|  |  |
| --- | --- |
| Name of Student | PRESI S PREMACHANDRAN |
| Internship Project Title | TCS ion RIO-125: Operations management-Inventory Module using MERN Stack |
| Name of the Company | TCS ion |
| Name of the Industry Mentor | Debashis Roy |
| Name of the Institute | ICT Academy of Kerala |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Start Date | End Date | Total Effort (hrs.) | Project Environment | Tools used |
| 9-4-2023 | 29-05-2023 | 140hrs | MERN Stack (Mongo DB, Express JS, React JS, Node JS) | VS Code, npm, create-react-app, MongoDB Compass, postman, Chrome developer tools |

**TABLE OF CONTENT**

* Acknowledgements
* Objective
* Introduction / Description of Internship
* Internship Activities
* Approach / Methodology
* Challenges & Opportunities
* Risk Vs Reward
* Reflections on the Internship
* Recommendations
* Outcome / Conclusion
* Enhancement Scope
* Research questions and responses
* Link to code and executable file
* Link to the Demo video

**Acknowledgements**

I want to express my heartfelt appreciation to TCS iON for granting me the privilege to undertake this project. I extend my special thanks to the industry mentors, Mr. Debhasis Roy, for their invaluable assistance and guidance throughout the project. Lastly, I am grateful to all those who supported me in making this project a resounding success.

**Objective**

To create an automated software solution for the inventory management module within operations management, utilizing the MERN stack.

**Introduction / Description of internship**

The primary objective of my internship is to gain practical experience in developing software solutions using the MERN stack. The assigned industry project involves the creation of an inventory management module within operations management, encompassing four distinct sub-modules:

**Inventory:** This sub-module comprises inventory items, inventory groups, and inventory adjustments.

**Sales:** The sales sub-module encompasses customers, sales orders, invoices, packages, delivery challans, payments received, sales returns, and credit notes.

**Purchase**: The purchase sub-module includes vendors, purchase orders, bills, bill payment, and vendor credit.

**Dashboard and Reports**: The dashboard serves as the application's main landing page, providing a comprehensive summary from the other three sub-modules. Additionally, the module features four reports: inventory summary report, inventory aging summary report, product sales report, and sales by items/customer.

My internship spans a duration of 30 days, and this project report serves as a comprehensive documentation of my accomplishments throughout the industry project

