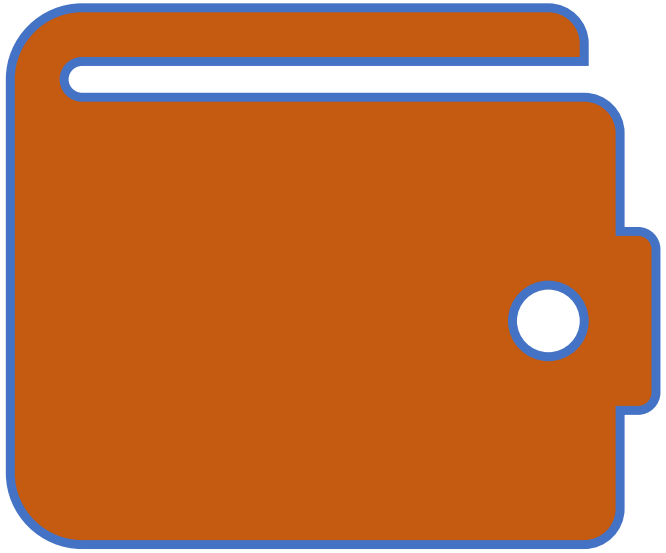
BCDE224 Server-Side Programming – PHP

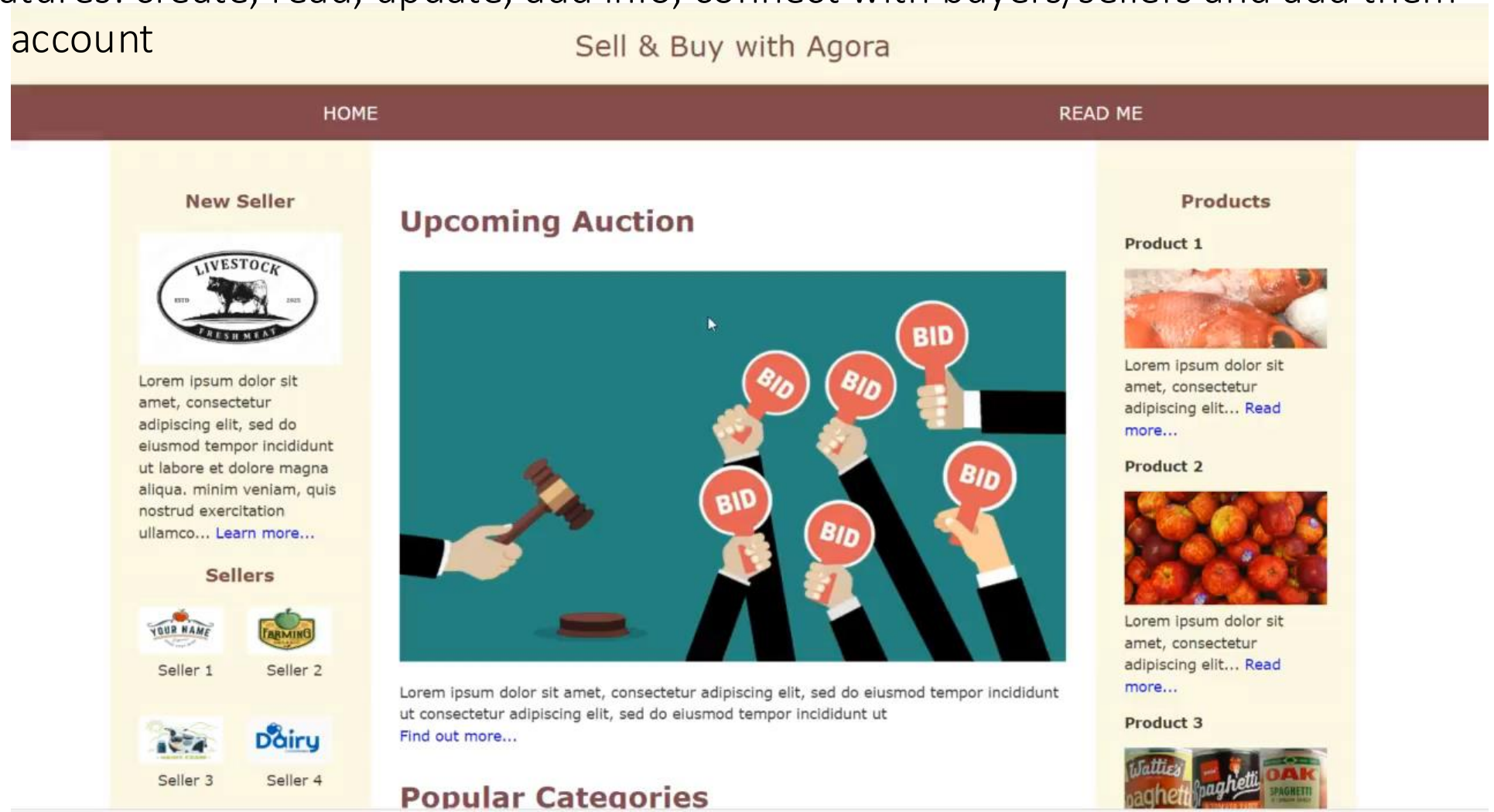
Portfolio. Prototype for e-commerce portal “Agora”



