 Internal Server Error. Upon reviewing the User entity, I discovered that it does not contain fields for first name or last name; it only has an email field. To resolve this issue, I updated the template to display just the email instead: `<span

		th:text="${req.user.email}">`. This adjustment resolved the problem, and the requests now load correctly on the page.
06.04.2025	1.83	Today, I implemented search functionality on the home page that enables users to search for products using keywords found in the name or description. Additionally, I expanded the search feature for entrepreneurs to include job and project postings based on keywords found in the descriptions and required skills. I also introduced filtering options: for products, users can filter by category, which dynamically loads from the database, while for jobs, entrepreneurs can filter by employment type (Full-time, Part-time, Internship). These search inputs and filters were integrated into the existing home page form and connected to the controller logic to apply the filtering when rendering the data. Creating questions for basic users for the survey: 1.How easy was it to navigate the platform and find what you were looking for? 2. Did the product listings (images, descriptions, pricing) provide enough information for you to make a decision? 3. How likely are you to purchase Indigenous-made products through NativeSpark? 4. What features do you find most valuable as a basic user? 5. If a product is out of stock, how useful is the 'Request Custom Amount' feature for you? 6. What additional features would make you more likely to use this platform regularly? 7. How comfortable were you with the design and layout of the site? 8. Did you experience any issues while browsing, searching, or saving items? 9. Based on what you've seen and the idea presented, how likely are you to recommend NativeSpark to others?
09.04.2025	2	Writing final report. Updated the Installation Instructions.
10.04.2025	0.5	Explaining the idea to my basic users and business to collect the feedback
10.04.2025	1.75	User Guide report

11.04.2025	4	Final Report Presentation Speech Deck
13.04.2025	1.5	The recommendation script was failing because I used the scaler object without defining it inside the function where it was needed. I fixed it by moving scaler = MinMaxScaler() inside the loop to ensure it's properly initialized for each category subset. Populating the program a little bit.

References

1. Statistics Canada. (2023, July 18). Indigenous entrepreneurship in Canada. Retrieved from https://www150.statcan.gc.ca/n1/daily-quotidien/230718/dq230718c-eng.htm
2. Richmond, C. A. M., & Cook, C. (2017). Creating conditions for Canadian Aboriginal health equity: The promise of healthy public policy. Social Science & Medicine, 176, 93–103.
3. Indigenous Corporate Training Inc. (n.d.). 8 key issues for Indigenous peoples in Canada. Retrieved from https://www.ictinc.ca/blog/8-key-issues-for-indigenous-peoples-in-canada
4. **Indigenous Industrial and Contracting Network. (n.d.). Retrieved from https://www.imcn.ca/**
5. **Indigenous Business Development Services. (n.d.). Retrieved from https://ibdssk.com/**
6. Python Software Foundation. (n.d.). *venv — Creation of virtual environments*. Python 3 documentation. Retrieved April 9, 2025, from https://docs.python.org/3/library/venv.html

Tools used

AI Tools were used for coding backend and frontend, ML Scripts, reports (for proper English, Grammarly to polish things off).

7. W3Schools. (n.d.). *W3Schools Online Web Tutorials*. Retrieved April 9, 2025, from https://www.w3schools.com/ (Coding)
8. OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/chat (Paid, used for coding backend and frontend, ML Scripts, debugging, Reports)
9. Grammarly. (n.d.). Grammarly. Retrieved April 9, 2025, from https://www.grammarly.com/ (Reports)
10. DeepSeek. (n.d.). *DeepSeek: Exploring Uncharted Territories*. Retrieved April 9, 2025, from https://www.deepseek.com/ (used for coding backend and frontend)
11. Canva. (n.d.). *Amazingly Simple Graphic Design Software – Canva*. Retrieved April 9, 2025, from https://www.canva.com/ (Paid, Presentation)

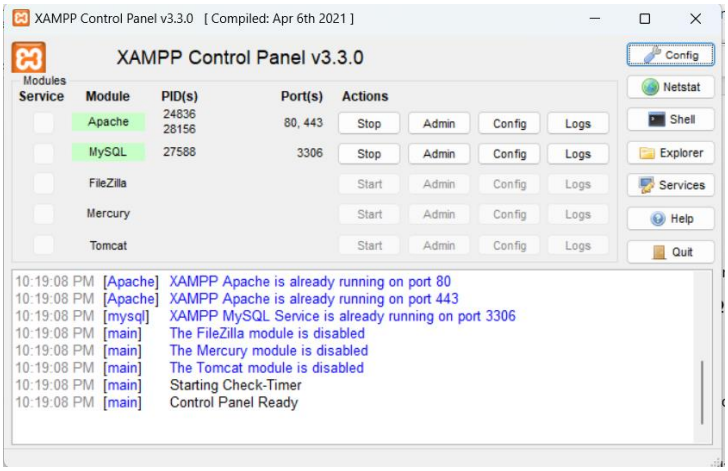
Downloads links:

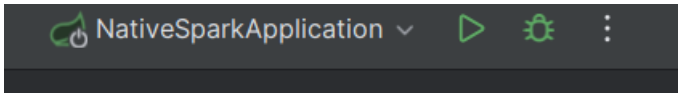
12. Python Software Foundation. (n.d.). *Download Python*. Retrieved April 9, 2025, from https://www.python.org/downloads/
13. Oracle Corporation. (n.d.). *MySQL: The world's most popular open source database*. Retrieved April 9, 2025, from https://www.mysql.com/
14. JetBrains. (n.d.). *IntelliJ IDEA: The IDE for Pro Java and Kotlin Development*. Retrieved April 9, 2025, from https://www.jetbrains.com/idea/ (Douglas Credentials)
15. Apache Friends. (n.d.). *Apache Friends - XAMPP*. Retrieved April 9, 2025, from https://www.apachefriends.org/

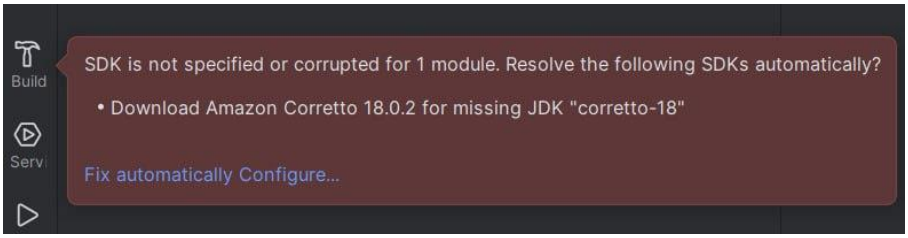
Other

16. Friends help

Appendix A: Installation Guide

1. Start XAMPP:
    - Open the XAMPP control panel.
    - Start the **Apache** and **MySQL** services.
    - 

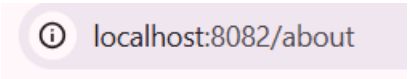
2. Open the NativeSpark Project:
    - Launch **IntelliJ IDEA**.
    - Open the NativeSpark project
3. Run the Application:
    - Click the **Run** button in IntelliJ IDEA to start the application.
    - 

    - In case of the warning like below, please press “Fix auto”
    - 

4. Run the Python Script:
    - Open the Terminal and run the following commands:
    - cd desktop cd W25_4495_S2_ValeriiaN

- cd reccomendations Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser
- venv/Scripts/activate
- pip install -r requirements
- cd src
- python basic.py
- python -m db.seed (if needed to populate the database with fake data)
- python basic.py (repeat the following command to update the product recommendations based on the added products to the database)

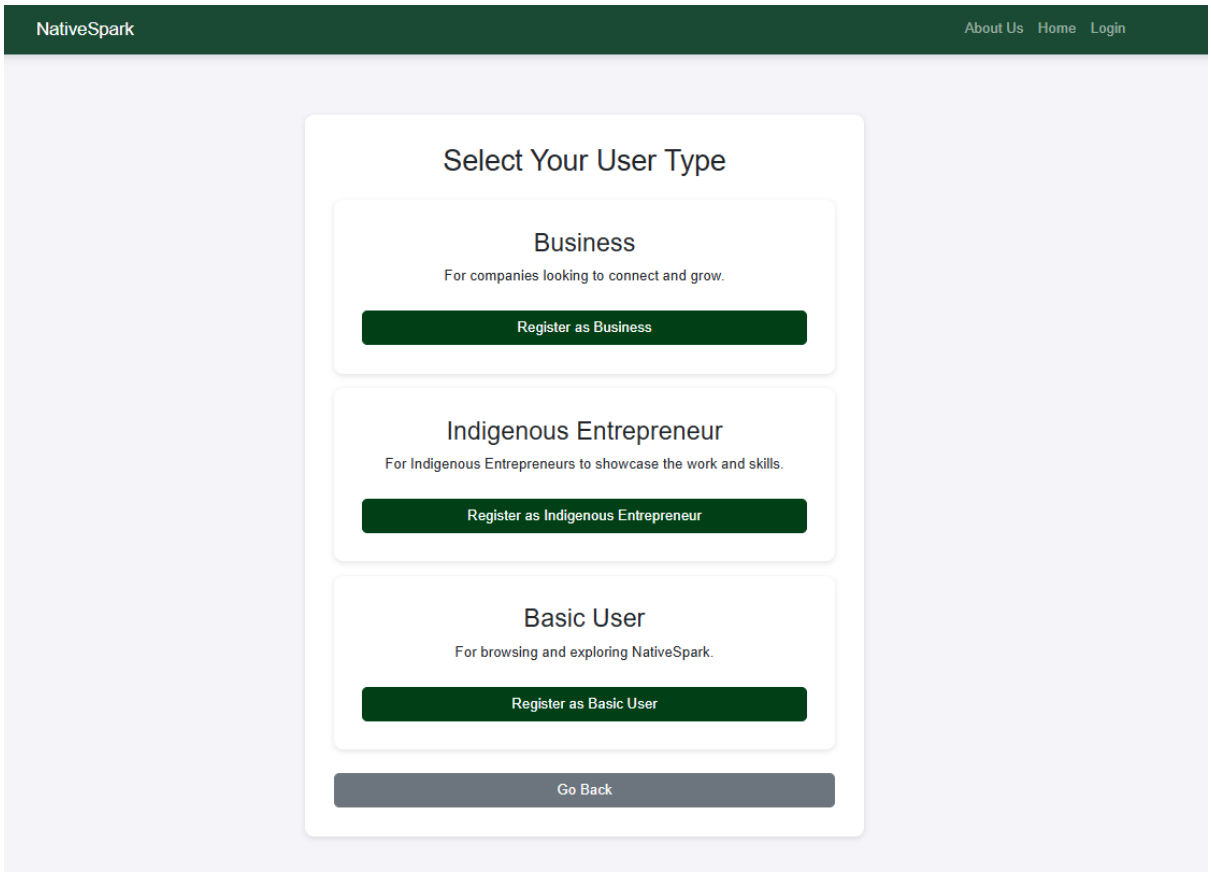
5. Access the Application:
   - Open a web browser.
   - Navigate to http://localhost:8082 to access the NativeSpark application.



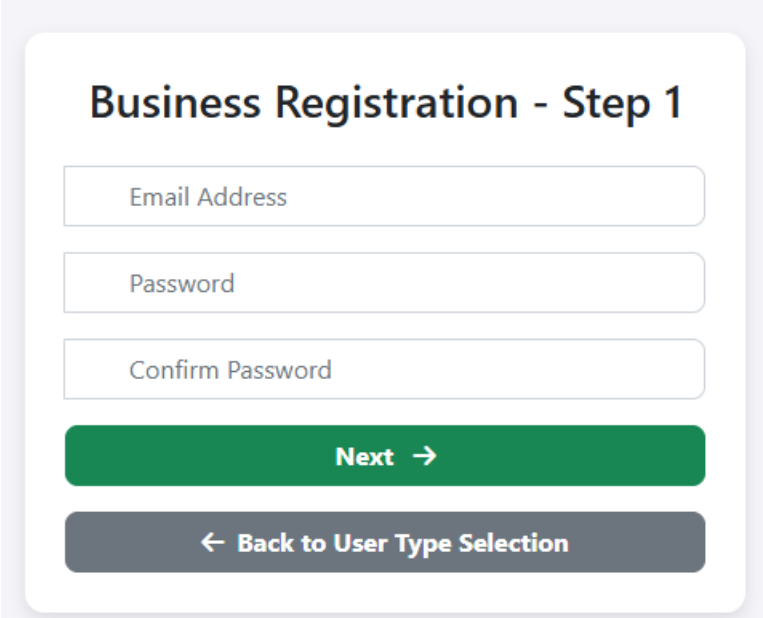
Appendix B: User Guide

Registration

The user must select a role before accessing the application and registering.

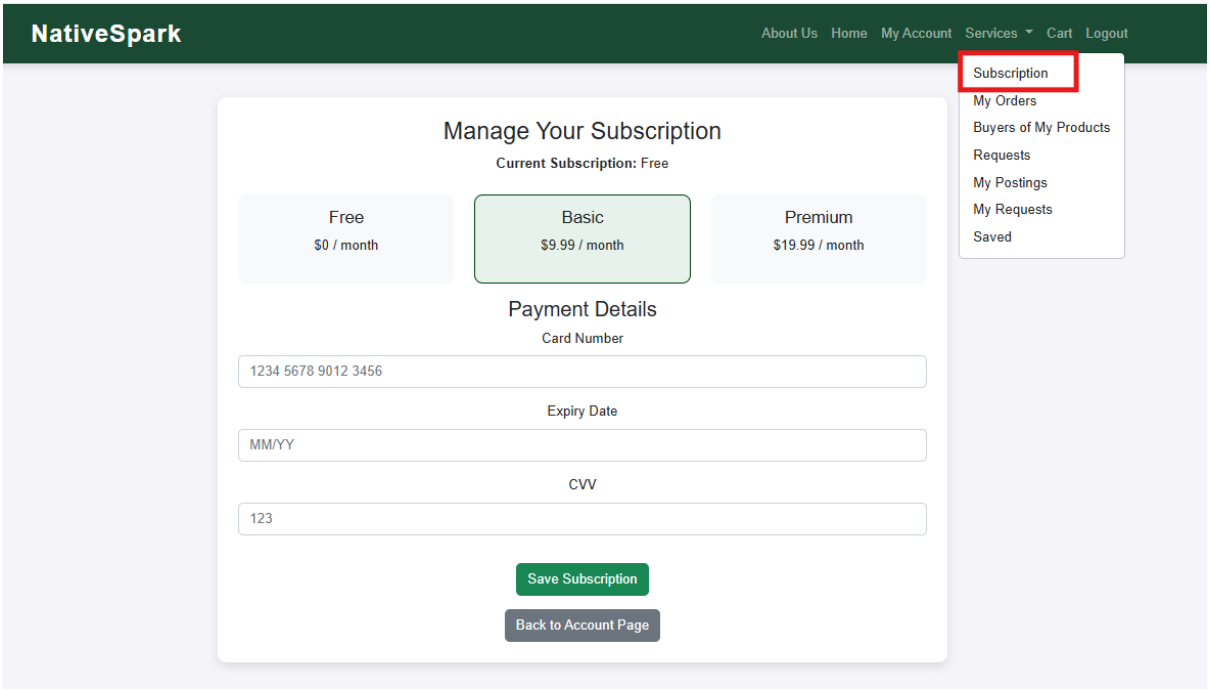


The first step for all three users is the same: to choose the login (email) and password. The second step askes for specific information addressed to the user type.