**Internship Activities**

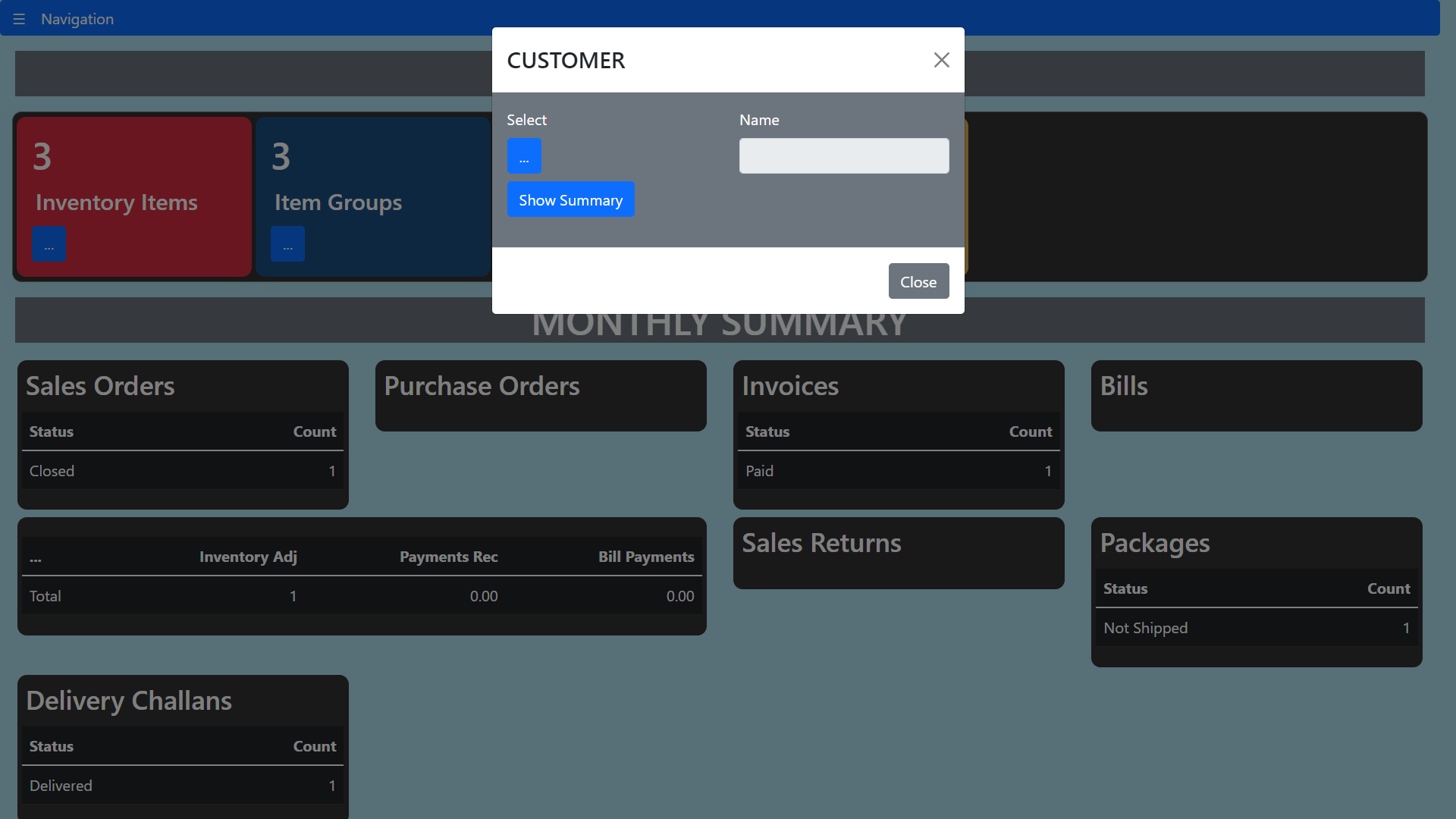
During the internship, my activities revolved around self-study, participating in knowledge-sharing sessions in the digital discussion room, and working on the assigned industry project under the guidance of mentors. At the end of each day, I diligently submitted activity reports detailing the tasks accomplished along with their respective time durations.

For the backend development, I utilized the Express framework and the Mongoose ODM to establish a connection with and perform data operations on the MongoDB Atlas database. On the frontend, I leveraged React and the create-react-app framework, along with react-router for seamless page routing. To ensure responsive web design, I incorporated the Bootstrap CSS framework. Asynchronous HTTP requests to the backend were facilitated using Axios, while user authentication was implemented using JSON Web Tokens (JWT). Furthermore, I employed the html-pdf package to generate PDF documents from HTML templates on the backend, streaming them to the frontend via specific endpoints. Additionally, I used the xlsx package to convert MongoDB data into Excel format on the backend, enabling users to download Excel files. Other packages utilized include Multer for handling multipart form posting, bcrypt for password encryption, dotenv for managing environment variables through a .env file, and cors for cross-origin resource sharing.