Agora e-commerce portal:
FEATURES

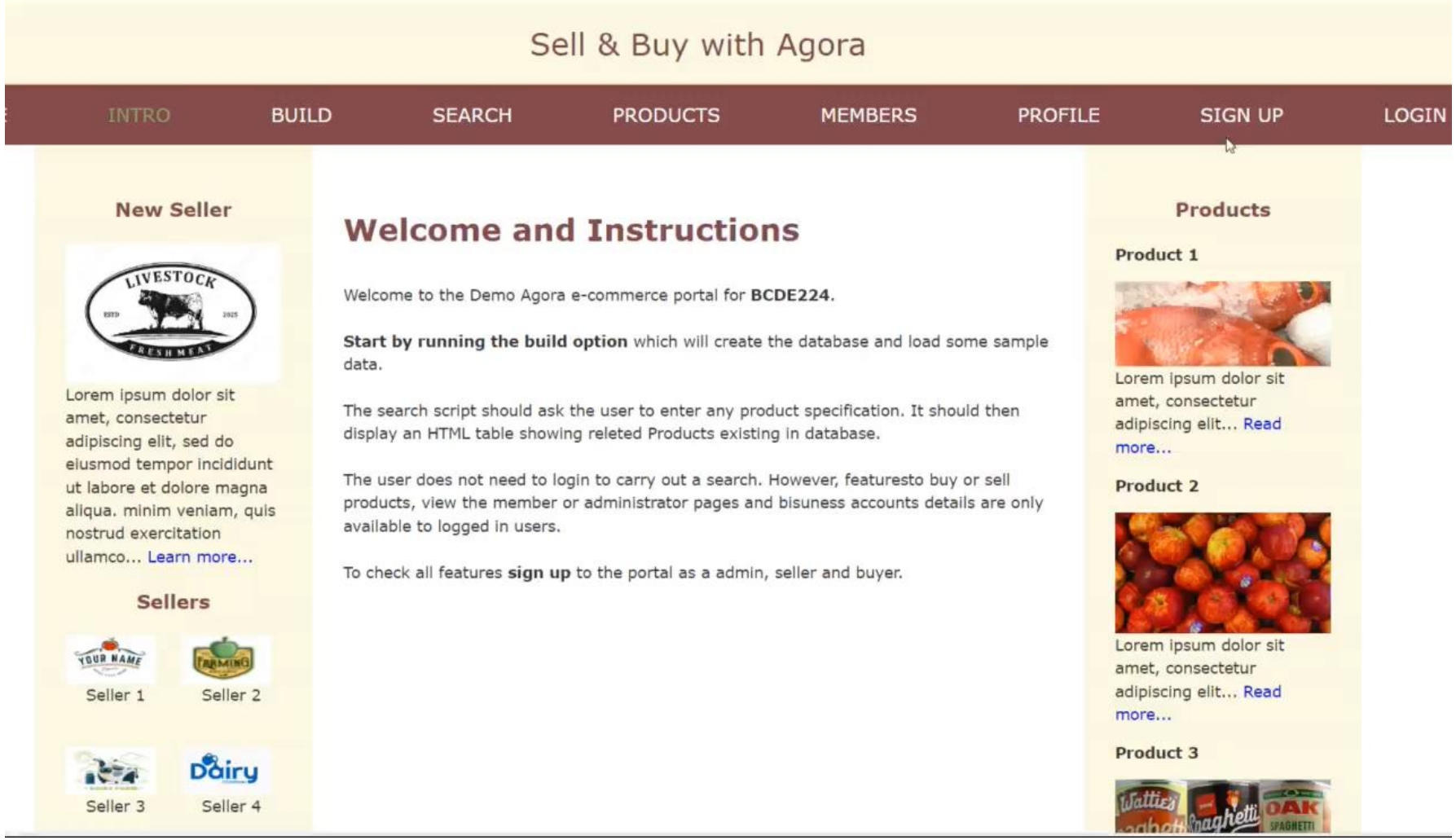
An admin of business account

Demo of features: create, read, update, add info, connect with buyers/sellers and add them to business account