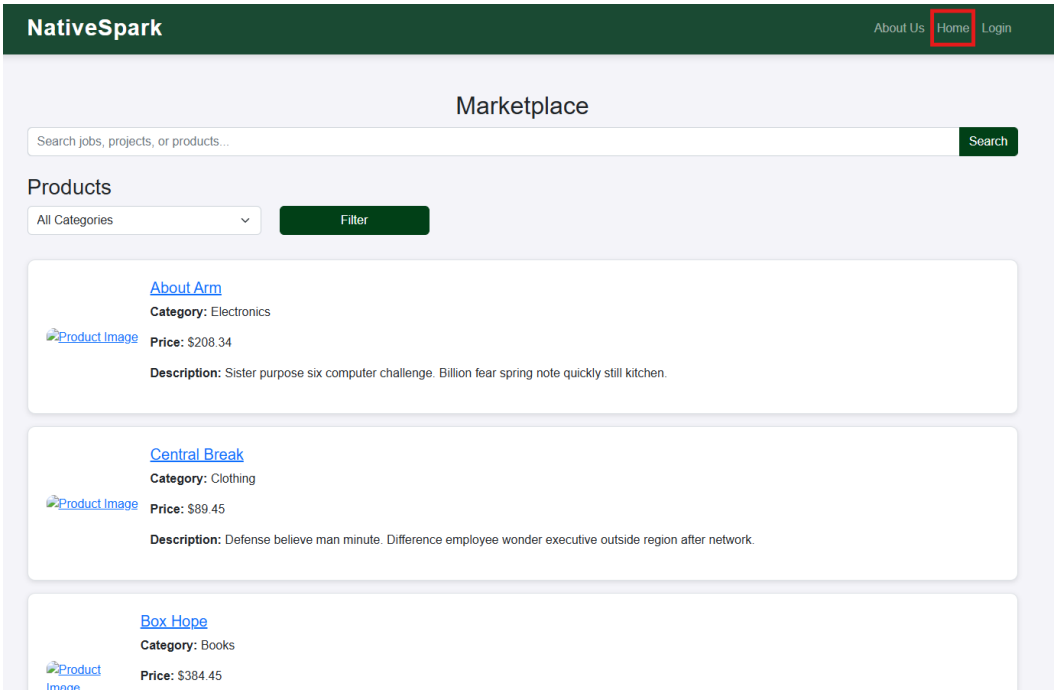
Subscription

Thee uses can change their subscription Service dropdown



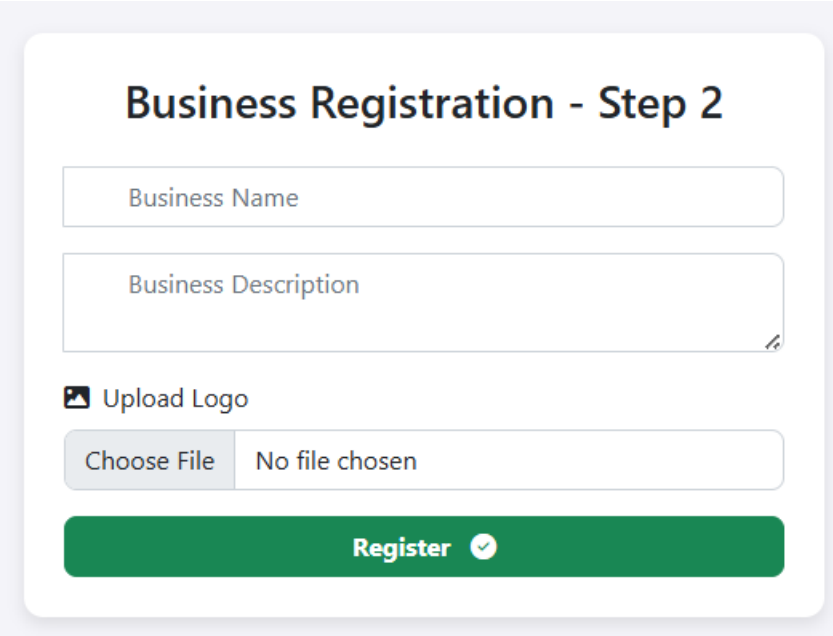
Home Page

Non-logged-in users can browse products posted by Indigenous entrepreneurs on the home page. To view product details, users must log in. They can also search for or filter products.



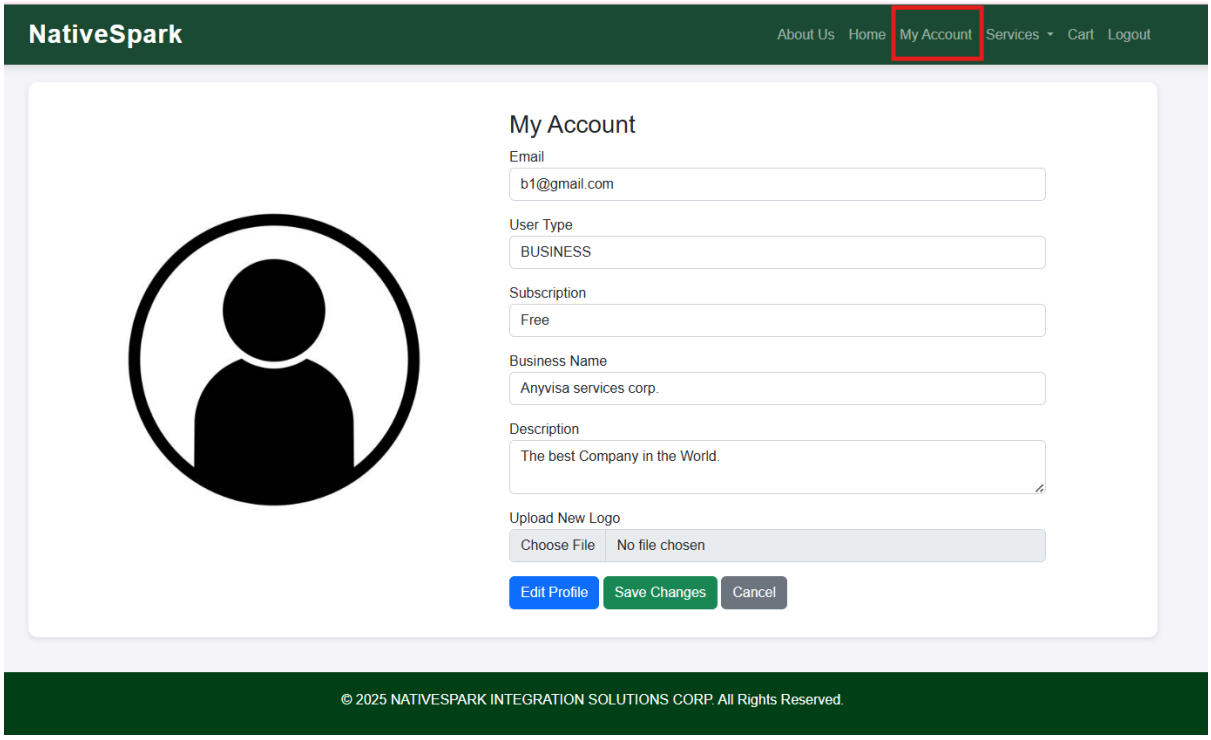
Business user

Registration



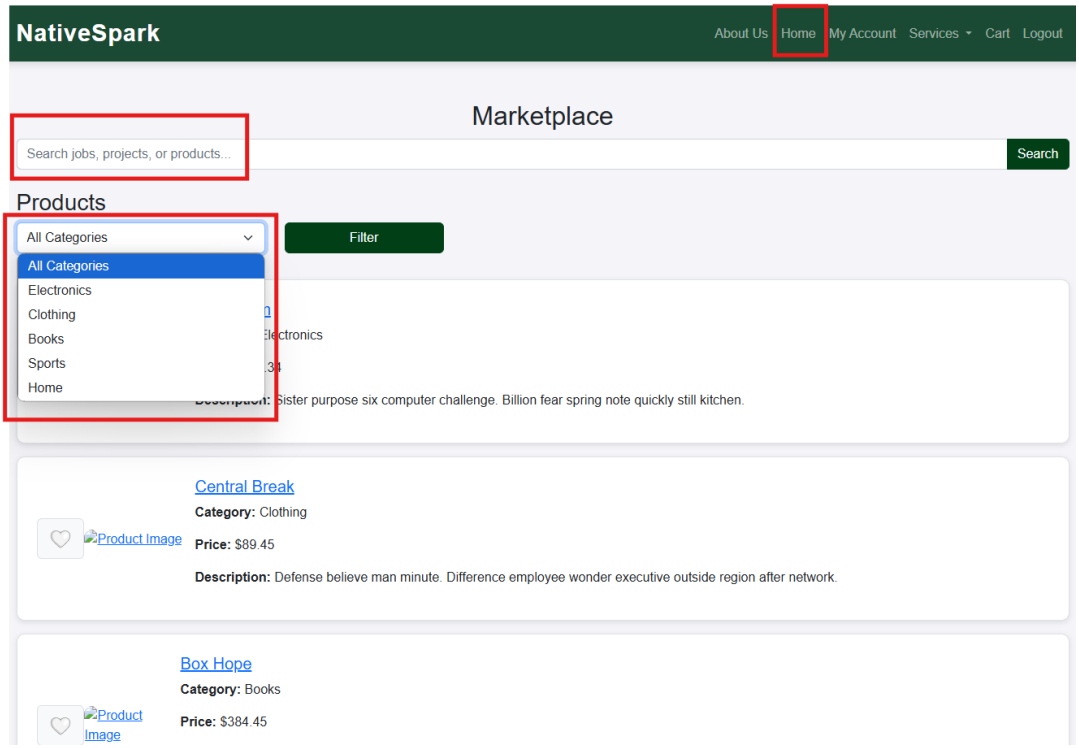
My Account

In my account you can edit information about yourself.

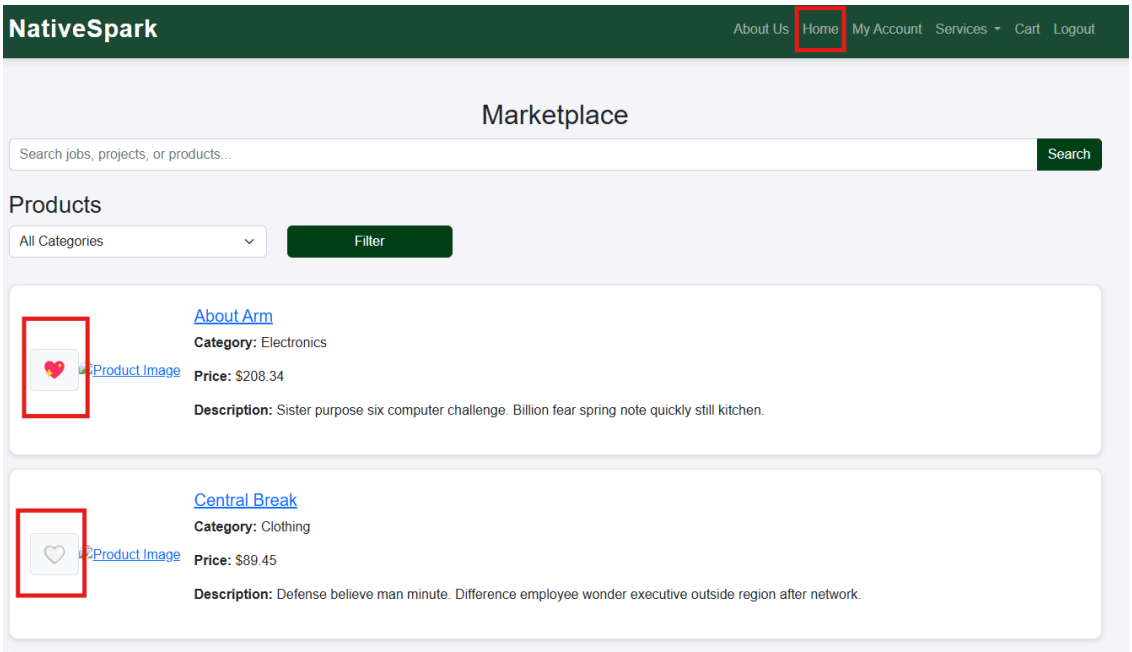


Home Page

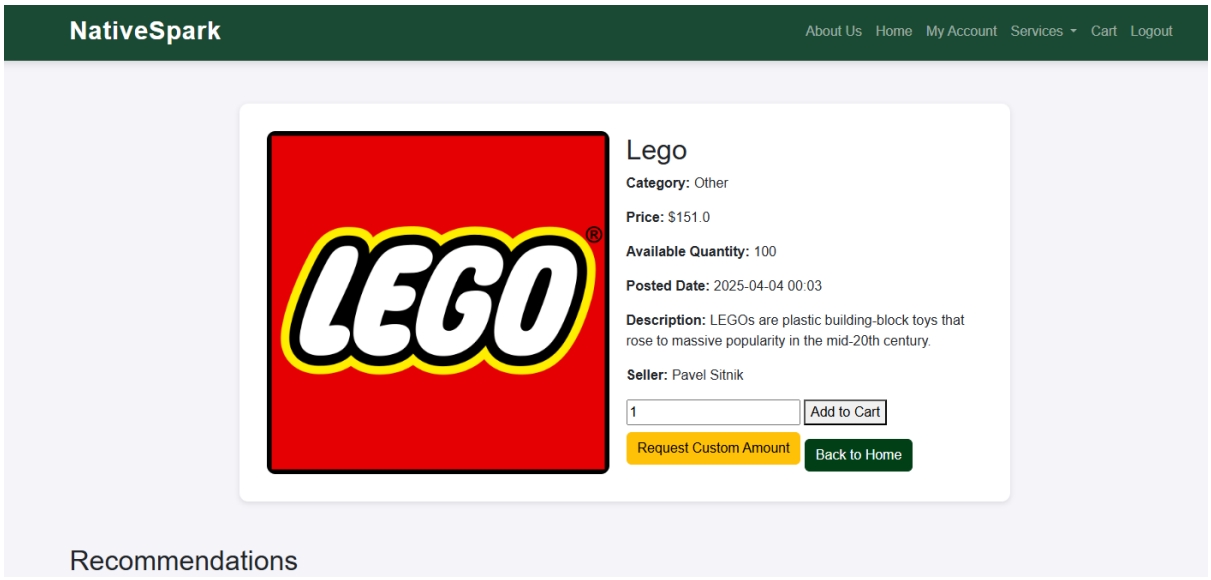
Business users on the home page can filter products by existing categories or by keywords searched in the product description and name.



Users can save their favourite products for later purchase or set a custom amount if their desired quantity exceeds the stock available at that moment.

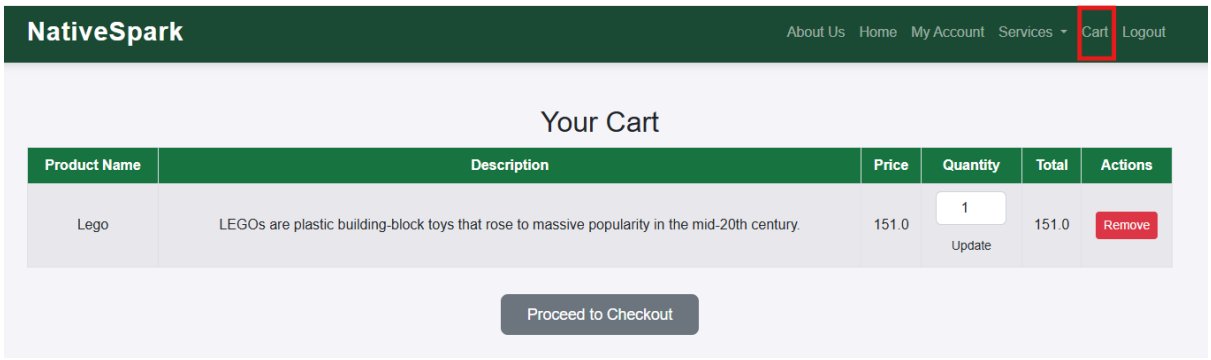


Additionally, by clicking on the product image or name, a detailed summary will open.