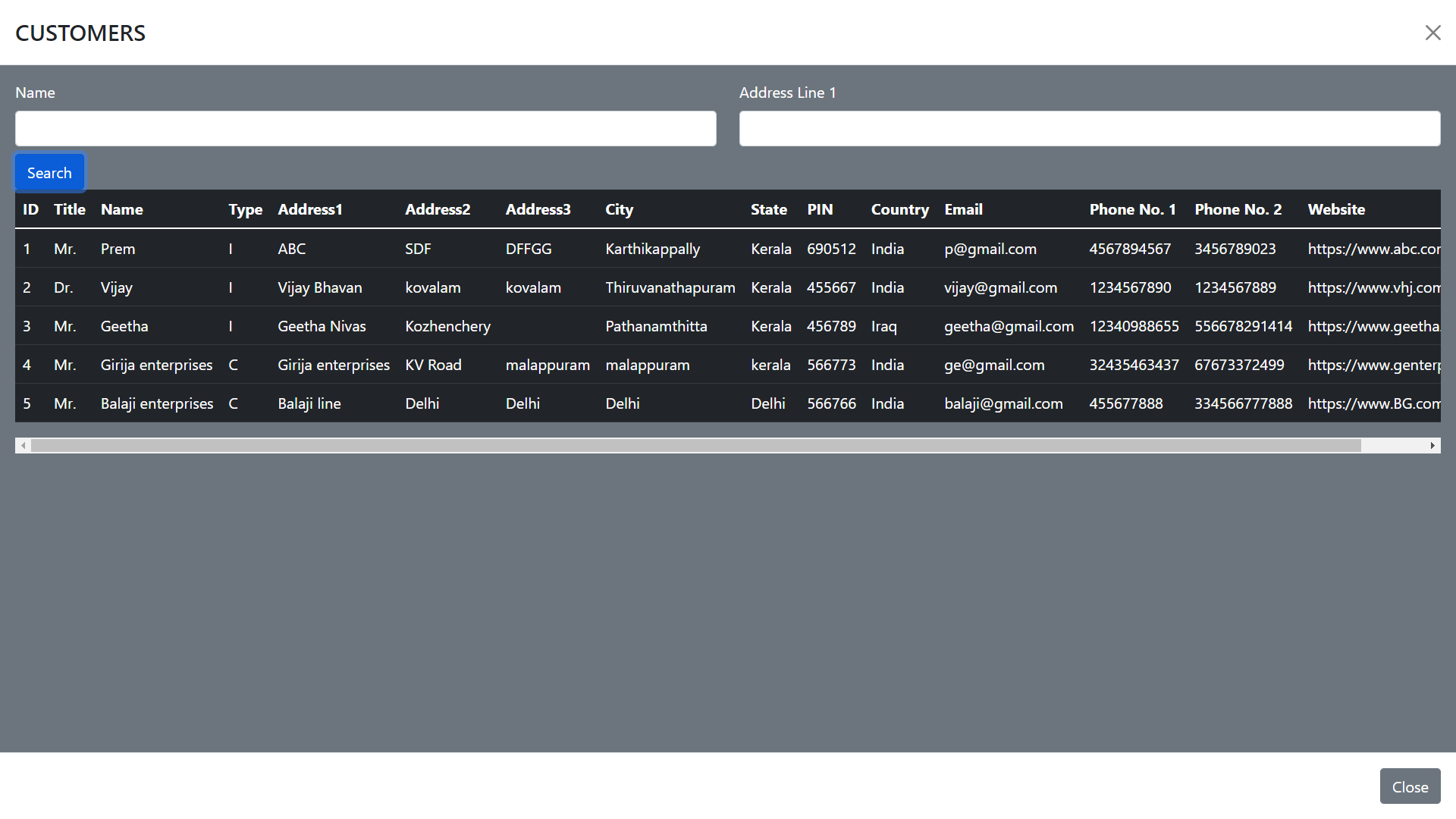
In terms of development, I created data entry forms for inventory items, groups, and adjustments within the inventory sub-module. Similarly, in the sales sub-module, I designed forms for data entry of customers, sales orders, packages, delivery challans, invoices, payments received, sales returns, and credit notes. The purchase sub-module also featured data entry forms for vendors, purchase orders, bills, and payment and vendor credit. Furthermore, I developed a comprehensive dashboard comprising general and monthly summaries. The general summary provided insights into individual items, customers, vendors, and item groups. The monthly summary showcased the number and status of sales orders, purchase orders, packages, delivery challans, invoices, bills, and sales returns created during the current month. It also presented monthly receipts, payments, and the total number of inventory adjustments made. Moreover, I implemented reports that could be viewed in PDF format and exported as Excel files.