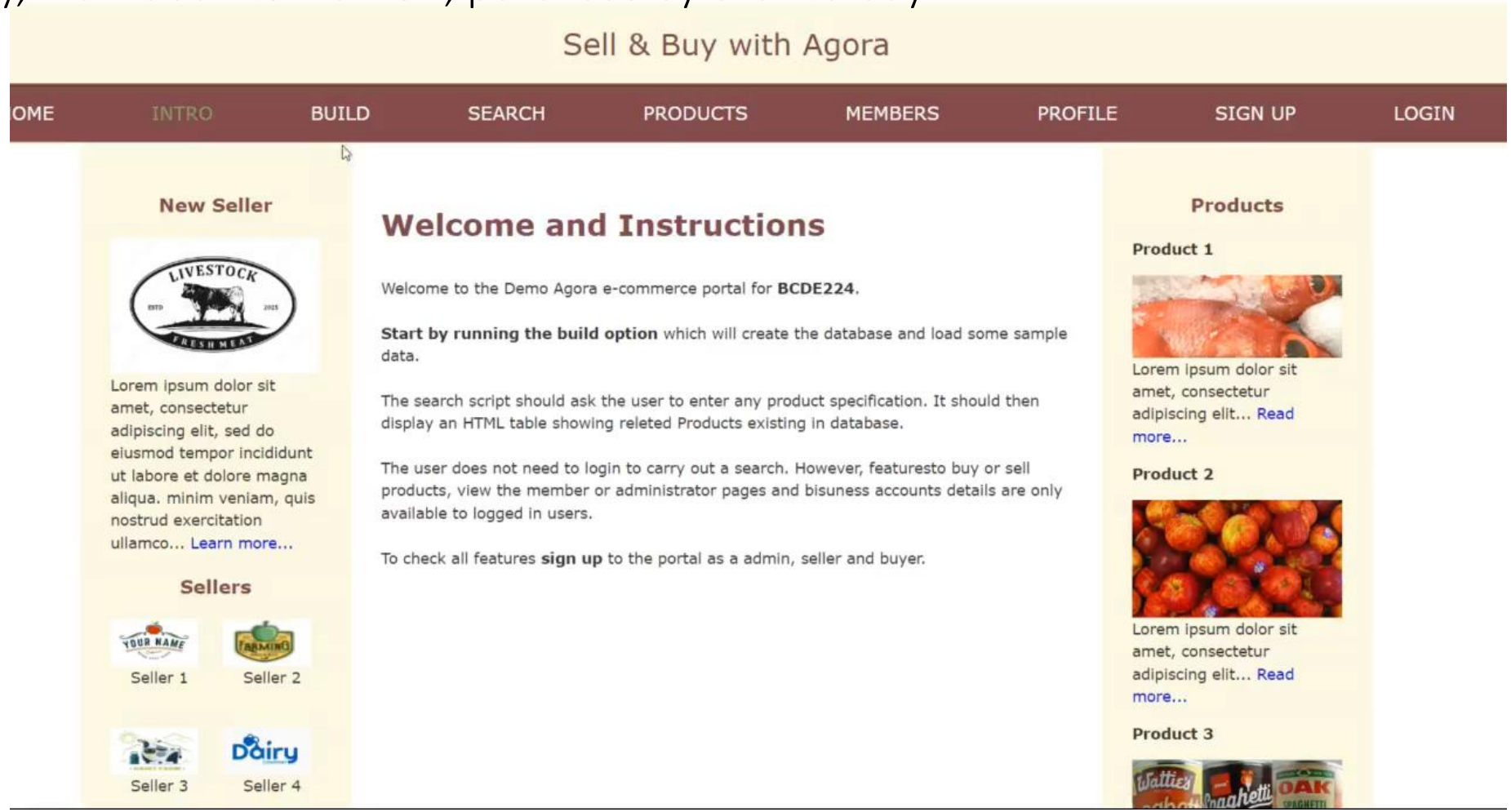
A seller account

Demo of features: create, read, update, connect to business, list a new item, view a list of seller's items



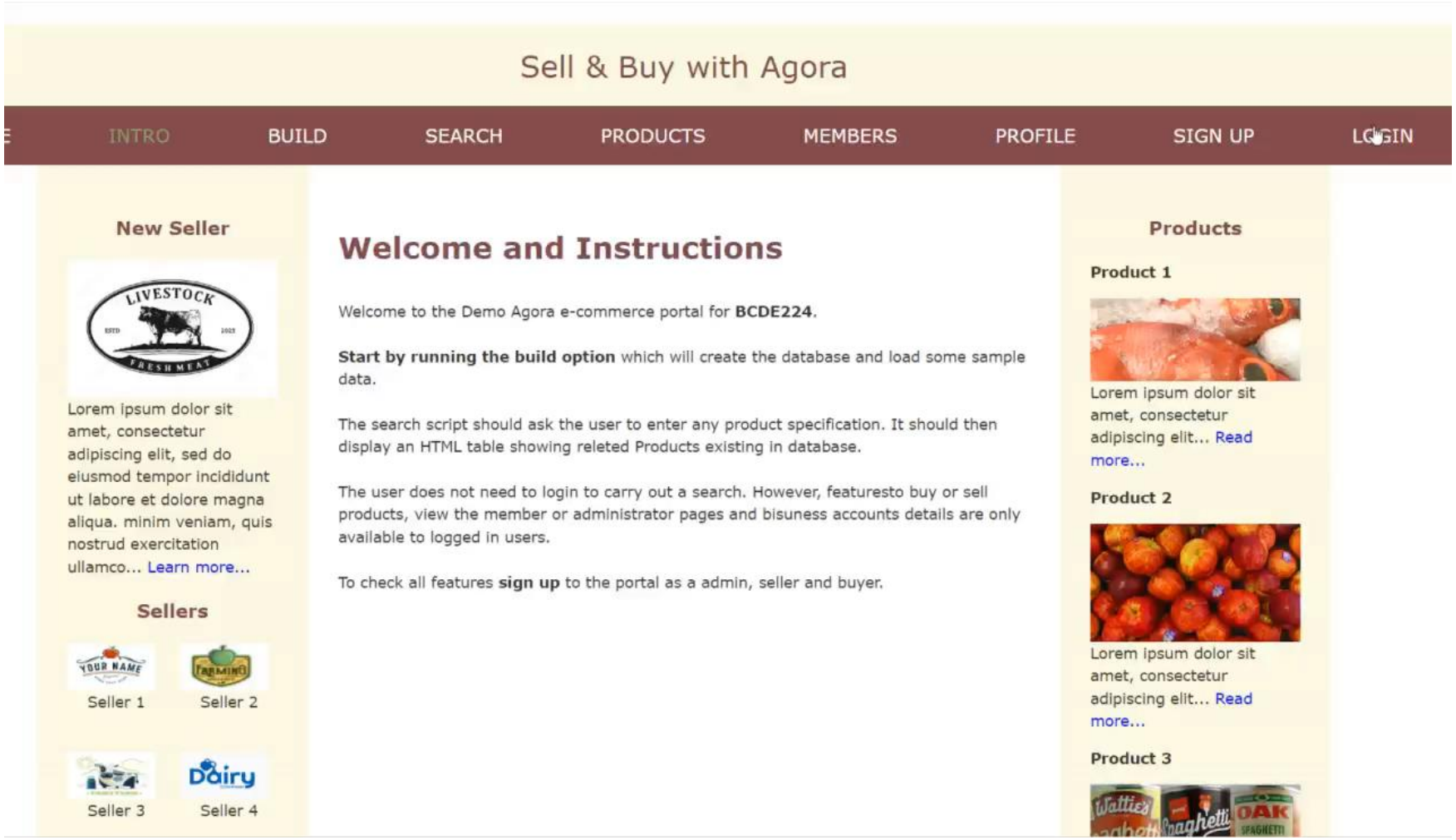
A buyer account

Demo of features: create, read, update, connect to business, list a new item, view of all items to buy, individual item's view, purchase by click to buy.