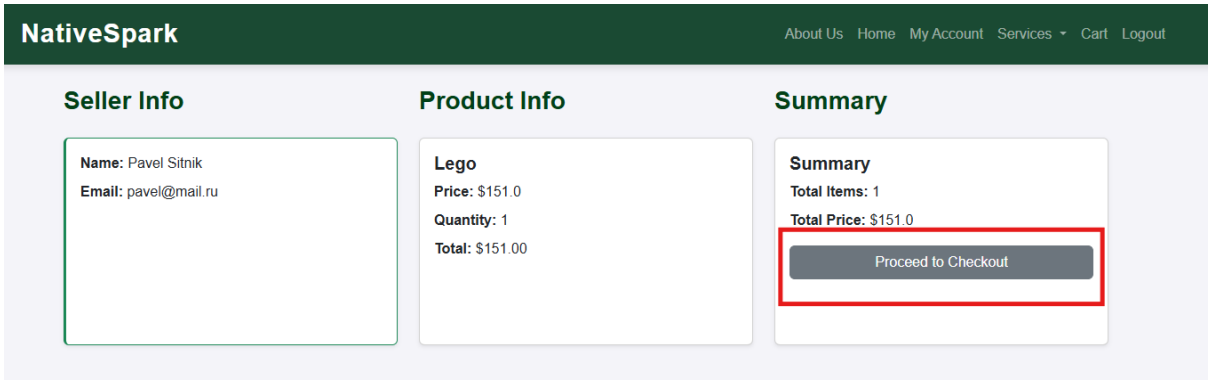


Cart

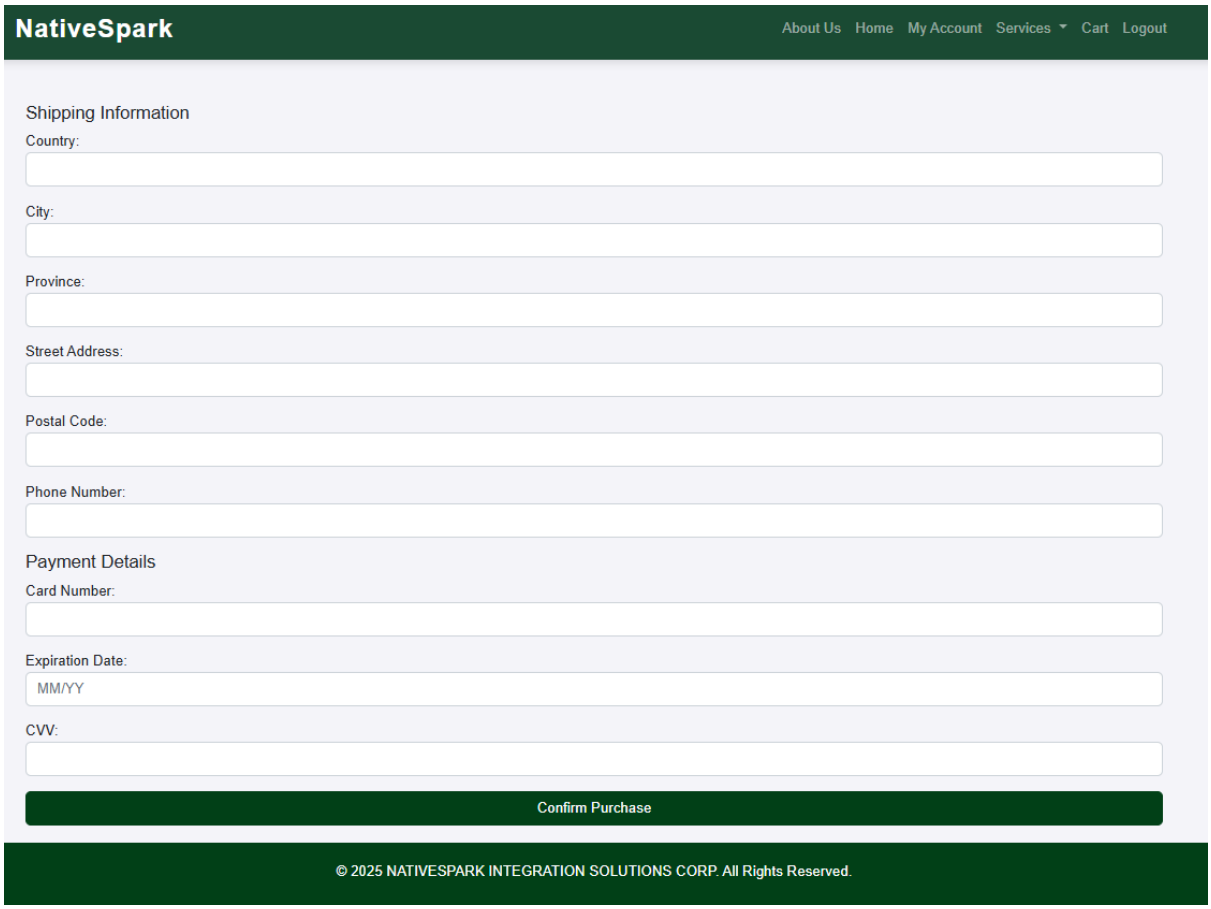
The product can be added to cart and proceeded to check out.



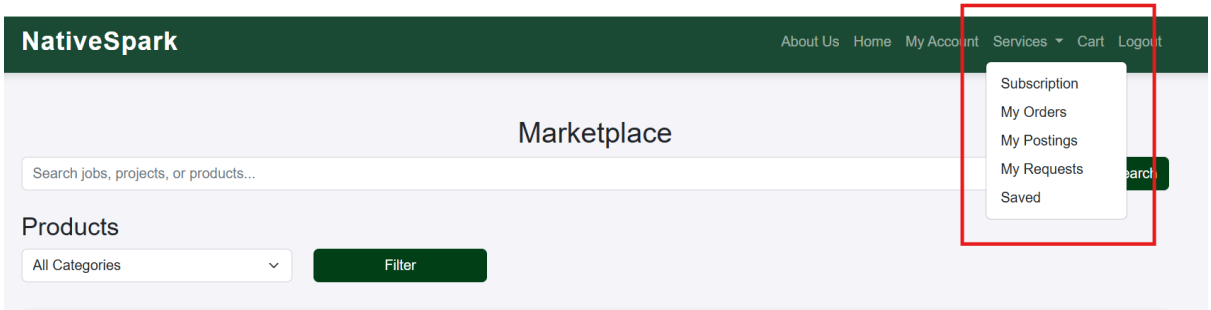
On summary page the information about seller and the product will be shown.



To place the order shipping information needed.

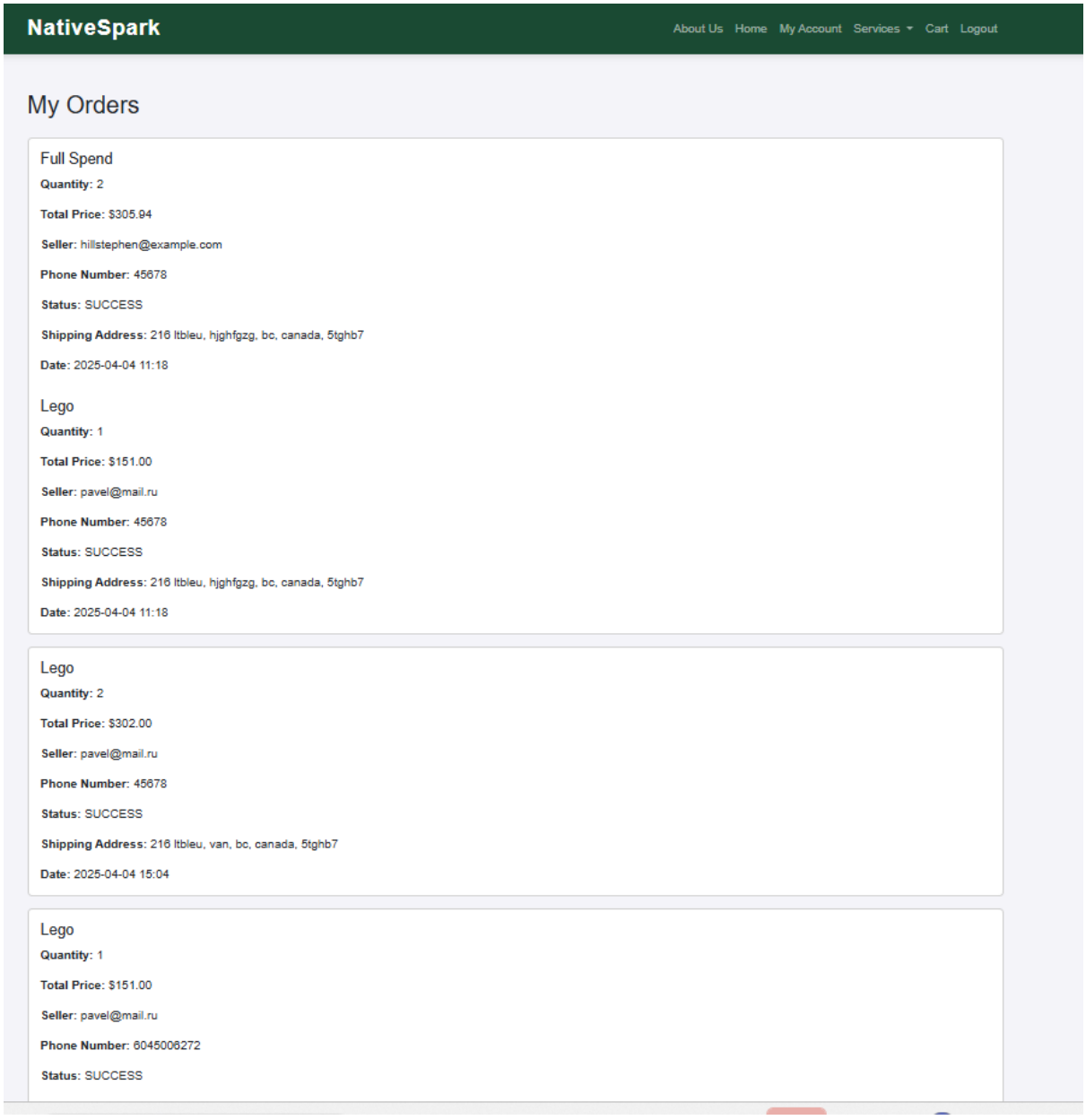


Services dropdown



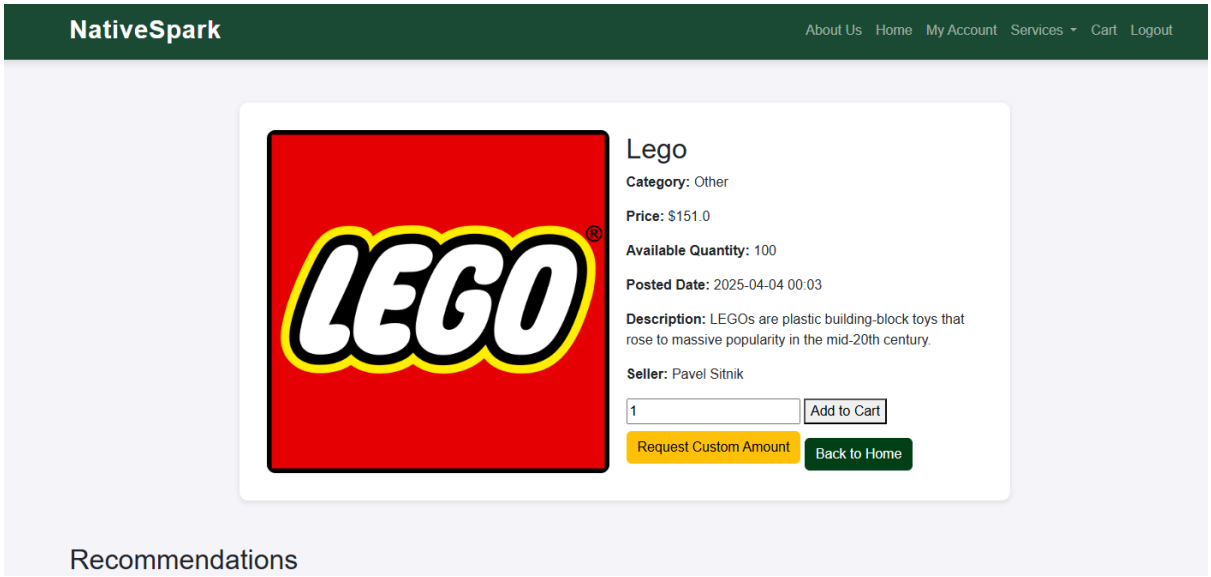
My Orders

To see the information about purchased products click My Orders

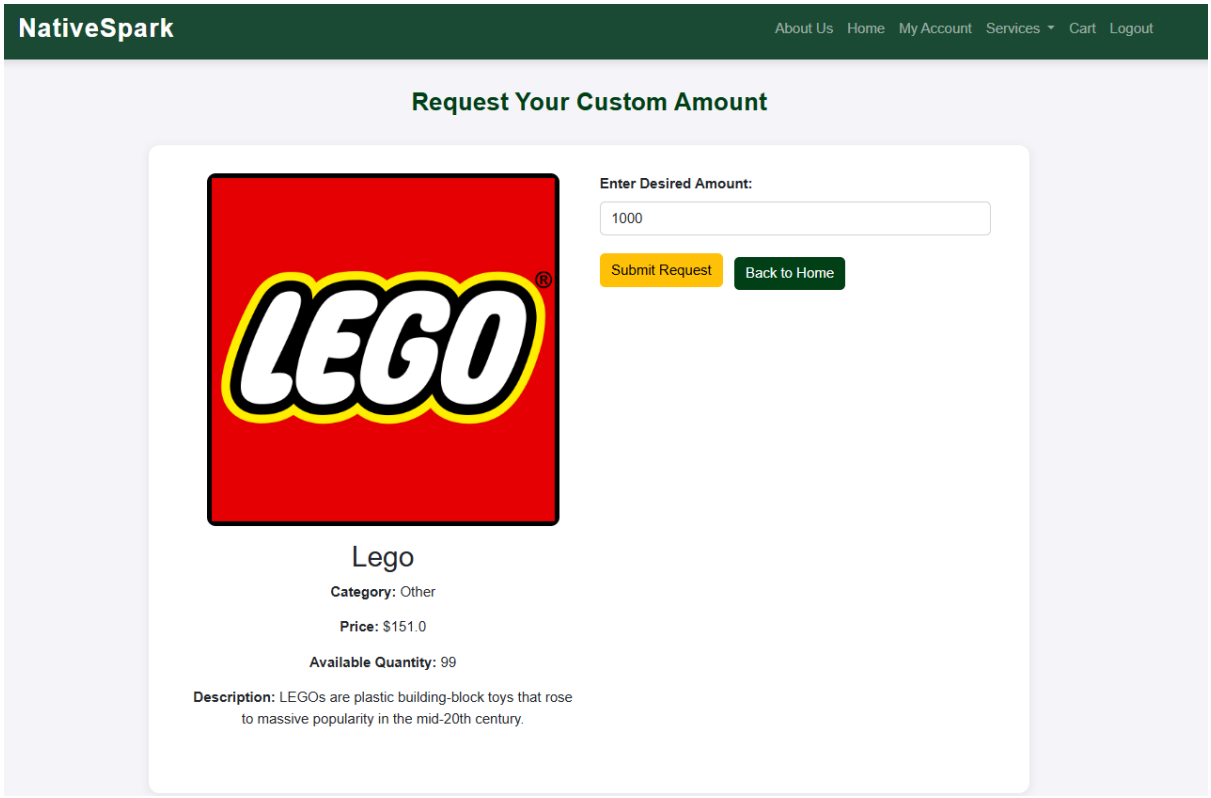


My Requests

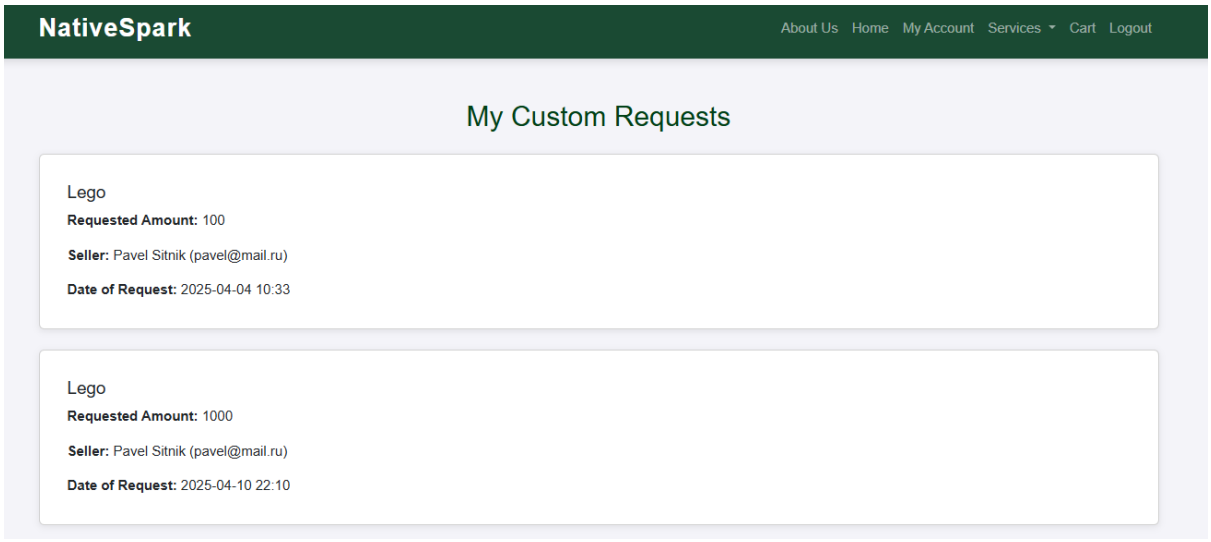
To request custom amount the user needs to submit it on product details page.