Certain forms were equipped with editing capabilities for modifying existing data. I also implemented a component that allowed for item selection to be added to sales orders, packages, invoices, delivery challans, etc. Another component I developed was a picklist, which facilitated the selection of items from a table of contents (e.g., a detailed customer list). Additionally, I enabled the viewing of sales orders, purchase orders, and invoices in PDF format. Users could also export inventory adjustments made during a specified period to Excel format.

Throughout the internship, I successfully completed all of these activities.



**fig.1 – Picklist**



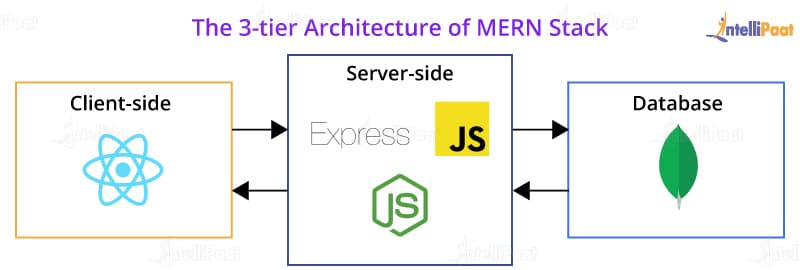
**Fig.2 - After clicking the … button and searching with name.**

**Approach/Methodology**

The MERN stack comprises MongoDB as the database, Express framework on Node.js for the backend server, and React.js for building the frontend. One of the major advantages of the MERN stack is its unified use of JavaScript across all components, making it sufficient to have knowledge of JavaScript alone to develop web applications using this stack. Therefore, the MERN stack was chosen for this project.

To simplify data operations with MongoDB, an ODM called Mongoose was used. Express framework facilitated the setup of an HTTP server for the backend, simplifying the creation of a REST API. React.js, with its component-based architecture, made frontend design for single-page applications more manageable, and the create-react-app framework streamlined the development process. The inclusion of Bootstrap ensured that the web application would be accessible on various devices with different screen sizes.

Figure 4 depicts the basic MERN architecture, which served as the foundation for this industry project.



**Fig.3– MERN Architecture**

The frontend comprised data entry forms, reports, and tables to display data fetched from the backend. Asynchronous requests between the frontend and backend were made using the Axios package. Express facilitated handling HTTP requests and responses on the backend. Data underwent primary validation before being sent from the data entry form, and further validation occurred on the backend prior to storing the data in MongoDB. Mongoose performed validations based on the specified schema for the data model. Post requests were primarily used for submitting data, while get requests were used to retrieve data. HTTP put requests were employed for updating existing entries. Express also handled routing for the various API calls originating from the frontend.

Authentication was implemented using JSON Web Token (JWT) for all requests except for the login request. A JWT was sent back to the user upon successful authentication, and it was stored in the browser's local storage on the frontend. Before making subsequent requests, the JWT was retrieved from the local storage and embedded in the HTTP header field 'x-access-token'. At the backend, the JWT was verified after extracting it from the HTTP header field. The JWT contained a payload consisting of the username and whether the user was an admin. This information could be obtained on the backend after JWT verification. JWT served the purpose of authentication and authorization. In this web application, only an admin had the privilege to create a new user.

For handling multipart-form requests, the Multer package was utilized. In this project, Multer was employed to handle the uploading of item images. Item images were uploaded to a public folder on the Node server, and their filenames were dynamically generated and saved in MongoDB. When retrieving item details, the image path was constructed using the filename stored in the database.