Agora e-commerce portal: SECURITY

Security: login & hashed password



Hash password script

```php
$stmt = mysqli_prepare($db->dbConn, "SELECT memberID, passwordHash FROM members WHERE login = ?");
mysqli_stmt_bind_param($stmt, "s", $u);
mysqli_stmt_execute($stmt);
$result = mysqli_stmt_get_result($stmt);

if (mysqli_num_rows($result) === 1) {
    $row = mysqli_fetch_assoc($result);
    $hash = $row['passwordHash'];
    $id = $row['memberID'];

    // Compare hashed passwords using password_verify
    if (password_verify($p, $hash)) {
        return $id;
    } else {
        if ($hash === $h) {
            return $id;
        }

        if (empty($hash) || $hash === null) {
            // Update the password hash in the database
            $updateStmt = mysqli_prepare($db->dbConn, "UPDATE members SET passwordHash = ? WHERE memberID = ?");
            mysqli_stmt_bind_param($updateStmt, "si", $h, $id);
            mysqli_stmt_execute($updateStmt);
            return $id;
        }

        if ($result->num_rows === 1) {
            // Existing code...

            if ($hash === $h) {
                return $id;
            }

            // Debugging statements
            echo "Hash Mismatch: Hash: $hash, Expected: $h"; // Or log this information instead of echoing.
            // Or use error_log("Hash Mismatch: Hash: $hash, Expected: $h");

            // Existing code...
        }
    }
```

Injection

Explanation: Injection attacks involve injecting malicious code or commands through vulnerable inputs, targeting data-driven applications. Common types include SQL injection and NoSQL injection.

Potential Attack Points: Forms, search fields, login pages, or any user input that is used to construct queries to the database.

Demo of Attack: See on Security video.

Code Defense Example: Use parameterized queries or prepared statements to sanitize user inputs before interacting with the database.

Demonstration of Protection: See on Security video

Broken Authentication

Explanation: Broken Authentication involves weak implementation of authentication and session management leading to unauthorized access.

Potential Attack Points: Weak password policies, insecure session management, or exposed credentials.

Demo of Attack: Show how a weak password policy allows brute force attacks or how sessions are not adequately managed, allowing session hijacking.

Code Defense Example: Enforce strong password policies, implement multi-factor authentication, and ensure secure session handling.

Demonstration of Protection: See script and Security vidoe

Sensitive Data Exposure

Explanation: Sensitive Data Exposure occurs when sensitive information is exposed due to weak encryption or insecure data handling.

Potential Attack Points: Storage of sensitive data, insecure transmission, or weak encryption methods.

Demo of Attack: Sensitive data like credit card details is stored in plain text or transmitted insecurely.

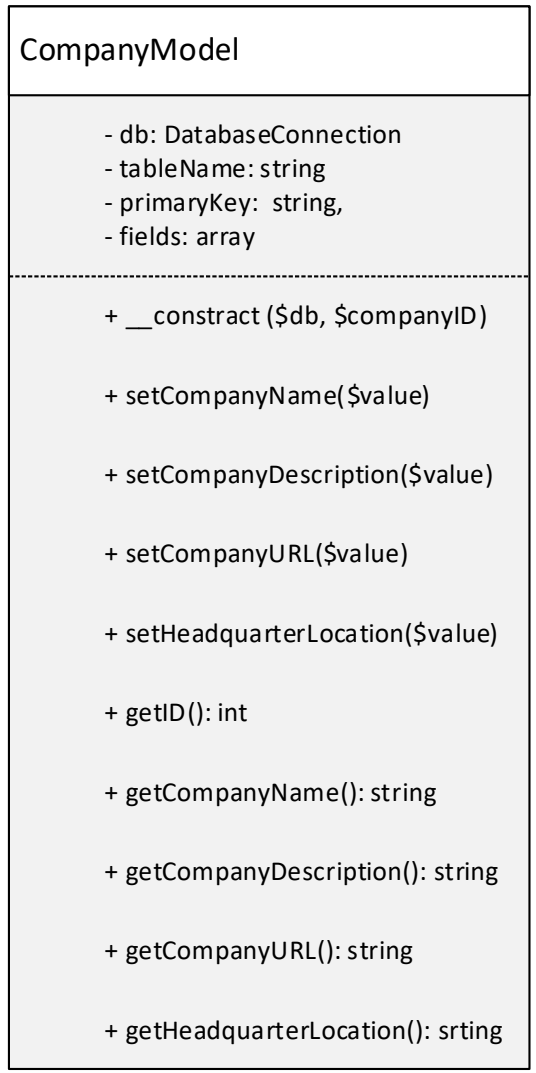
Code Defense Example: Encrypt sensitive data, use secure channels (HTTPS), implement strong encryption algorithms, and ensure secure storage.

Demonstration of Protection: Attempt to access sensitive data in a secured version, where encryption and secure transmission prevent unauthorized access. Password hashed. See script and video.