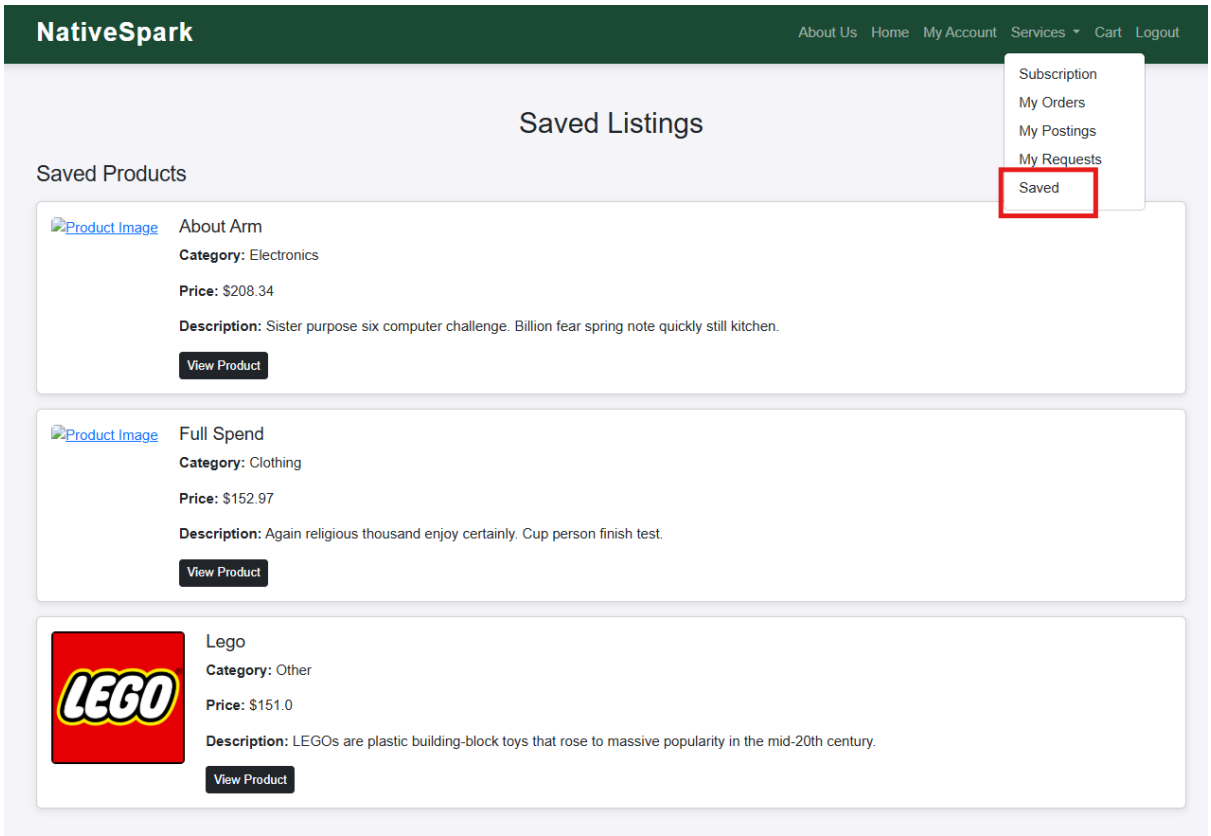
Enter custom amount and place the request



To see the requested press My requests in Services dropdown.

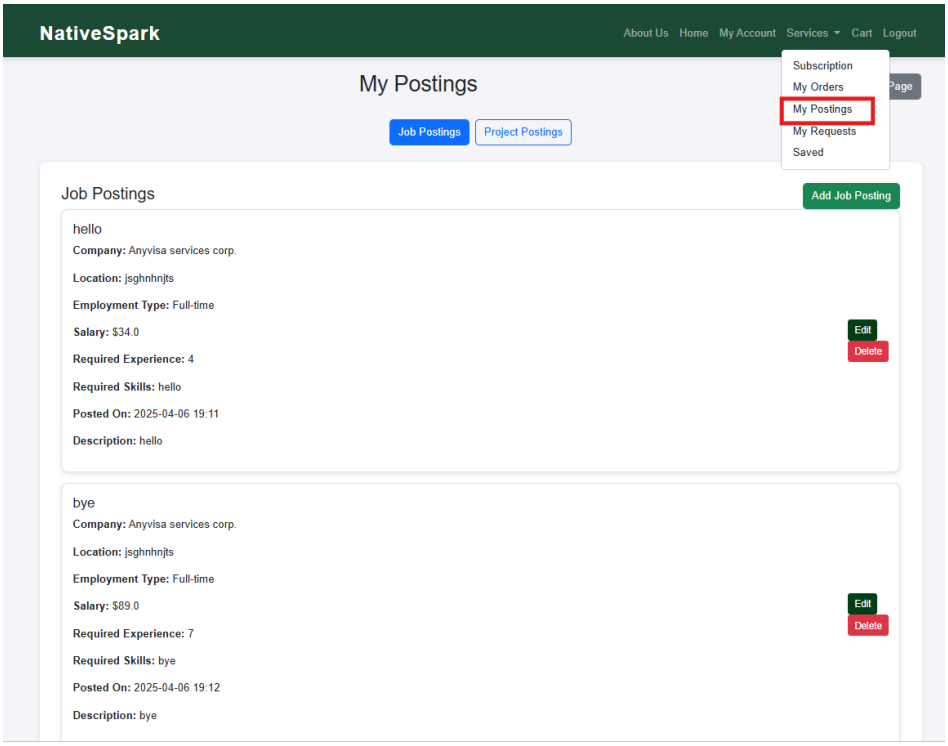


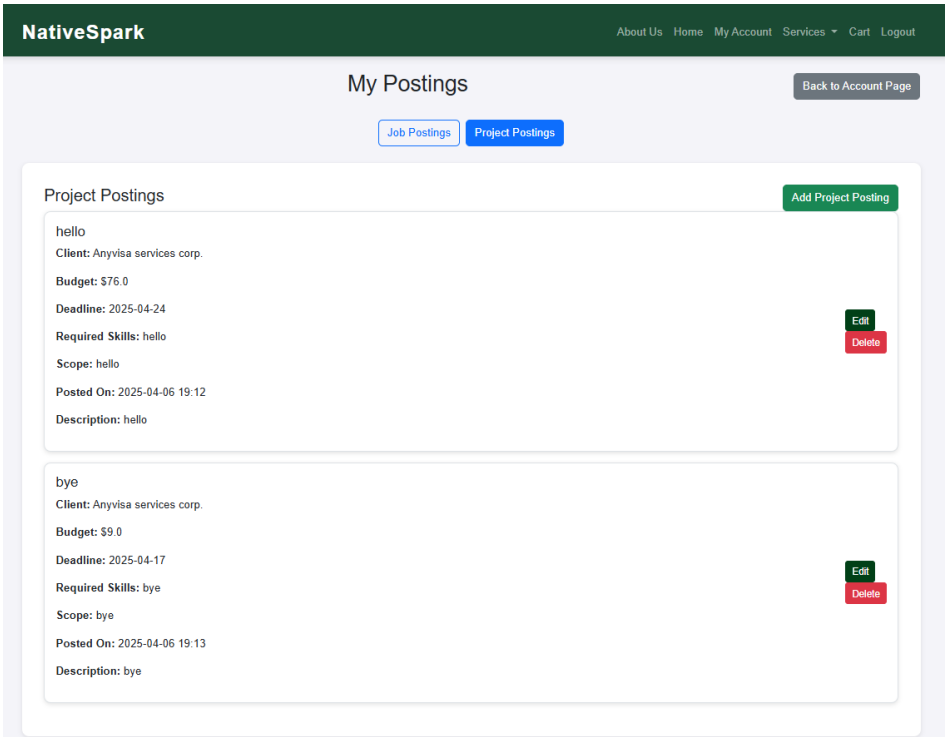
Saved



My Postings

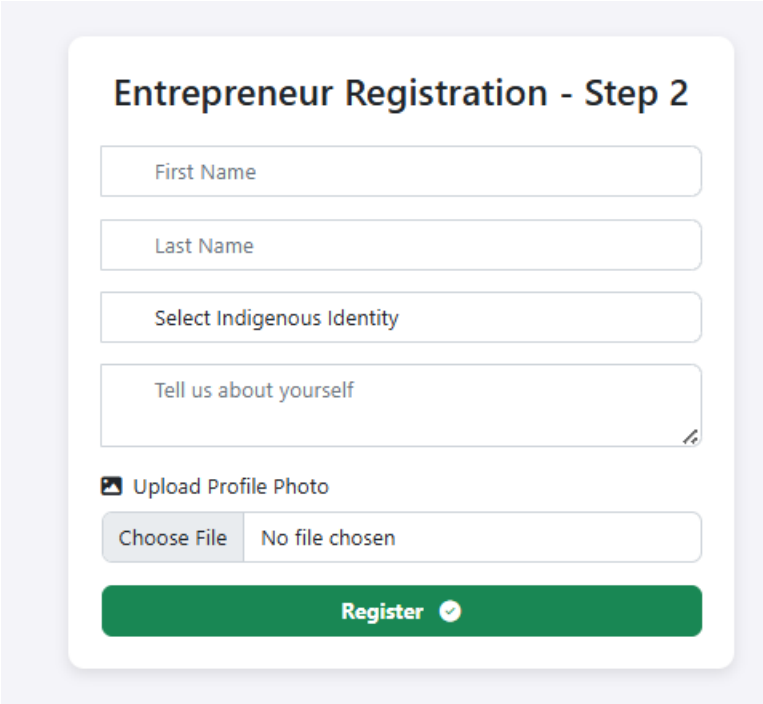
The Business User can post/edit/delete jobs and projects.





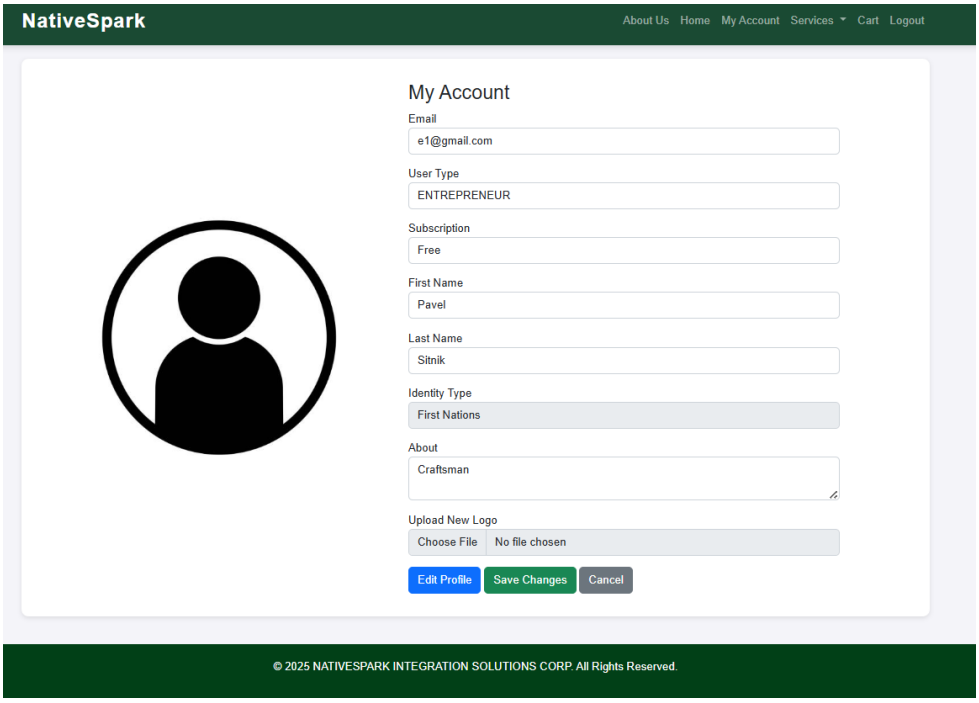
Entrepreneur User

Registration



My Account

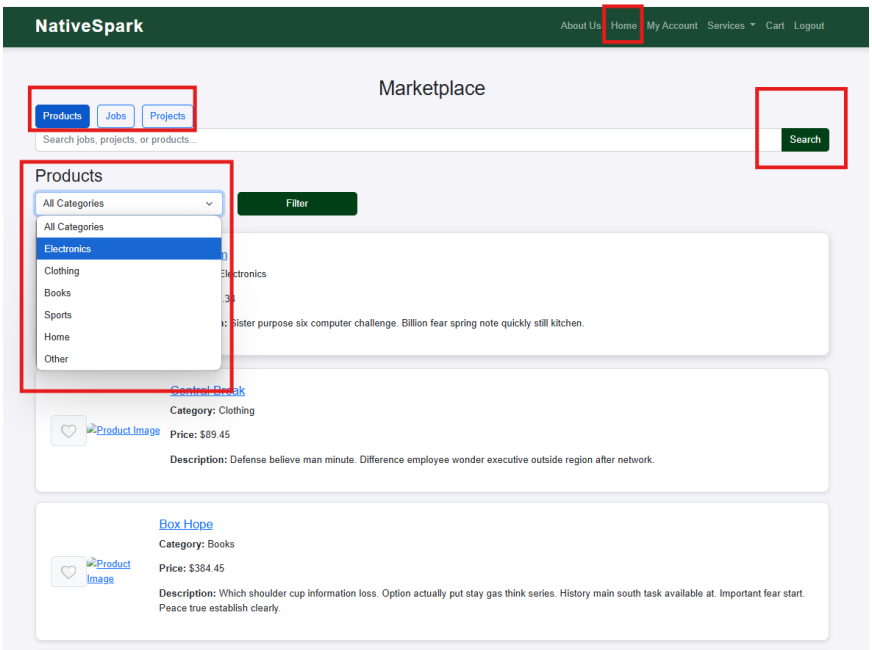
In my account you can edit information about yourself.



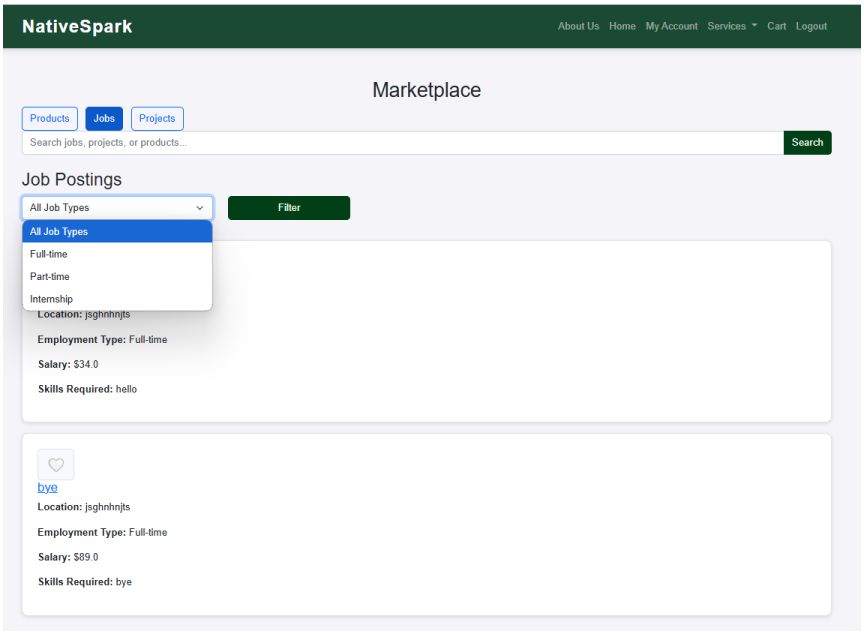
Home Page

Users can save their favourite products, jobs and projects.

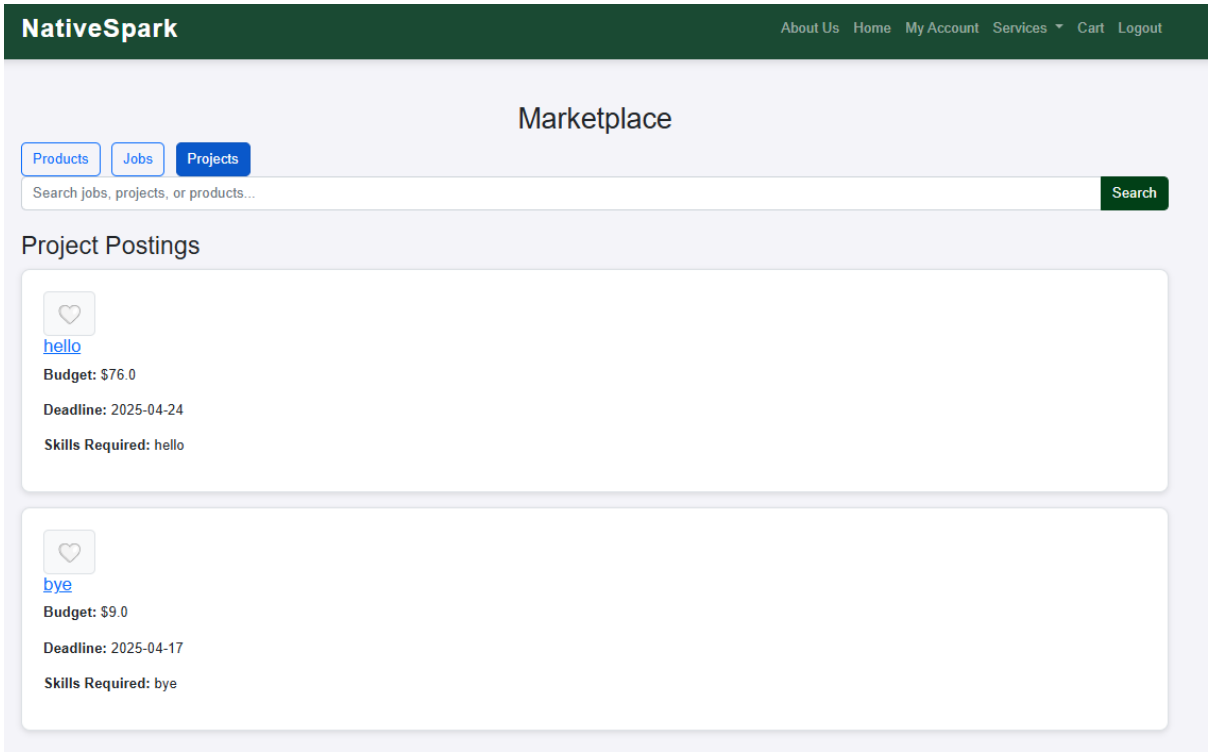
Entrepreneurs users on the home page can filter products by existing categories or by keywords searched in the product description and name.