To generate PDF files from HTML templates, the html-pdf package was utilized. For each report, an HTML template was created, and the create function of the html-pdf package was used to convert the HTML string to a PDF. The create function accepted two parameters: the HTML string and an options object, allowing for specification of orientation and paper size. The generated PDF was then streamed to the frontend as a binary buffer with the response type set to PDF.

The xlsx package was employed to create Excel files from arrays of objects. Using utility functions provided by xlsx, JSON data was converted to a worksheet using json\_to\_sheet, which was then appended to a workbook object created using the book\_new() utility function. The resulting workbook was written to a buffer using the write function of xlsx. This buffer was subsequently streamed to the frontend for downloading.

The frontend consisted of numerous data entry forms, tables, and reports. All data entry forms followed a similar logic, as illustrated in the flowchart. Each form and its functionality will be detailed in the subsequent sections.

**Flow Chart**

Yes

No

No

Yes

Show failure message

Show success message

Is posting success?

Get response from backend server

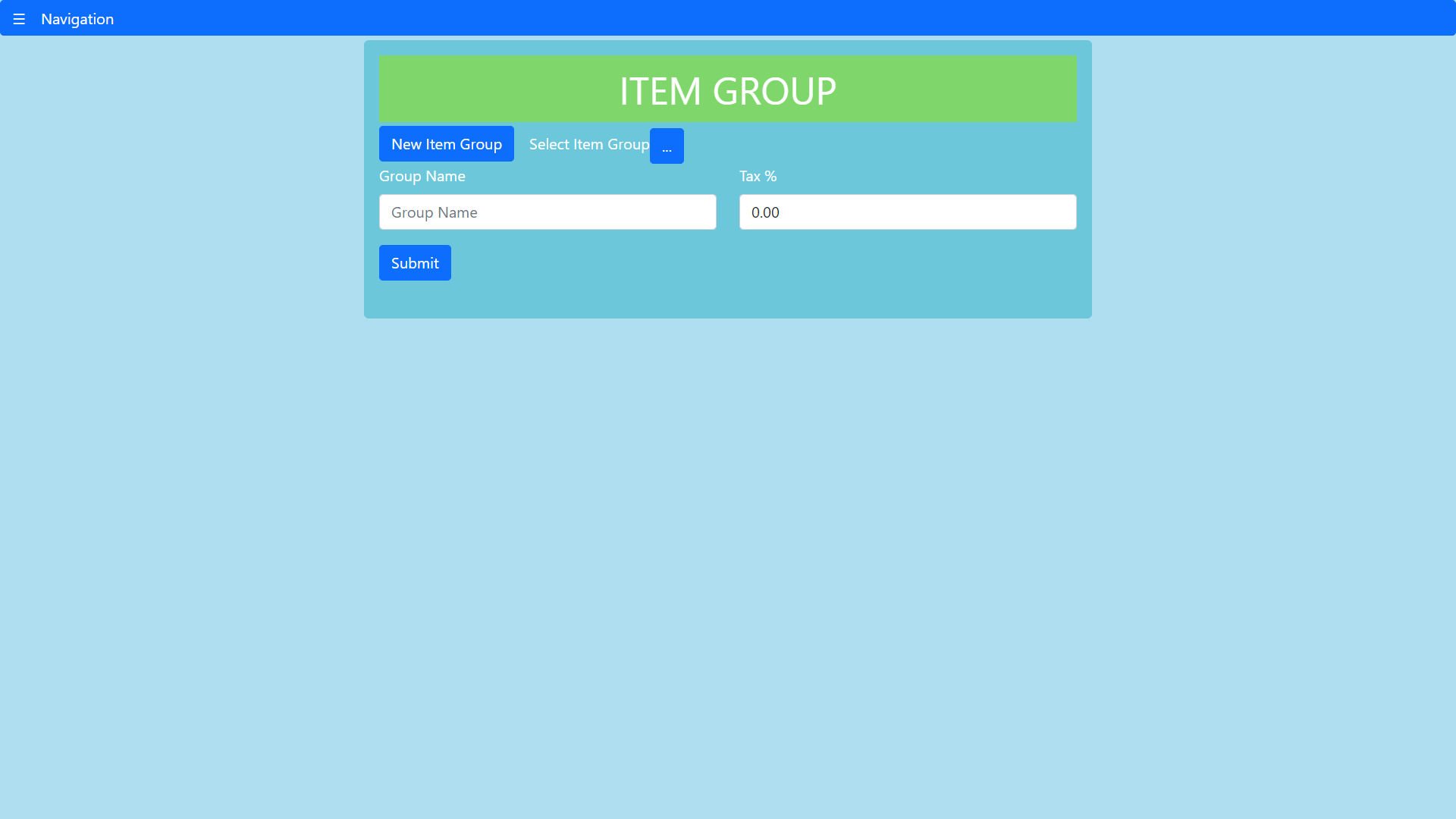
Post the form data to backend server

Validate Entry

Enter details in data entry form

1. **Inventory Sub module**
2. **Item Group**

An item group is a collection of products which share similar attributes like color, production, features, or usage. Item groups can also be formed based on the markets in which they’re sold or if they’re similar in price.  In this web application, all the items in a specific group are having the same tax.



**Fig.5 – Item group data entry form**

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. groupID | Int32 |
| 1. groupName | string |
| 1. tax | Decimal128 |

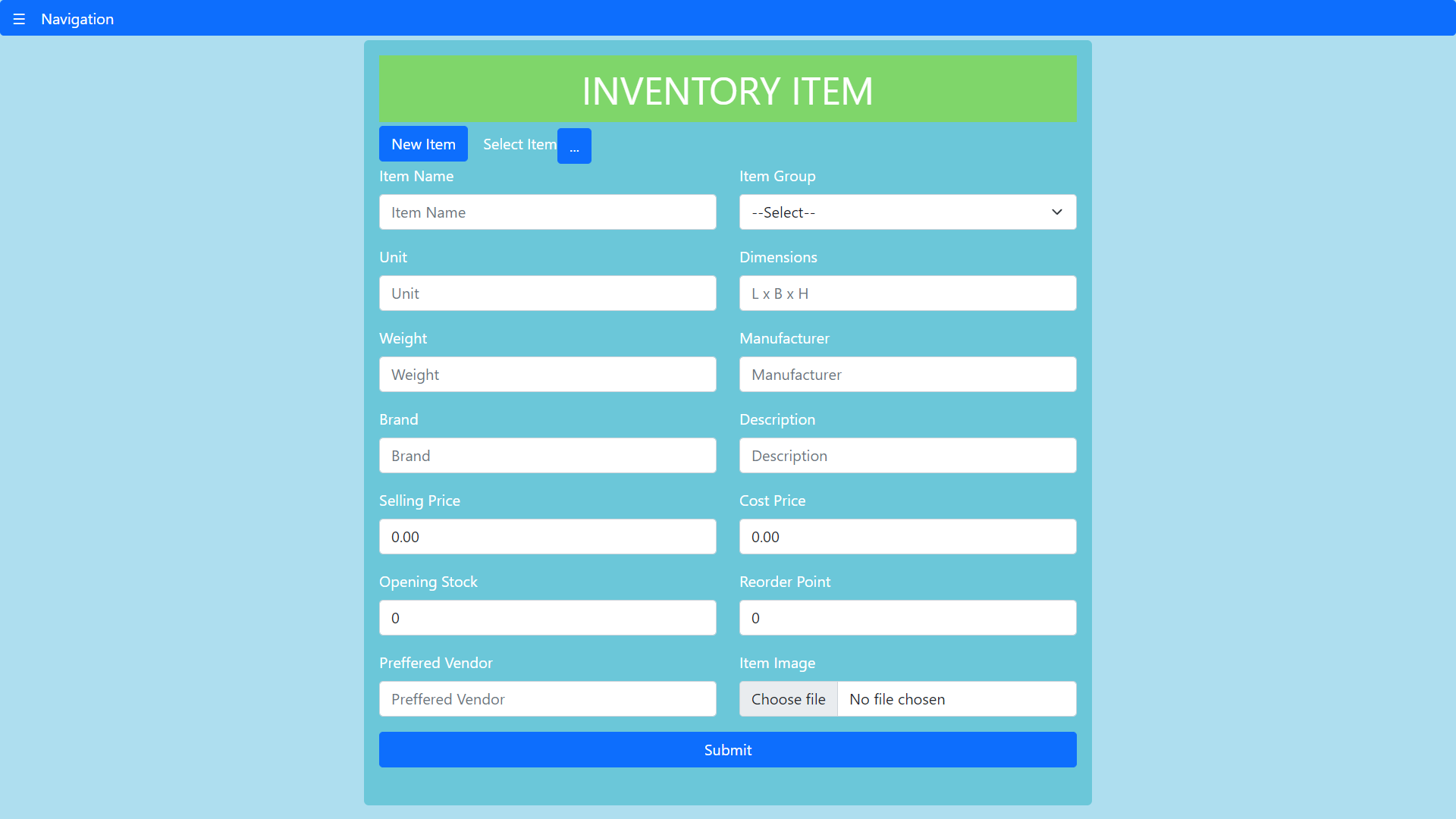
In this form groupID is automatically generated at the backend. Group name and tax percentage can be entered to insert a new item group. For updating, the picklist is used to select the item group which is to be updated.