Agora e-commerce portal:
SOLID

company.php

```php
class CompanyModel {
    private $db;
    private $tableName = 'companies';
    private $primaryKey = 'companyID';
    private $fields = [
        'companyName',
        'companyDescription',
        'companyURL',
        'headquartersLocation'
    ];

    public function __construct($db, $companyID) {
        $this->db = $db;
        parent::__construct($this->db, $this->tableName);
        parent::defineKey($this->primaryKey, $companyID);
        foreach ($this->fields as $field) {
            parent::defineField($field);
        }

        if ($companyID != null) {
            parent::load();
        }
    }

    public function setCompanyName($value) {
        parent::setValue('companyName', $value);
    }

    public function setCompanyDescription($value) {
        parent::setValue('companyDescription', $value);
    }

    public function setCompanyURL($value) {
        parent::setValue('companyURL', $value);
    }

    public function setHeadquartersLocation($value) {
        parent::setValue('headquartersLocation', $value);
    }

    public function getID() {
        return parent::getID();
    }

    public function getCompanyName() {
        return parent::getValue('companyName');
    }

    public function getCompanyDescription() {
        return parent::getValue('companyDescription');
    }

    public function getCompanyURL() {
        return parent::getValue('companyURL');
    }

    public function getHeadquartersLocation() {
        return parent::getValue('headquartersLocation');
    }
}
```



SOLID

Single Responsibility Principle

Company Table

```
1 •   SELECT * FROM agora.companies;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

companyID	companyName	companyDescription	companyURL	headquartersLocation
1001	Agora			Location1
1002	NZ Mega Supplies			Location2
1003	Fox Butcher			Location3
1004	Homenick - Haley			Location4
1005	Fritsch Group			Location5
1006	AA		uploads/share.png	Location8
NULL	NULL	NULL	NULL	NULL

Single Responsibility Principle (SRP):

- Implementation in CompanyModel: The CompanyModel class focuses solely on managing company-related data and database interactions.
- Example: It handles methods for setting/getting company properties and manages CRUD operations for a company entity.

Open/Closed Principle (OCP):

- Implementation: The CompanyModel class allows for extension but not modification. It's designed to be extended through inheritance but doesn't require modification when extending.
- Example: Extending CompanyModel by creating subclasses to handle specialized types of companies without changing the existing behavior. (not applicable in this demo)

3. Liskov Substitution Principle (LSP):

- Implementation: The class CompanyModel behaves consistently with its parent class (if any) and can be substituted with instances of its parent class without affecting the program's correctness.
- Example: If CompanyModel extends a generic Model class, instances of CompanyModel can be used in place of Model without breaking the system.

Interface Segregation Principle (ISP):

- Implementation: The class provides methods for managing company data without exposing unrelated functionality. It avoids implementing unnecessary methods.
- Example: CompanyModel defines only methods related to managing company properties and interactions with the database, segregating its interface from irrelevant methods.

Dependency Inversion Principle (DIP):

- Implementation: CompanyModel depends on abstractions rather than concrete implementations, allowing for easier changes in database strategies.
- Example: The class uses dependency injection to receive a database connection or repository instance, enabling flexibility in switching to different database implementations.

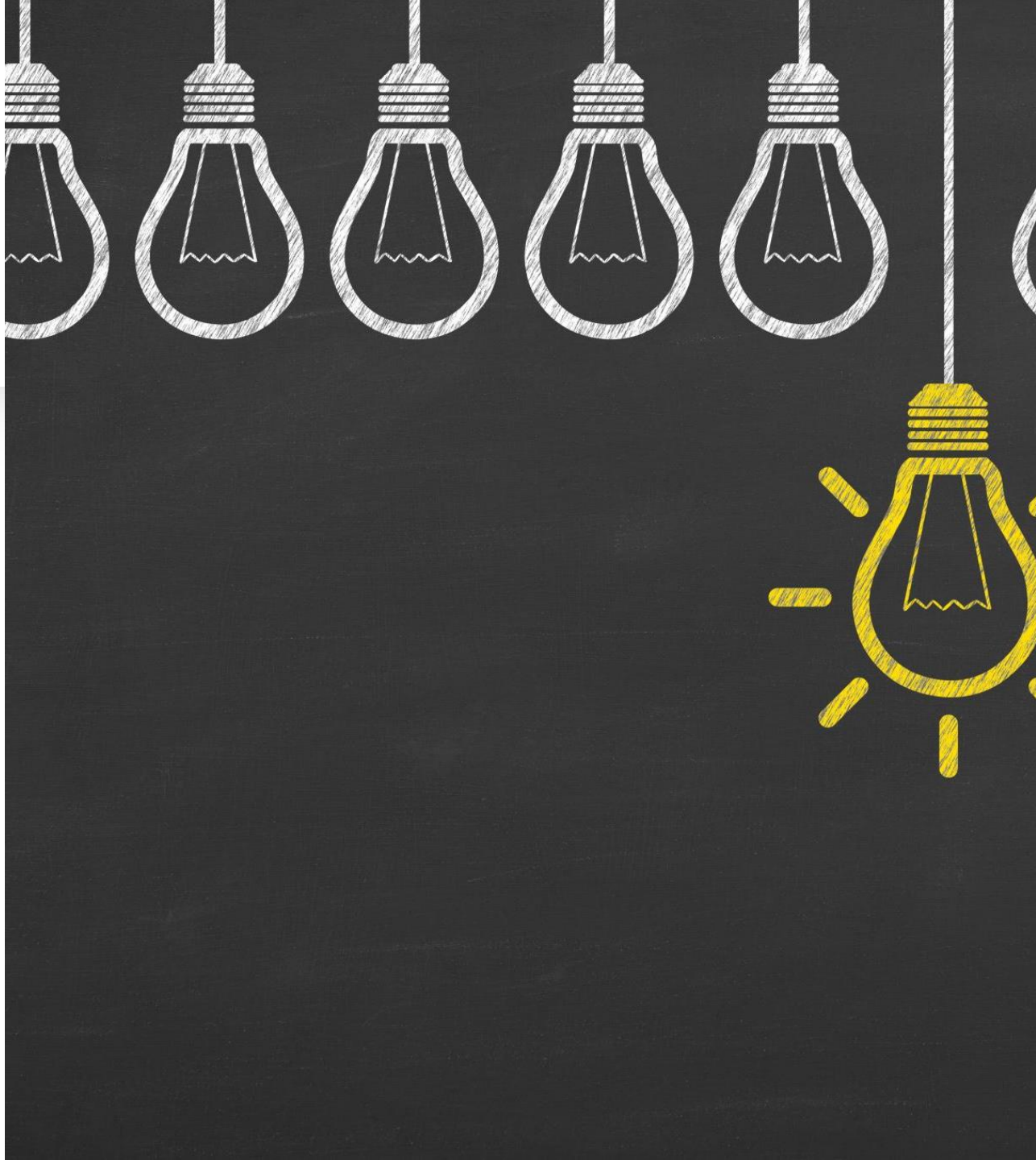
SOLID

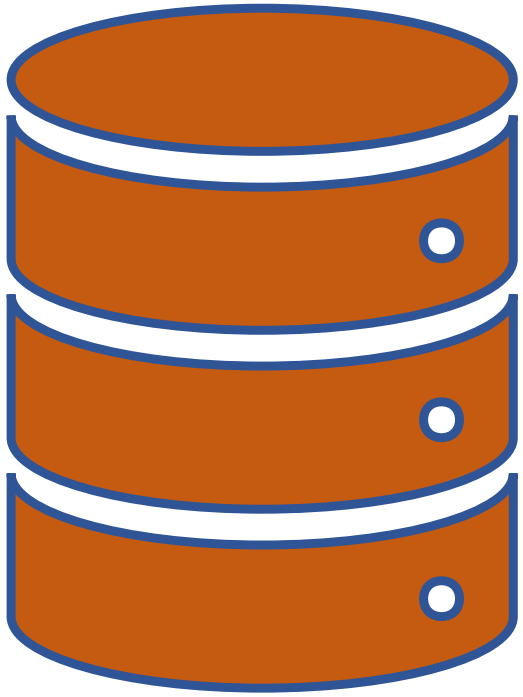
Single Responsibility Principle

- **Compliance with the Single Responsibility Principle (SRP):**
  - Responsibility: CompanyModel focuses on managing data related to a company entity, handling its properties and database interactions. It doesn't manage multiple concerns like view rendering or handling user authentication, thus adhering to SRP.
  - Separation of Concerns: The class deals specifically with company-related data operations, maintaining separation from other unrelated functionalities.

- **Possible Violations and Improvements:**
  - Incomplete Separation of Concerns: Although the CompanyModel class focuses on company-related data, it still directly interacts with the database (parent methods). To improve, it could leverage a separate class or repository responsible solely for database interactions, adhering more strictly to SRP.
  - Increased Abstraction: The class might benefit from further abstraction, such as using interfaces for database operations or creating separate classes for specific functionalities (e.g., a separate class for database CRUD operations).

- **Potential Improvements for Better SRP Adherence:**
  - Database Abstraction Layer: Implement a separate class or interface solely responsible for database interactions. The CompanyModel class can then use this abstraction layer instead of directly handling database operations.
  - Dependency Injection: Consider injecting the database connection or repository dependency into the CompanyModel class. This approach separates the concerns further and allows for easier testing and flexibility in changing the database implementation.
  - Service Layer: Introduce a service layer to handle the logic between the data model and other components of the application. This layer can further isolate business logic from data access.





Agora e-commerce portal: TECHNICAL

Tables database Agora