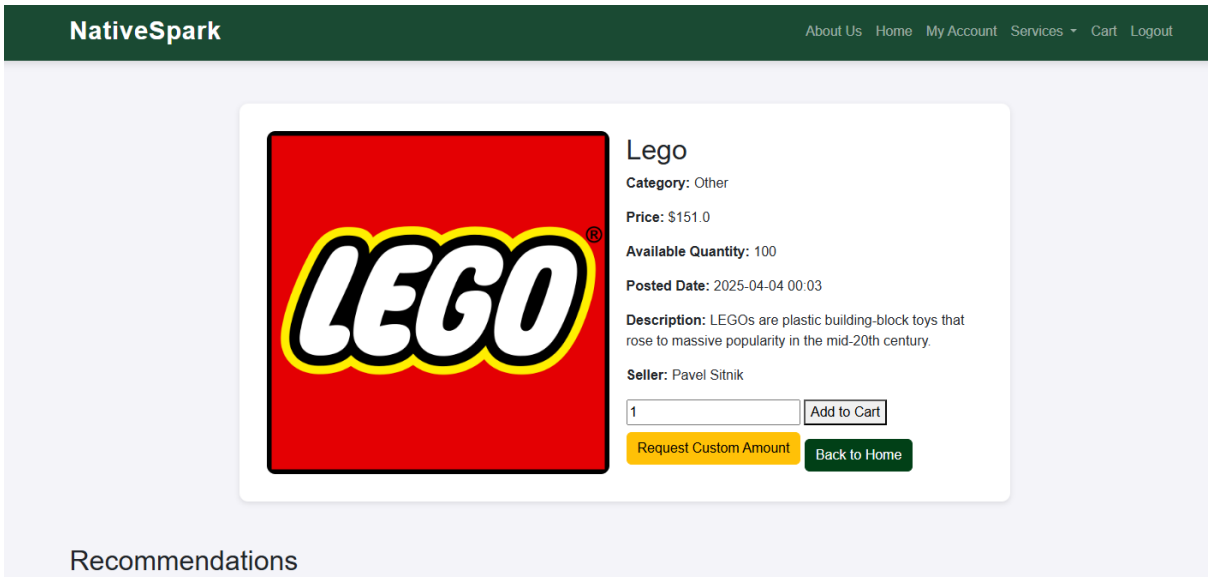
Entrepreneurs users on the home page can filter jobs by existing categories or by keywords searched in the description and name.

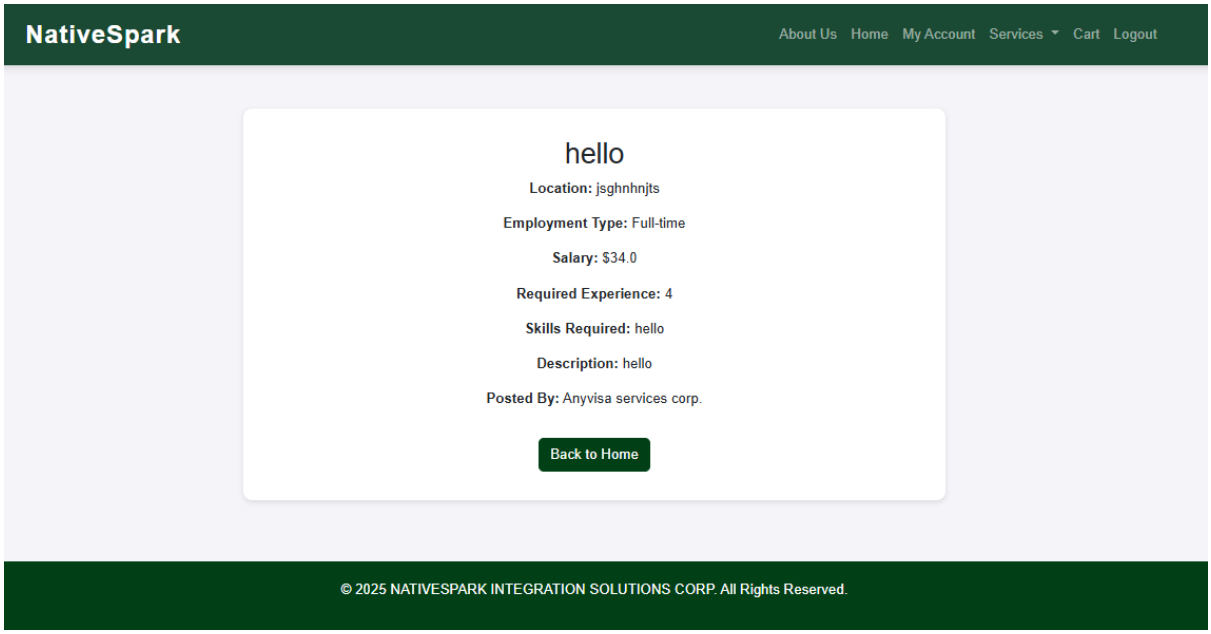


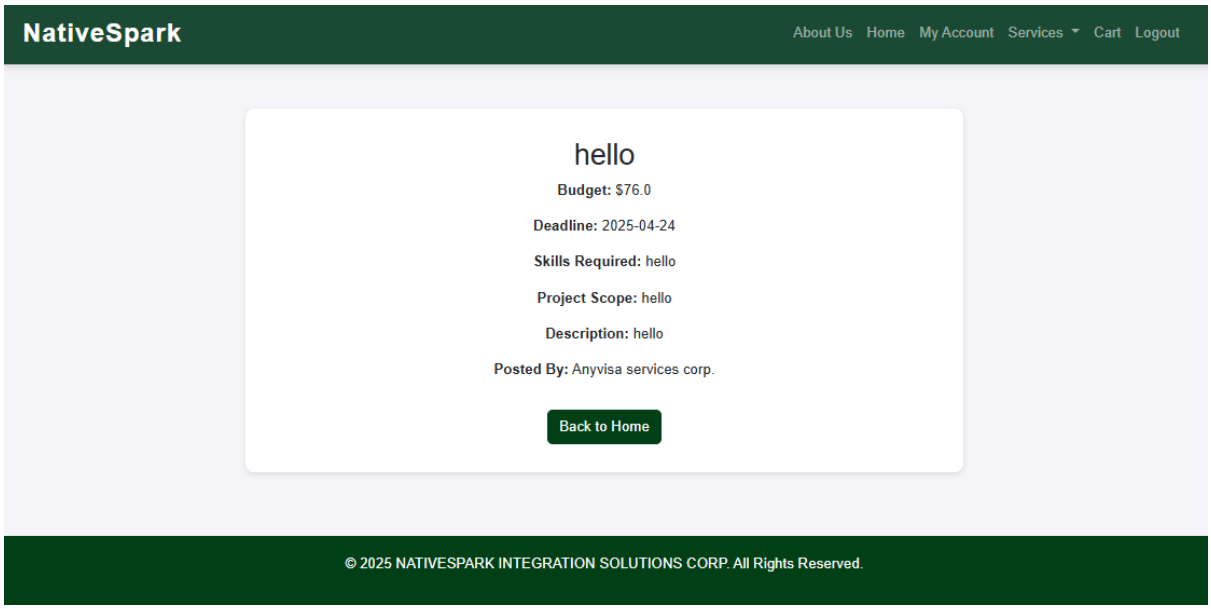
Entrepreneurs users on the home page can filter projects by existing categories or by keywords searched in the description and name.



Additionally, by clicking on the product image or name, a detailed summary will open.

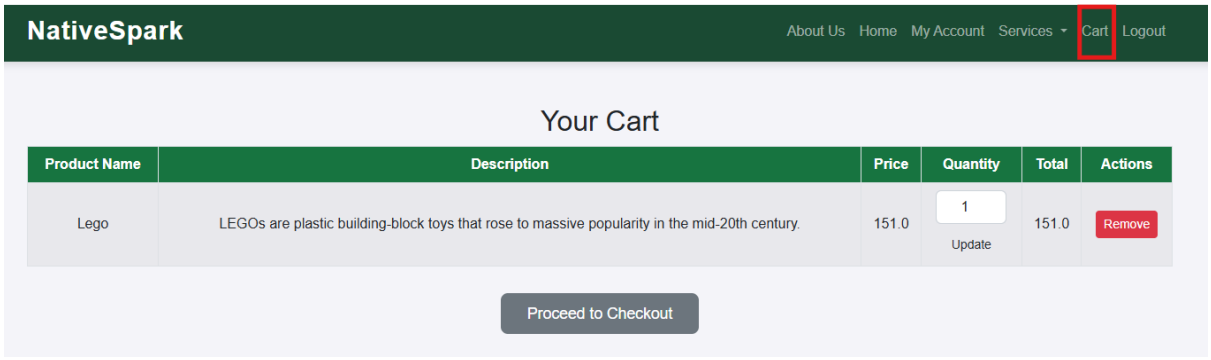




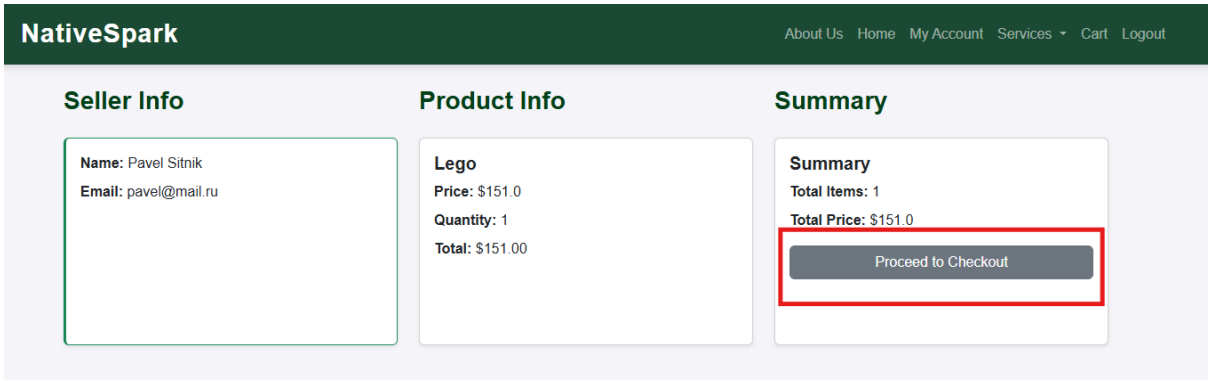


Cart

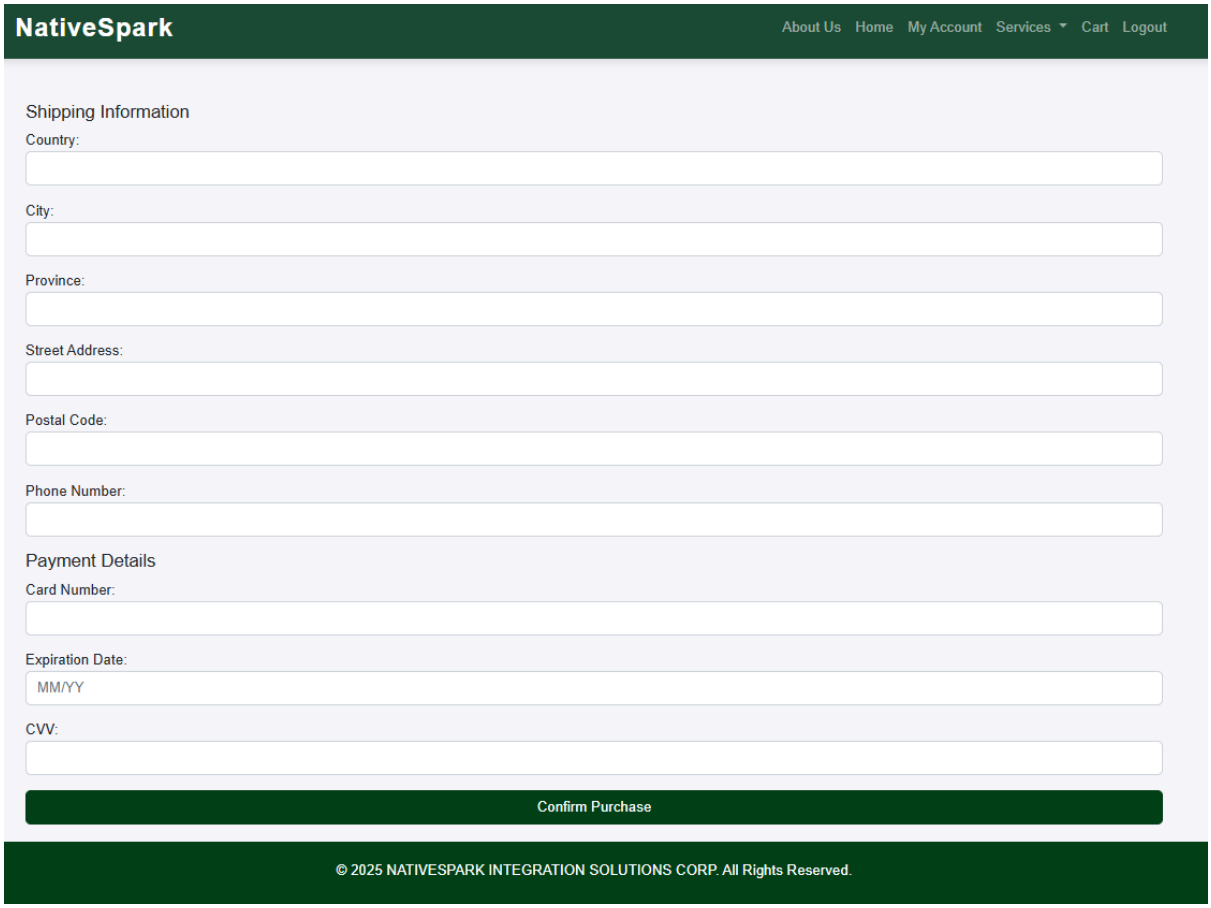
The product can be added to cart and proceeded to check out.



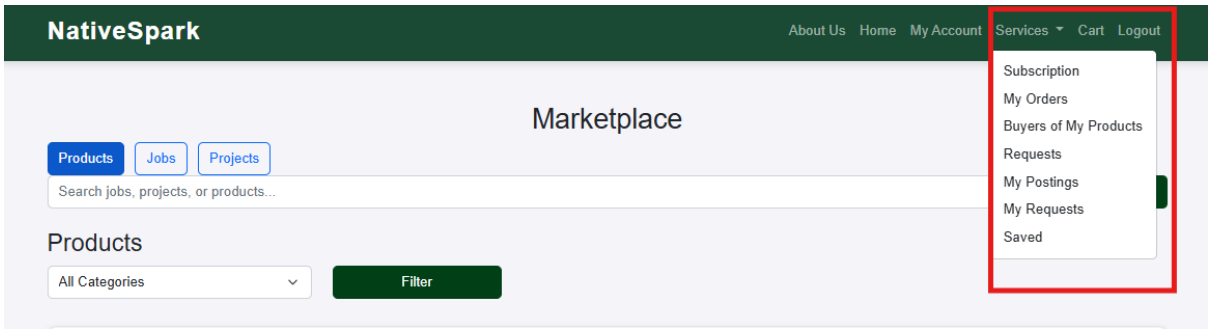
On summary page the information about seller and the product will be shown.



To place the order shipping information needed.

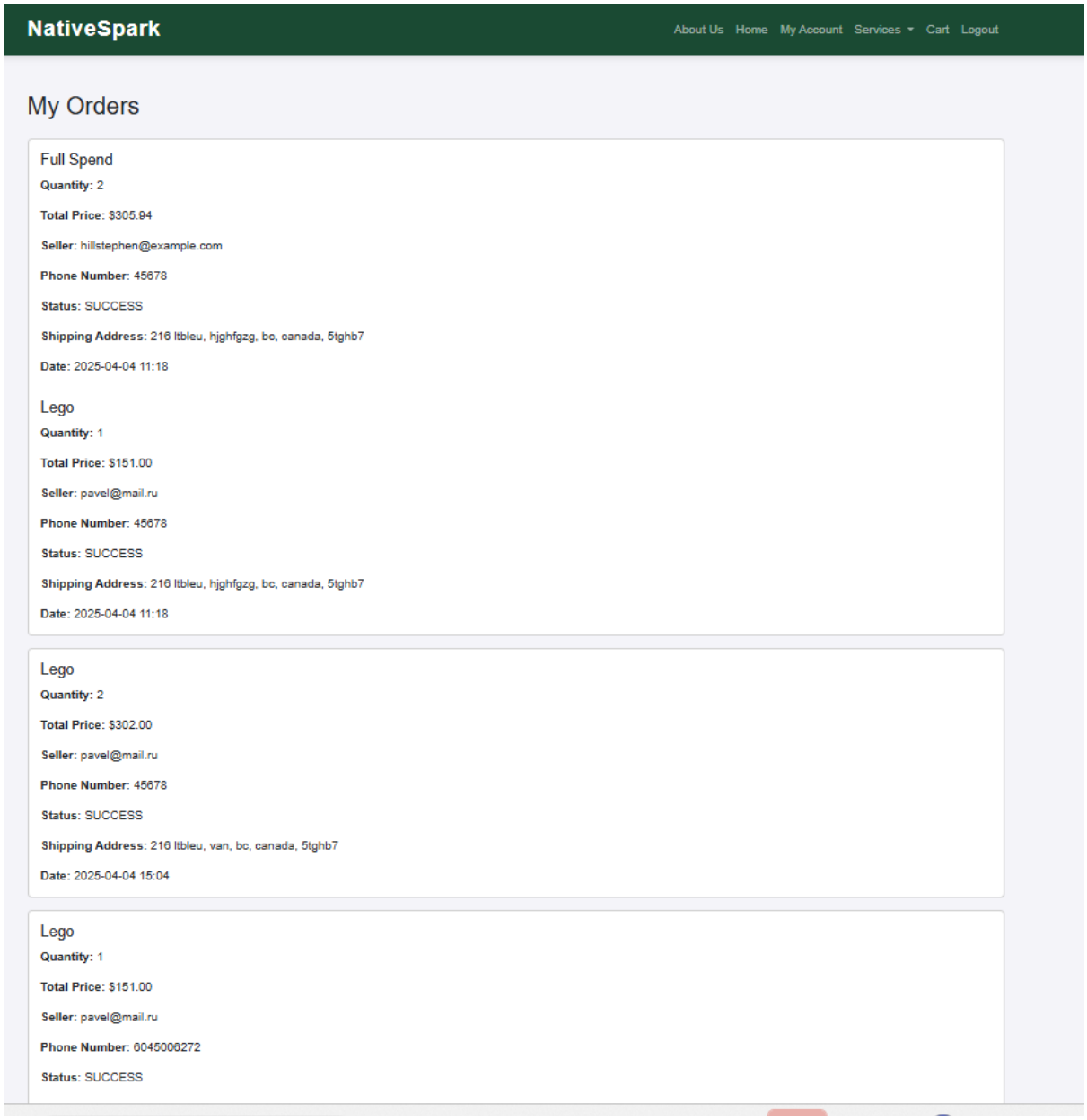


Services dropdown



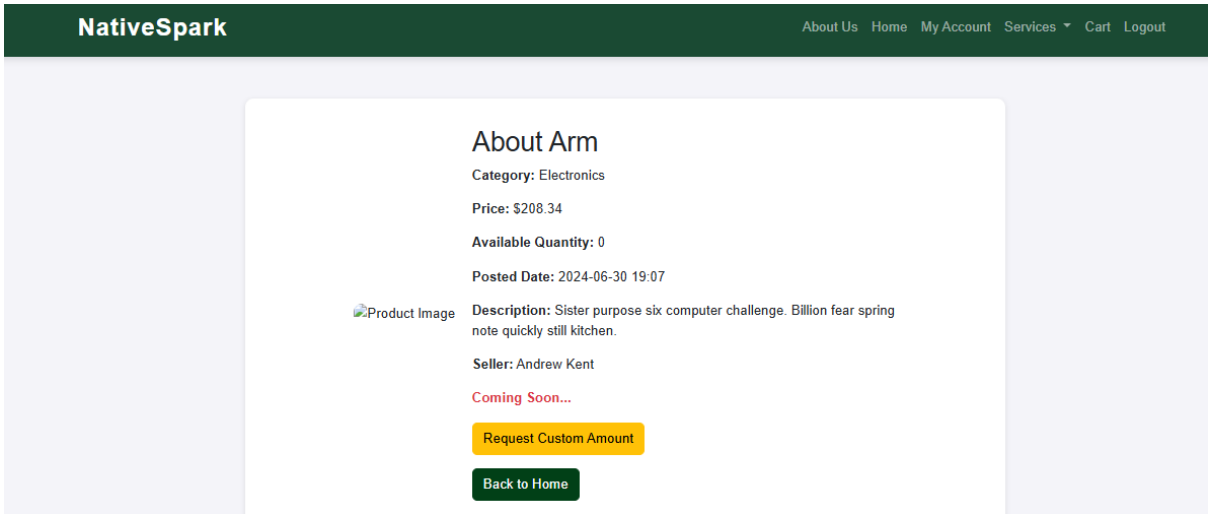
My Orders

To see the information about purchased products click My Orders

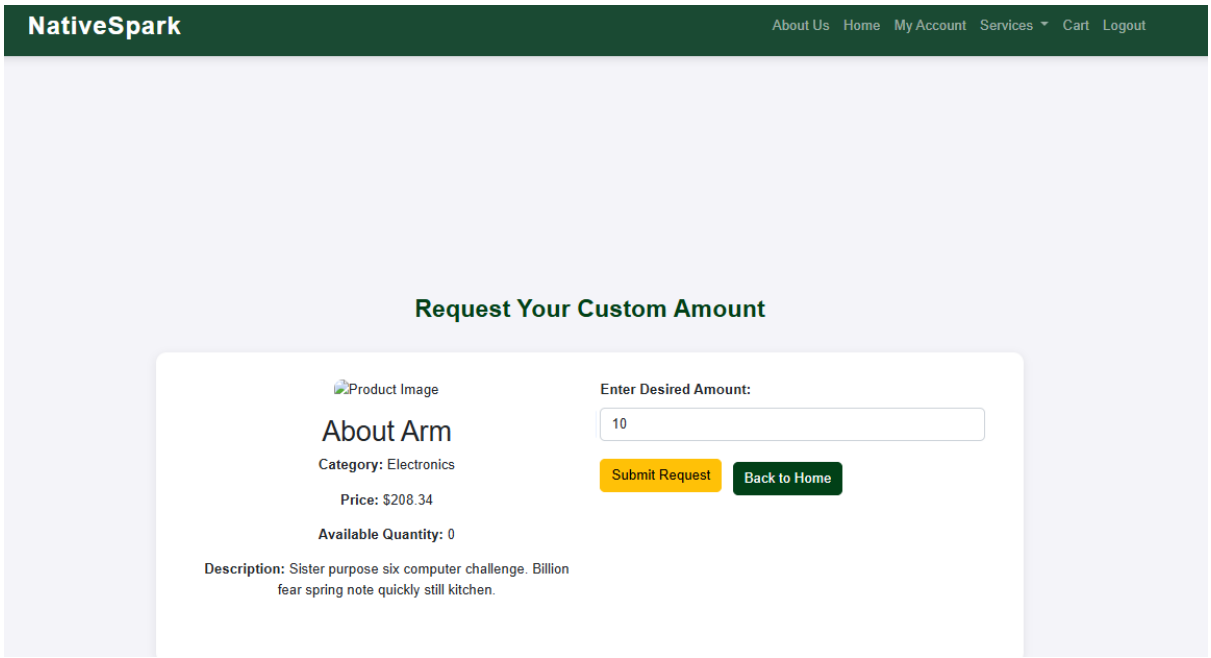


My Requests

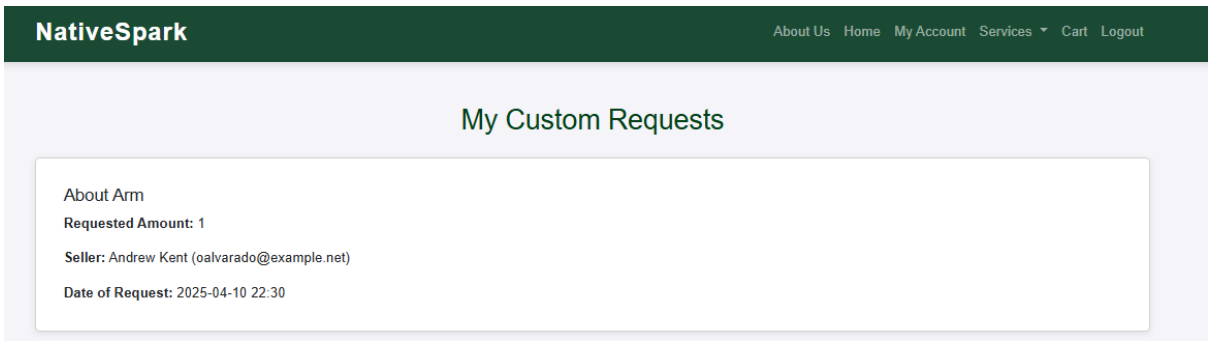
If the product is out of stock the user can request custom amount.



Enter custom amount and place the request

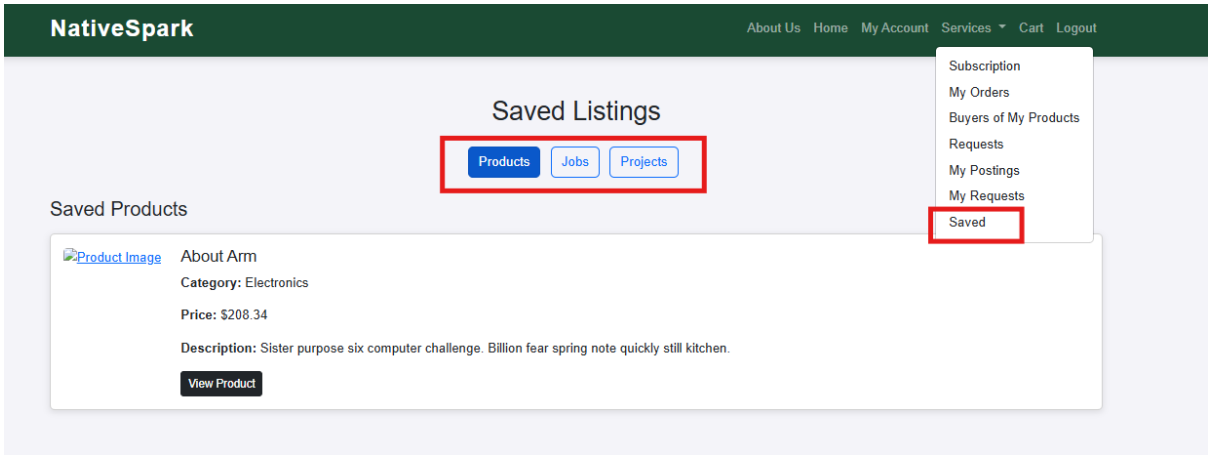


To see the requested press My requests in Services dropdown.



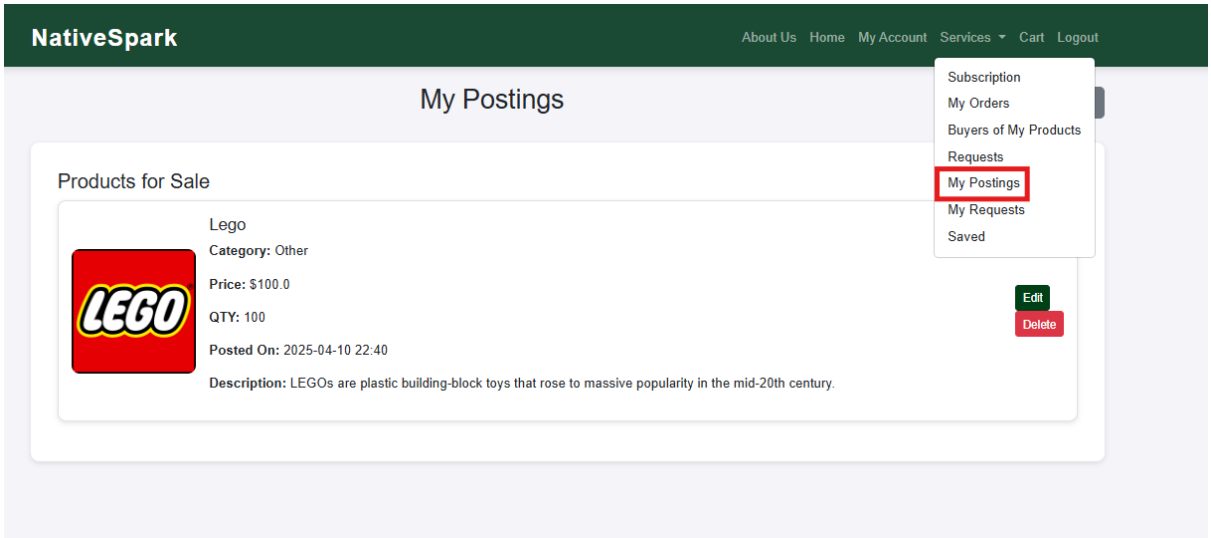
Saved

Browse saved products/jobs/products



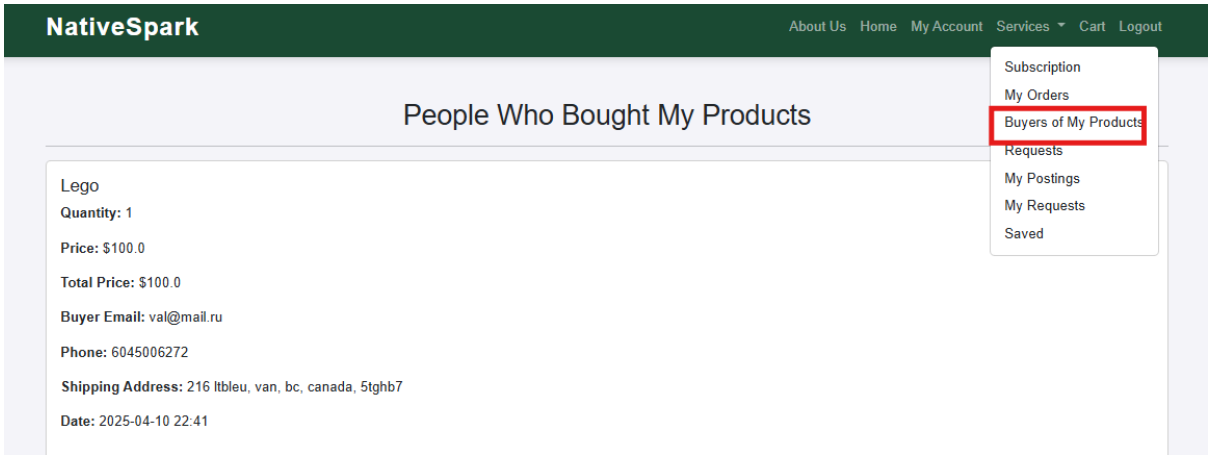
My Postings

The Entrepreneurs User can post/edit/delete products.



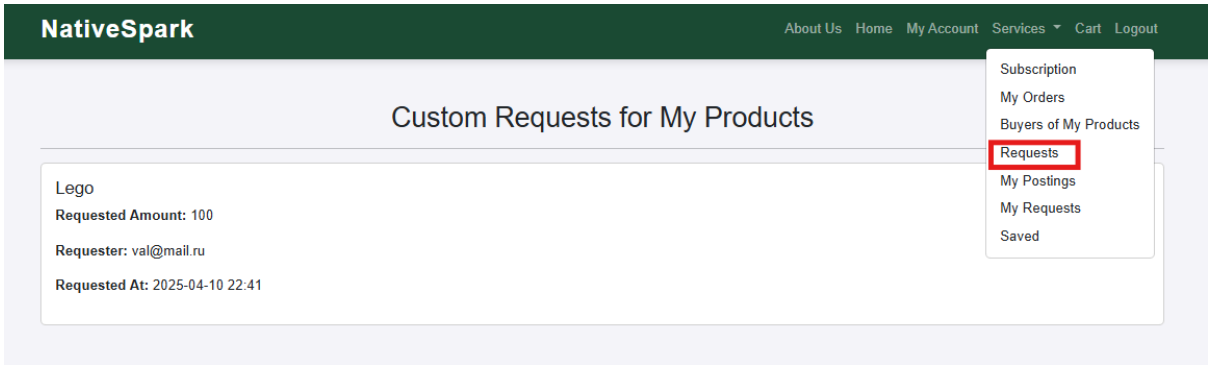
Buyers of my products

The user can see the buyers of the product for future shipping.