```
1 •  SELECT * FROM agora.orders;
```

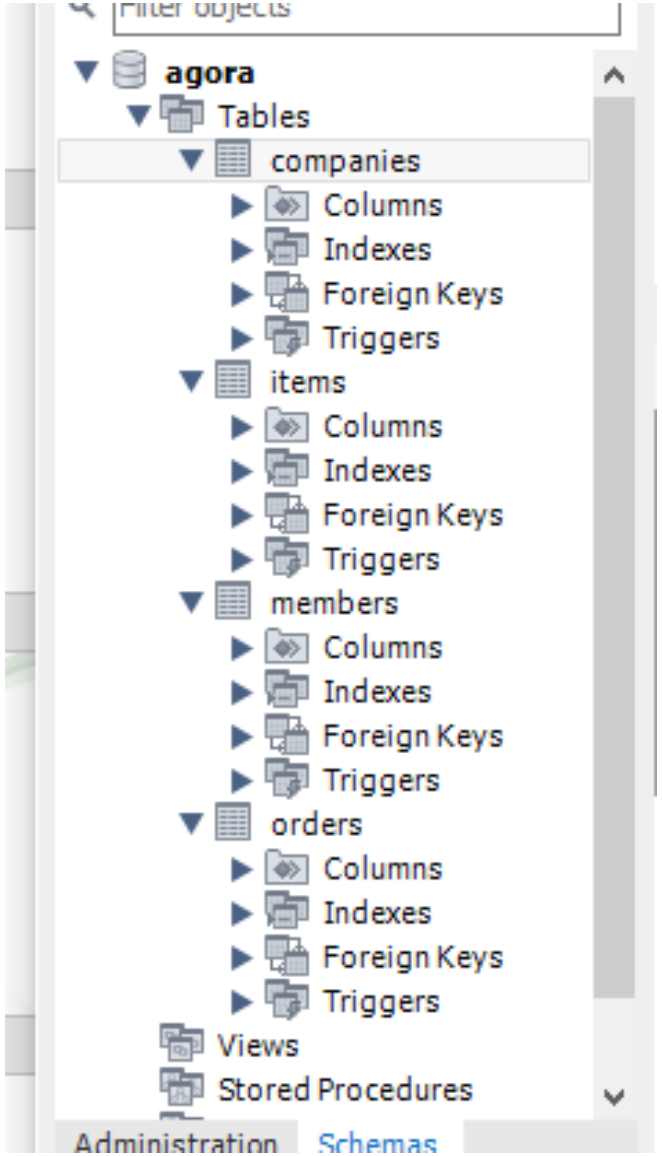
Result Grid | Filter Rows: | Edit: | Export

orderID	itemID	memberID	dateTaken	datePurchased
2000	4	103	2023-11-12	NULL
2001	3	109	2023-11-20	NULL
2002	7	109	2023-11-20	NULL
2003	6	110	2023-11-20	NULL
NULL	NULL	NULL	NULL	NULL

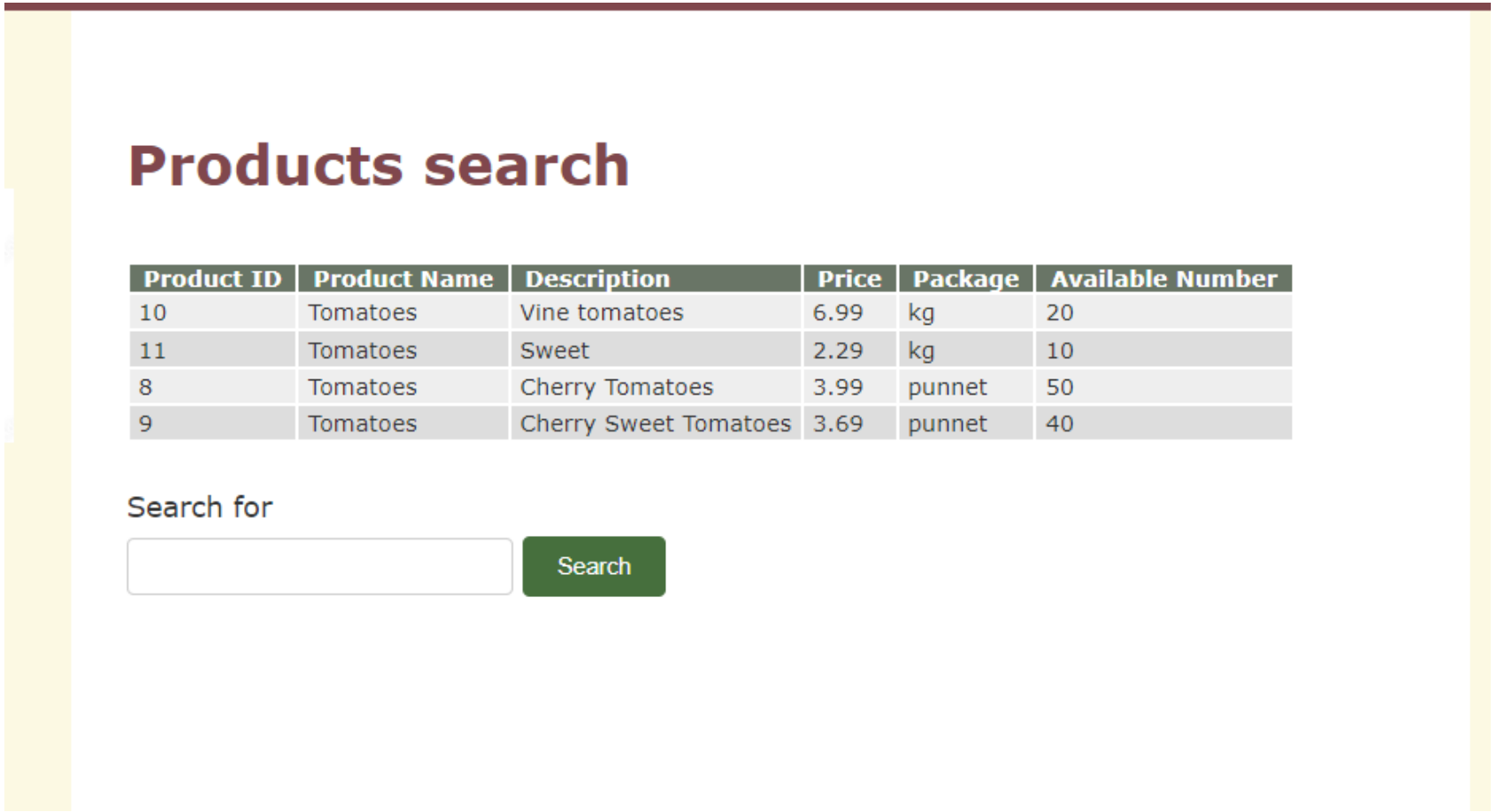
```
1 •  SELECT * FROM agora.items;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