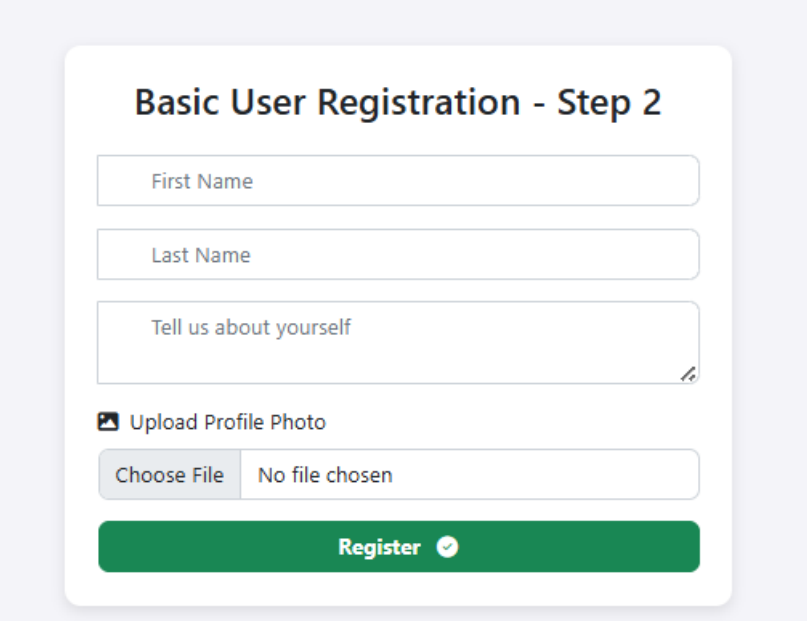
Requests

The user can see the requests of the product for future manufacturing.



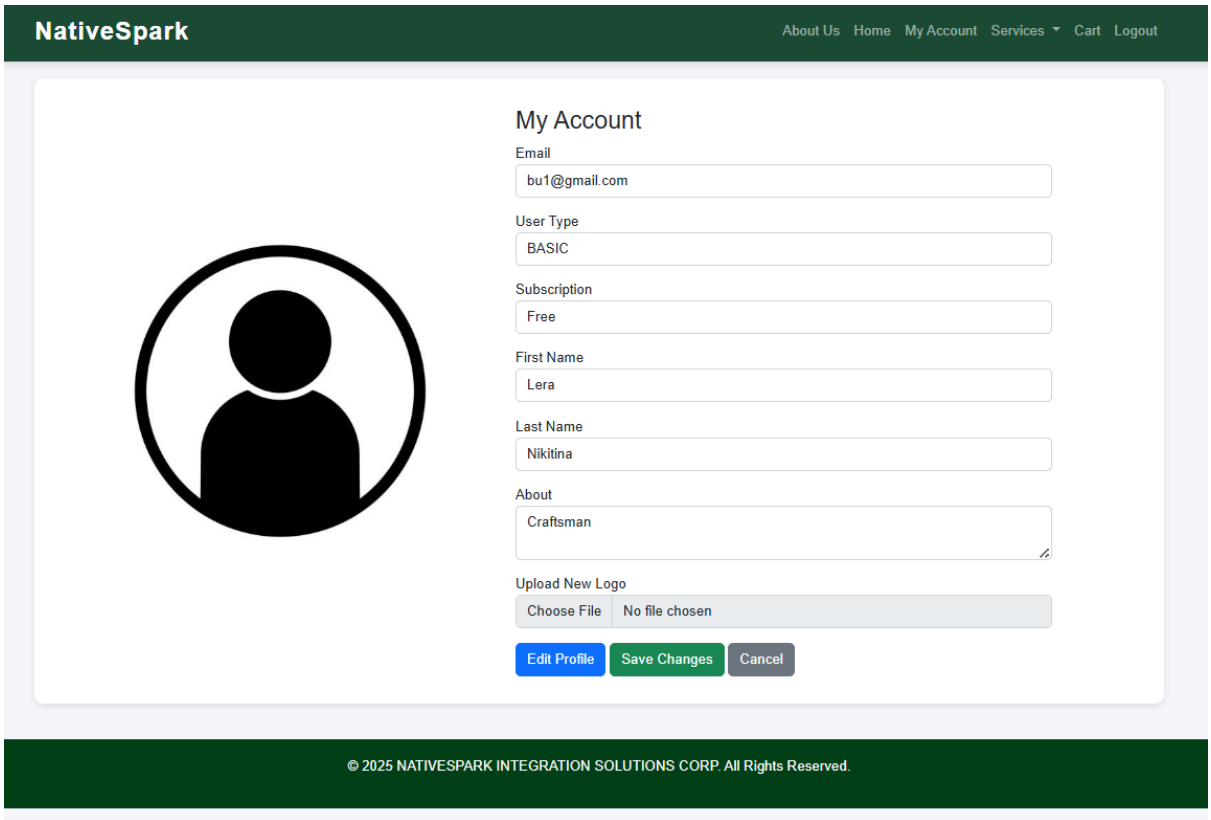
Basic User

Registration



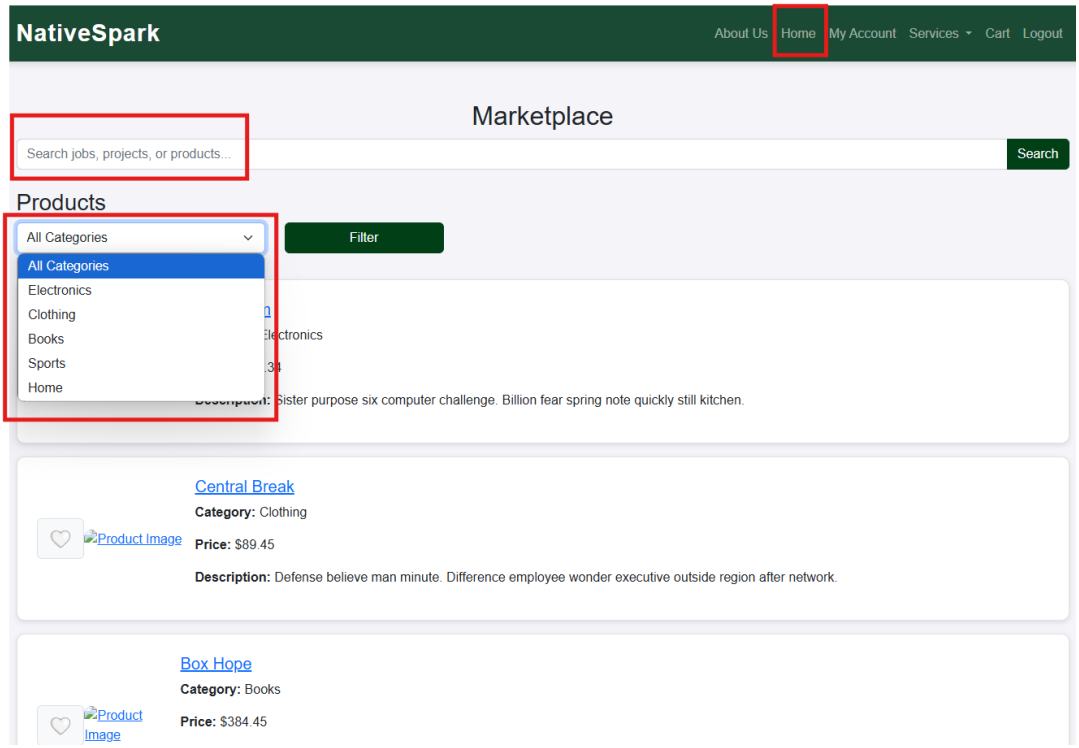
My Account

In my account you can edit information about yourself.

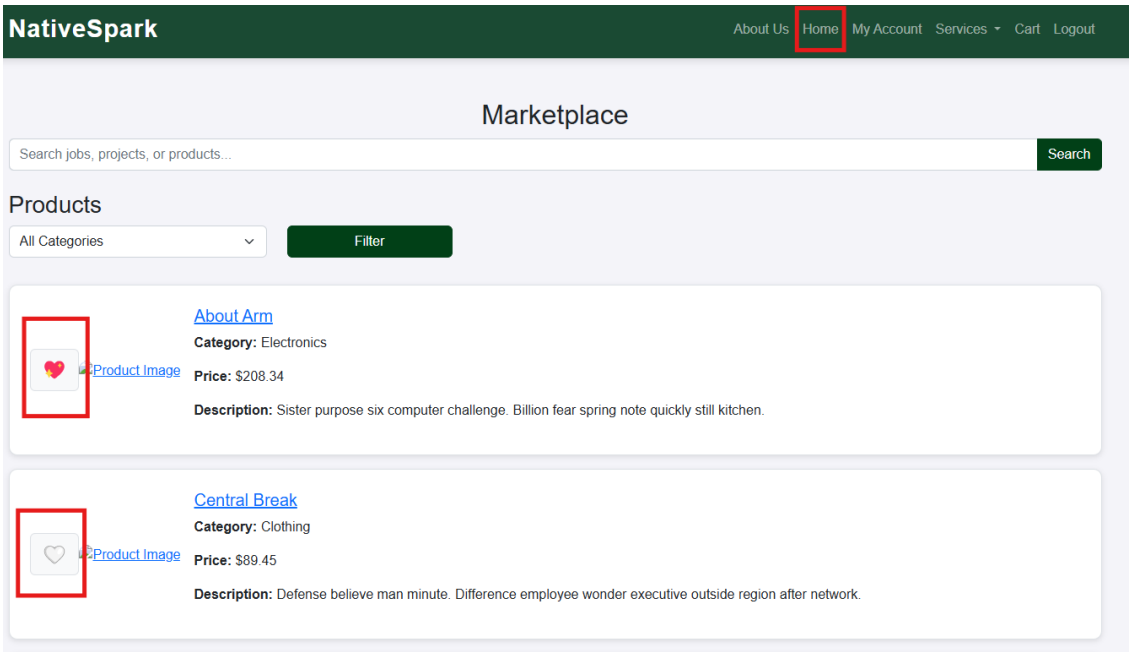


Home Page

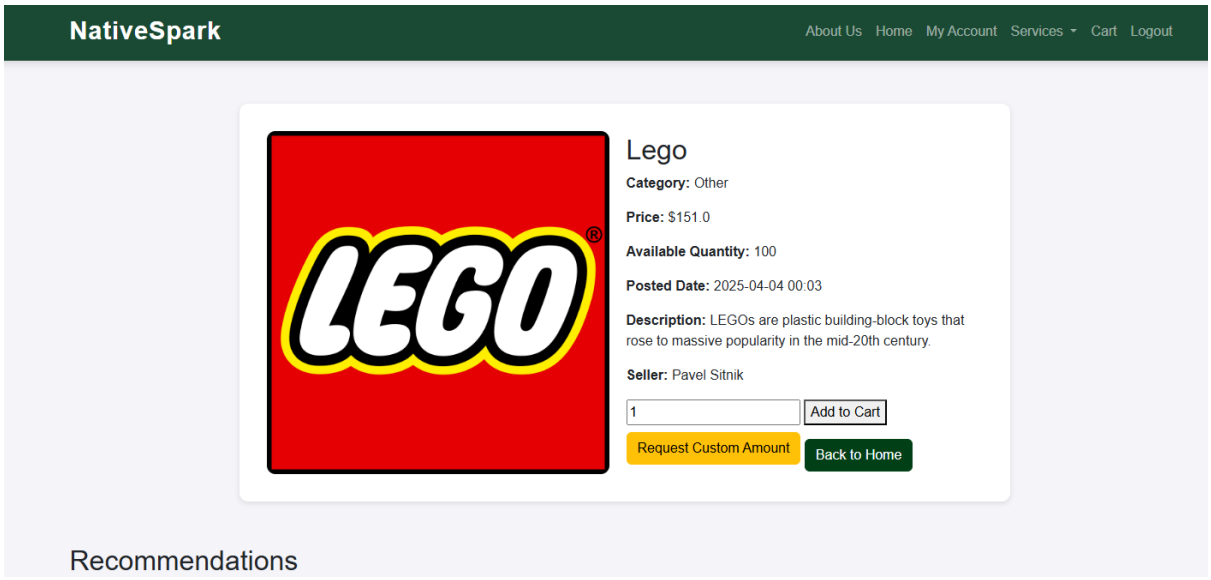
Basic users on the home page can filter products by existing categories or by keywords searched in the product description and name.



Users can save their favourite products for later purchase or set a custom amount if their desired quantity exceeds the stock available at that moment.

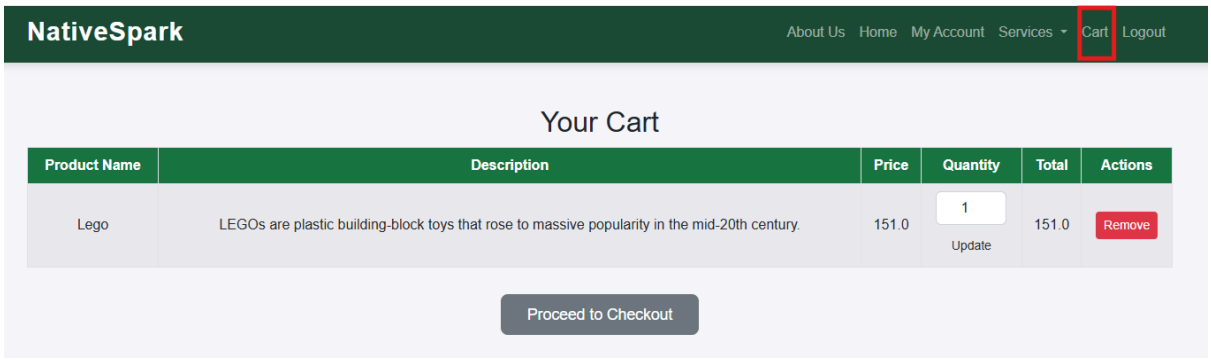


Additionally, by clicking on the product image or name, a detailed summary will open.

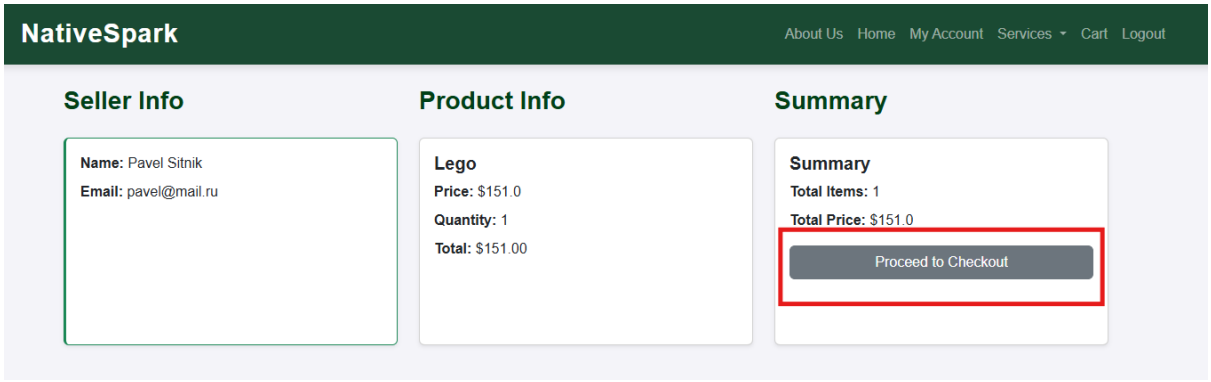


Cart

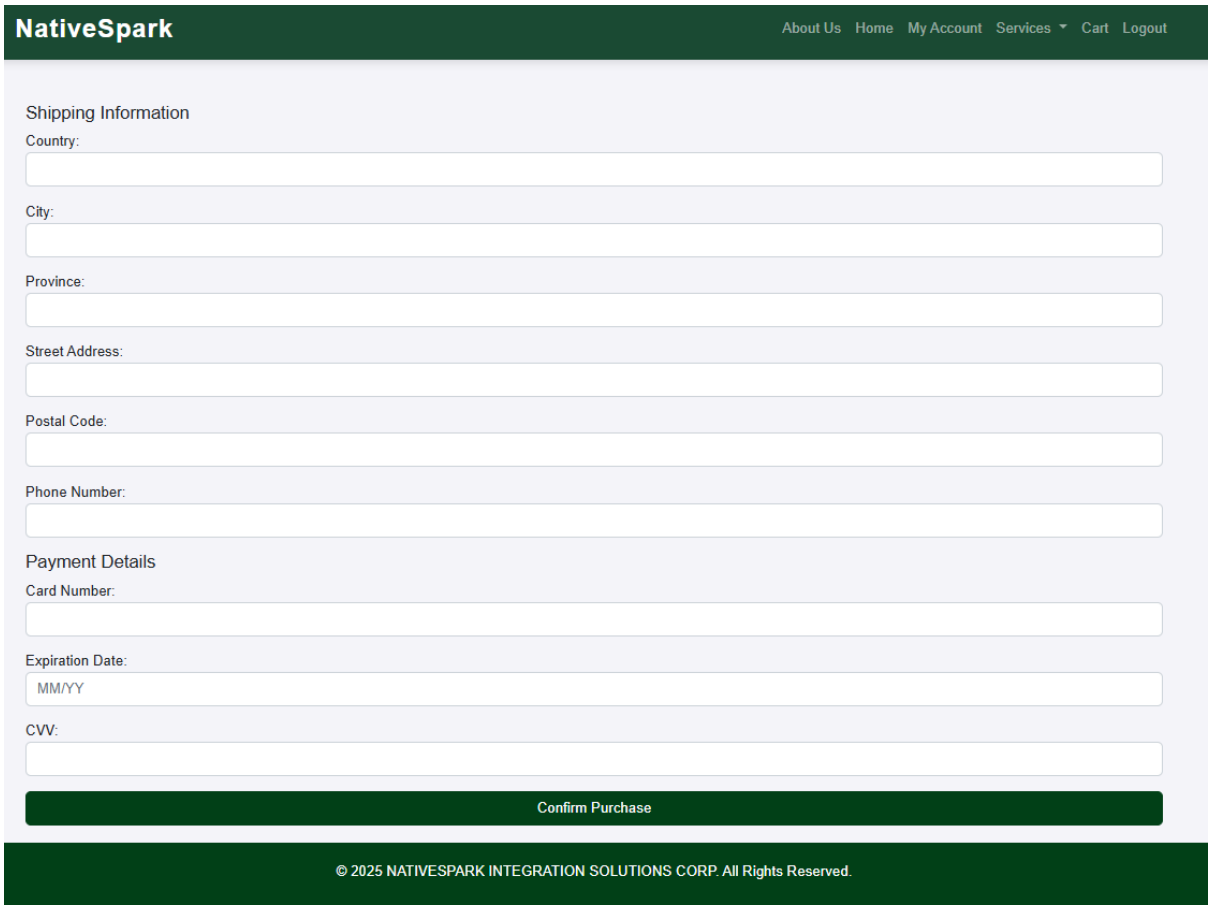
The product can be added to cart and proceeded to check out.