itemID	itemName	itemCategory	itemDescription	itemPackage	itemPrice	itemImageUrl	qtyOnHand	memberID	availability
1	Pork	Meat	Pork Shoulder	kg	14.99	uploads/img1.jpg	50	105	In Stock
2	Pork	Meat	Pork Leg	kg	5.29	uploads/img2.jpg	50	105	In Stock
3	Bananas	Fruits	Yellow bananas	kg	1.39	uploads/img3.jpg	60	102	Pending
4	Strawberries	Fruits	Big size strawberries	punnet	4.99	uploads/img6.jpg	30	105	Pending
5	Strawberries	Fruits	Small size strawberries	punnet	3.99	uploads/img6.jpg	50	105	In Stock
6	Beef	Meat	Ribs	kg	10.99	uploads/img5.jpg	60	107	Pending
7	Beef	Meat	Beef Lion	kg	32.29	uploads/img5.jpg	40	107	Pending
8	Tomatoes	Vegetables	Cherry Tomatoes	punnet	3.99	uploads/cherry.jpg	50	102	In Stock
9	Tomatoes	Vegetables	Cherry Sweet Tomatoes	punnet	3.69	uploads/cherry.jpg	40	107	In Stock
10	Tomatoes	Vegetables	Vine tomatoes	kg	6.99	uploads/cherry.jpg	20	107	In Stock
11	Tomatoes	Fruit&Veges	Sweet	kg	2.29	uploads/cherry.jpg	10	111	In stock
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL



Simple Query: search for product by name



Products search

Product ID	Product Name	Description	Price	Package	Available Number
10	Tomatoes	Vine tomatoes	6.99	kg	20
11	Tomatoes	Sweet	2.29	kg	10
8	Tomatoes	Cherry Tomatoes	3.99	punnet	50
9	Tomatoes	Cherry Sweet Tomatoes	3.69	punnet	40

Search for

Search

Complex Query: display product by Company

Products in stock

Product ID	Product Name	Product Description	Package	Price	In Stock	member ID	Company Name	Availability
3	Bananas	Yellow bananas	kg	1.39	60	102	NZ Mega Supplies	In Stock
6	Beef	Ribs	kg	10.99	60	107	Fritsch Group	In Stock
7	Beef	Beef Lion	kg	32.29	40	107	Fritsch Group	In Stock
1	Pork	Pork Shoulder	kg	14.99	50	105	NZ Mega Supplies	In Stock
2	Pork	Pork Leg	kg	5.29	50	105	NZ Mega Supplies	In Stock
4	Strawberries	Big size strawberries	punnet	4.99	30	105	NZ Mega Supplies	In Stock
5	Strawberries	Small size strawberries	punnet	3.99	50	105	NZ Mega Supplies	In Stock
8	Tomatoes	Cherry Tomatoes	punnet	3.99	50	102	NZ Mega Supplies	In Stock
9	Tomatoes	Cherry Sweet Tomatoes	punnet	3.69	40	107	Fritsch Group	In Stock
10	Tomatoes	Vine tomatoes	kg	6.99	20	107	Fritsch Group	In Stock
11	Tomatoes	Sweet	kg	2.29	10	111	NZ Mega Supplies	In Stock

Buy a product

LESSONS LEARNED: I did it!!!

Exploring Agora: Developing an E-commerce Platform with PHP

Unistructural Level:

- Embarking on the development of Agora, an e-commerce portal built using PHP, began with a clear vision: to create a seamless marketplace connecting buyers and sellers. PHP, known for its server-side scripting capabilities, formed the backbone of this project, facilitating dynamic content generation and database interactions.

Multistructural Level:

- The development process delved into the intricacies of PHP's integration with HTML, CSS, and JavaScript to shape Agora's user interface. Leveraging PHP, we seamlessly handled form submissions, processed user inputs, and interacted with the backend database to dynamically populate products and manage user accounts.
- Implementing PHP in Agora allowed us to create reusable code components. For instance, defining functions for user authentication and session management streamlined the development process and fortified the platform's security measures. PHP's ability to connect with various database systems, such as MySQL, enabled efficient data retrieval and storage, crucial for product management and order processing.

Relational Level:

- The synergy between PHP and other components was evident in Agora's functionality. PHP facilitated interactions between users and the platform, validating inputs, and ensuring secure data transmission. We ensured robust security practices, employing PHP's validation functions to prevent common vulnerabilities like SQL injection and implementing encryption for sensitive user data.
- Furthermore, PHP's flexibility enabled the creation of a modular architecture within Agora. This modular approach not only eased development but also facilitated scalability, allowing for seamless feature additions and enhancements as the platform evolved.

Extended Abstract Level:

- Developing Agora with PHP was an enriching experience that went beyond coding. Challenges arose during the development phase, from managing user sessions across various pages to optimizing database queries for performance. These challenges served as learning opportunities, deepening our understanding of PHP's capabilities and refining our problem-solving skills.
- The impact of Agora on its users was evident in its intuitive interface, efficient transaction processes, and robust security measures. Seeing users engage with the platform, confidently browsing through products, making purchases, and leaving positive feedback, was rewarding. It underscored the significance of robust PHP-based development in creating a user-friendly, secure, and scalable e-commerce ecosystem.
- In hindsight, while the journey was rewarding, there were areas that could have been further optimized. Fine-tuning database queries for better performance and exploring advanced PHP caching mechanisms could have enhanced Agora's responsiveness, especially during peak traffic times.

Looking ahead, continuous improvements to Agora involve exploring newer PHP frameworks and adopting more sophisticated security measures. Additionally, ensuring compatibility with evolving web standards and optimizing the platform for mobile users remain key focus areas.

In conclusion, developing Agora with PHP was not just about building an e-commerce portal but an immersive learning experience. It solidified the importance of PHP's versatility, robustness, and scalability in crafting feature-rich web applications that cater to diverse user needs.