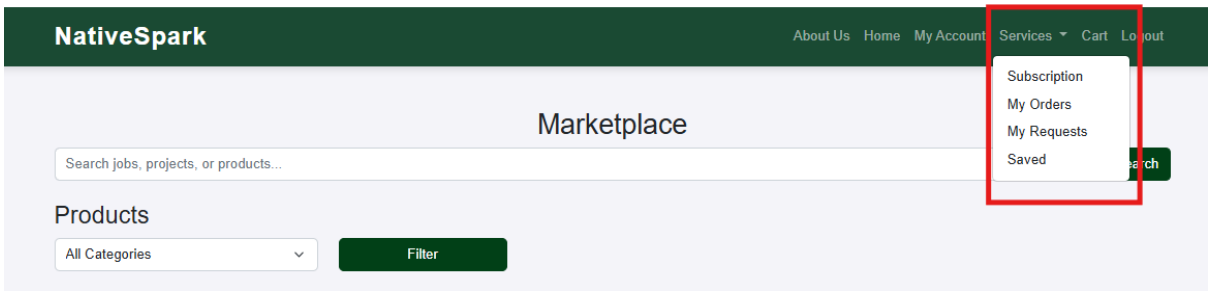
On summary page the information about seller and the product will be shown.



To place the order shipping information needed.

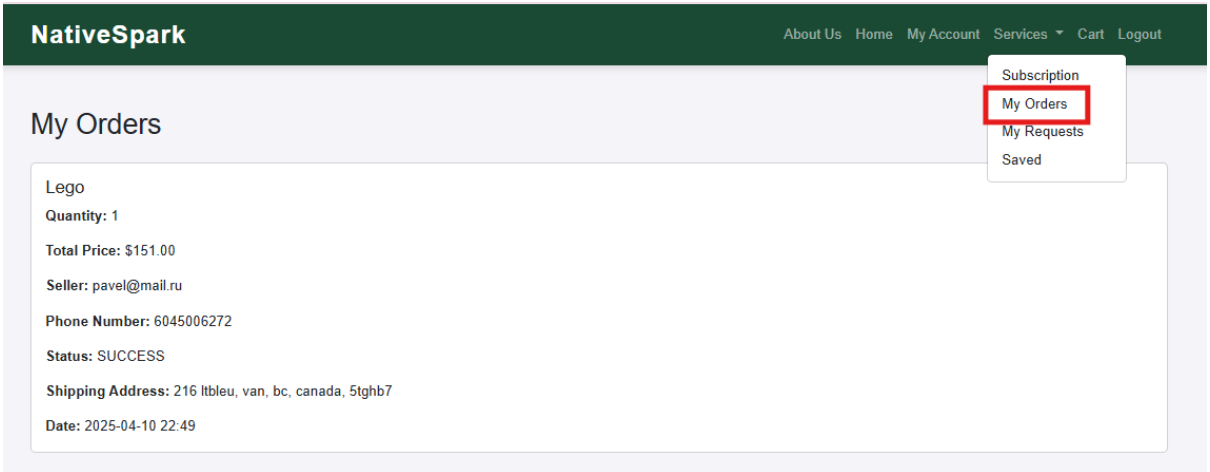


Services dropdown



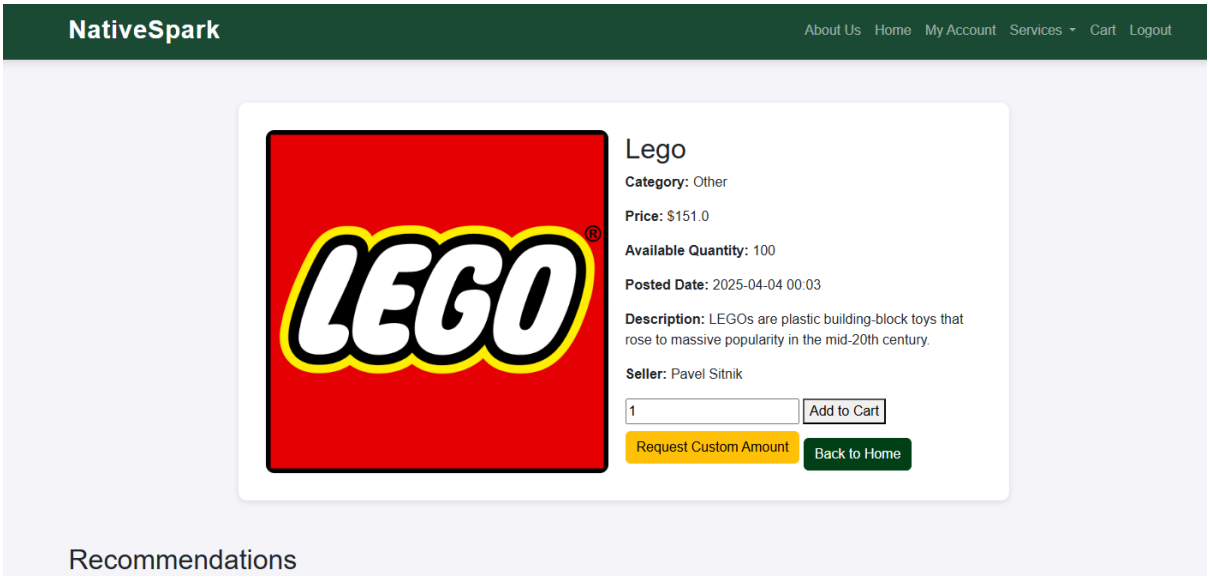
My Orders

To see the information about purchased products click My Orders

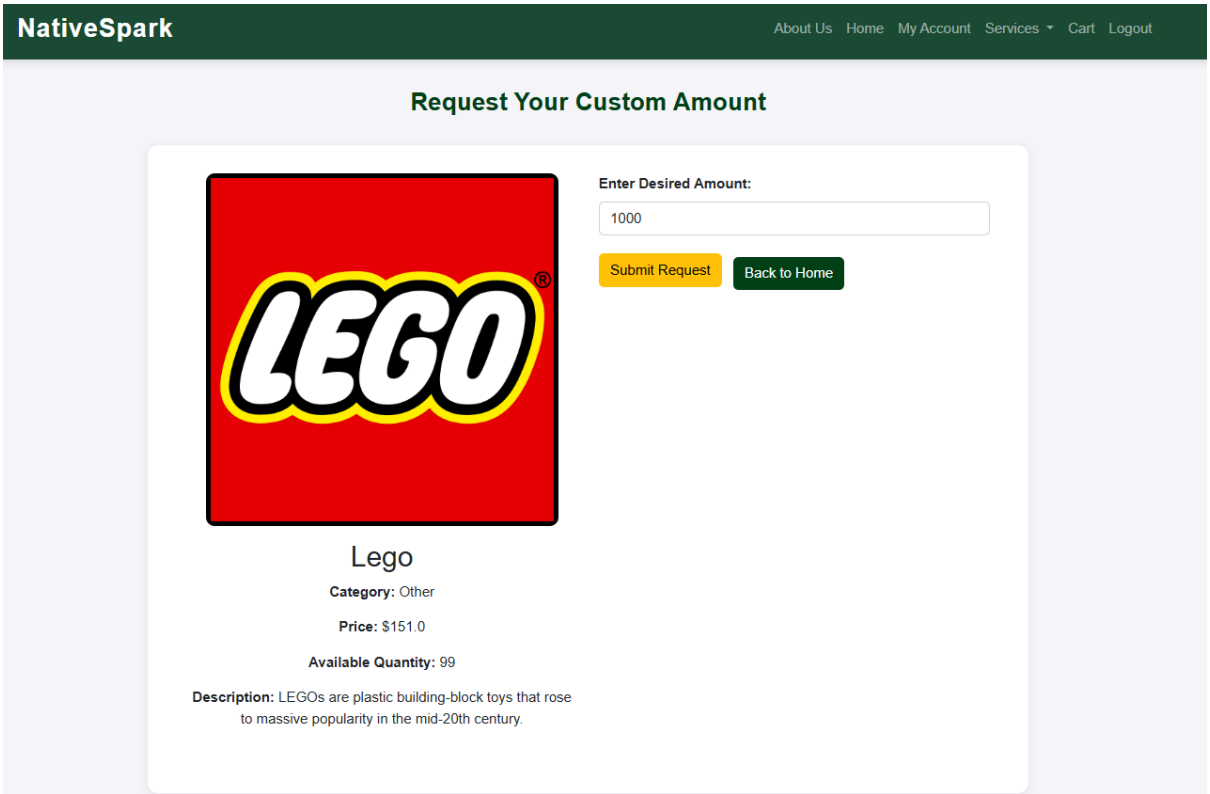


My Requests

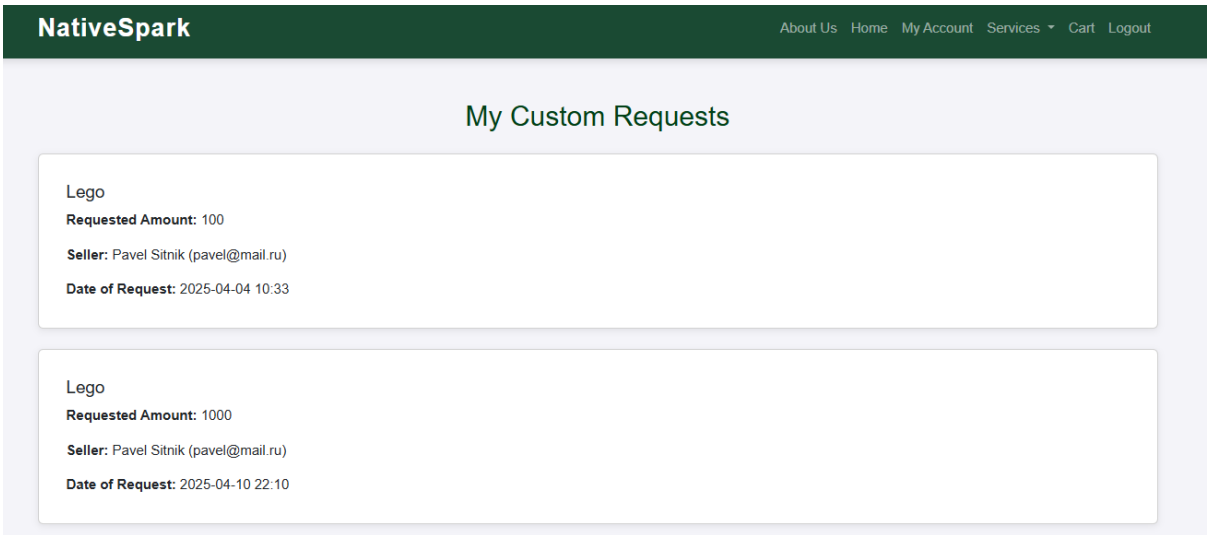
To request custom amount the user needs to submit it on product details page.



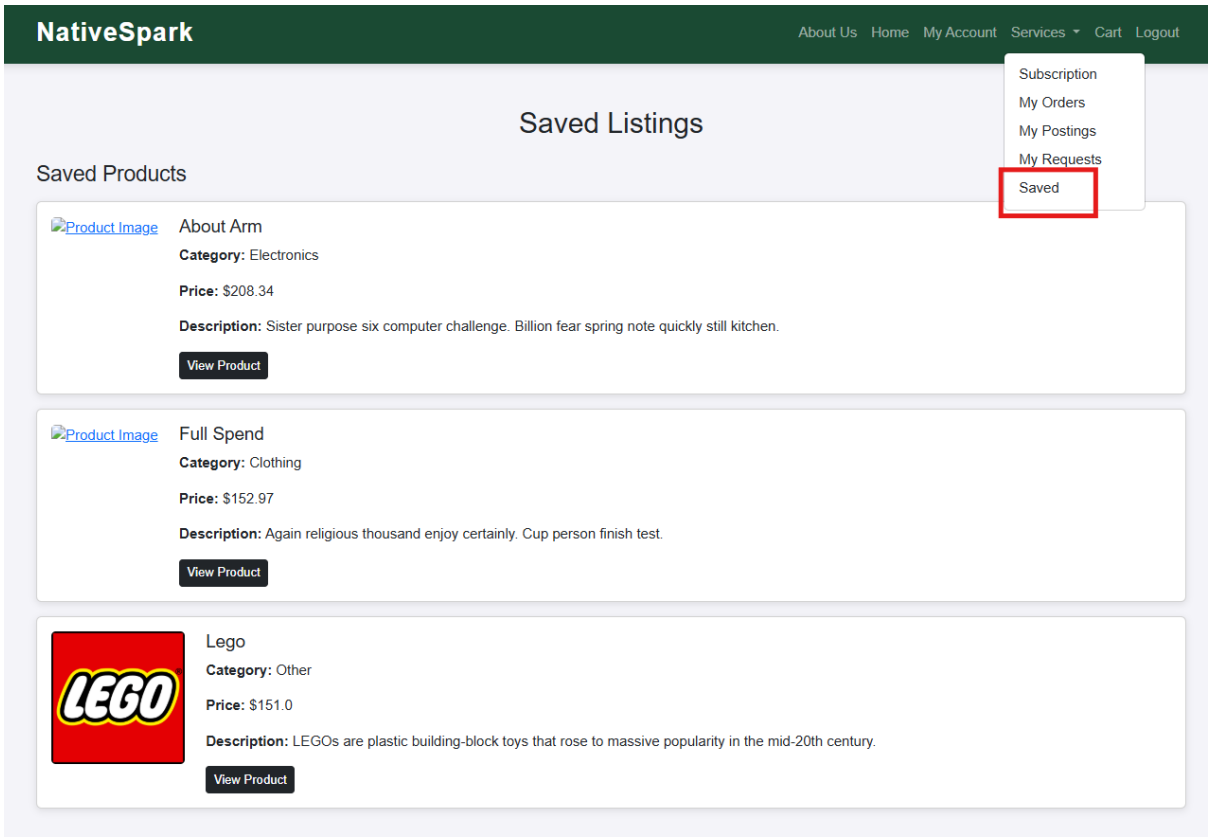
Enter custom amount and place the request



To see the requested press My requests in Services dropdown.



Saved


NativeSpark
About Us
Home
My Account
Services
Cart
Logout
Subscription
My Orders
My Postings
My Requests
Saved
Saved Listings
Saved Products
Product Image
About Arm
Category: Electronics
Price: $208.34
Description: Sister purpose six computer challenge. Billion fear spring note quickly still kitchen.
View Product
Product Image
Full Spend
Category: Clothing
Price: $152.97
Description: Again religious thousand enjoy certainly. Cup person finish test.
View Product
Lego
Category: Other
Price: $151.0
Description: LEGOs are plastic building-block toys that rose to massive popularity in the mid-20th century.
View Product