1. **Items**

This form is used for entering inventory items. In this form, it is possible to assign an item group to each item. Based on the item, other details can be filled. The file input is used to upload the item image. This form is a multipart form and its posting is handled by multer as a middleware at the backend before inserting to database. Updating certain fields like cost price, selling price, reorder point, preferred vendor and description is possible.

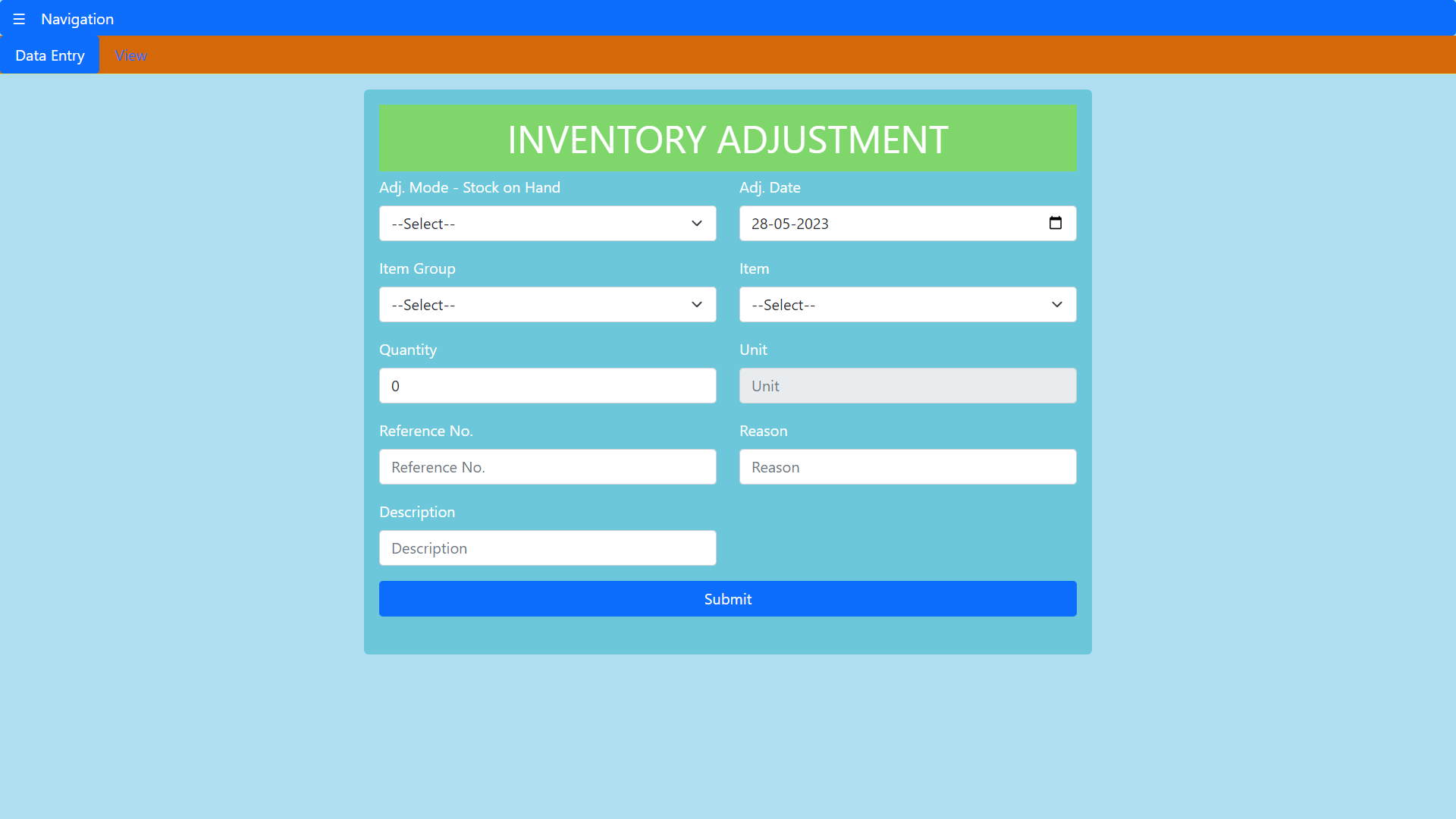
*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. itemID | Int32 |
| 1. itemName | string |
| 1. groupID | Int32 |
| 1. groupName | string |
| 1. manufacturer | string |
| 1. brand | string |
| 1. descr | string |
| 1. dimensions | string |
| 1. weight | Decimal128 |
| 1. costPrice | Decimal128 |
| 1. sellingPrice | Decimal128 |
| 1. unit | string |
| 1. reorderPoint | Decimal128 |
| 1. openingStock | Decimal128 |
| 1. prefVendor | String |
| 1. itemImg | String |



**Fig.6 – Inventory item data entry form**

1. **Inventory Adjustment**



**Fig.7 – Inventory adjustment form**



**Fig.8 – View inventory adjustments for a period or date**

Inventory adjustments are corrections of inventory or stock records to bring them into agreement with the findings of the actual physical inventory. Inventory adjustments are increases or decreases made in inventory to account for theft, loss, breakages, and errors in the amount or number of items received. In this form inventory adjustment mode includes two options namely increase or decrease. The item can be selected and the quantity for adjustment of inventory can be input. All the inventory adjustments done for a period can be viewed in tabular form and also can be exported as excel file.

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. adjID | Int32 |
| 1. adjDate | date |
| 1. adjMode | string |
| 1. description | string |
| 1. itemID | Int32 |
| 1. itemName | string |
| 1. quantity | Int32 |
| 1. reason | string |
| 1. refNo | string |
| 1. unit | string |

**(ii) Sales Sub module**

**(a) Customer**



**Fig.9 – Customer data entry form**

This form is used to enter details of sales customers. Sales customers can be individuals as well as institutions. Customer details can also be edited by selecting the customer using the picklist in the form.

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. customerID | Int32 |
| 1. customerName | string |
| 1. customerType | string |
| 1. addressLine1 | string |
| 1. addressLine2 | string |
| 1. addressLine3 | string |
| 1. city | string |
| 1. state | string |
| 1. country | string |
| 1. emailID | string |
| 1. pincode | string |
| 1. contactNo1 | string |
| 1. contactNo2 | string |
| 1. title | string |
| 1. website | string |

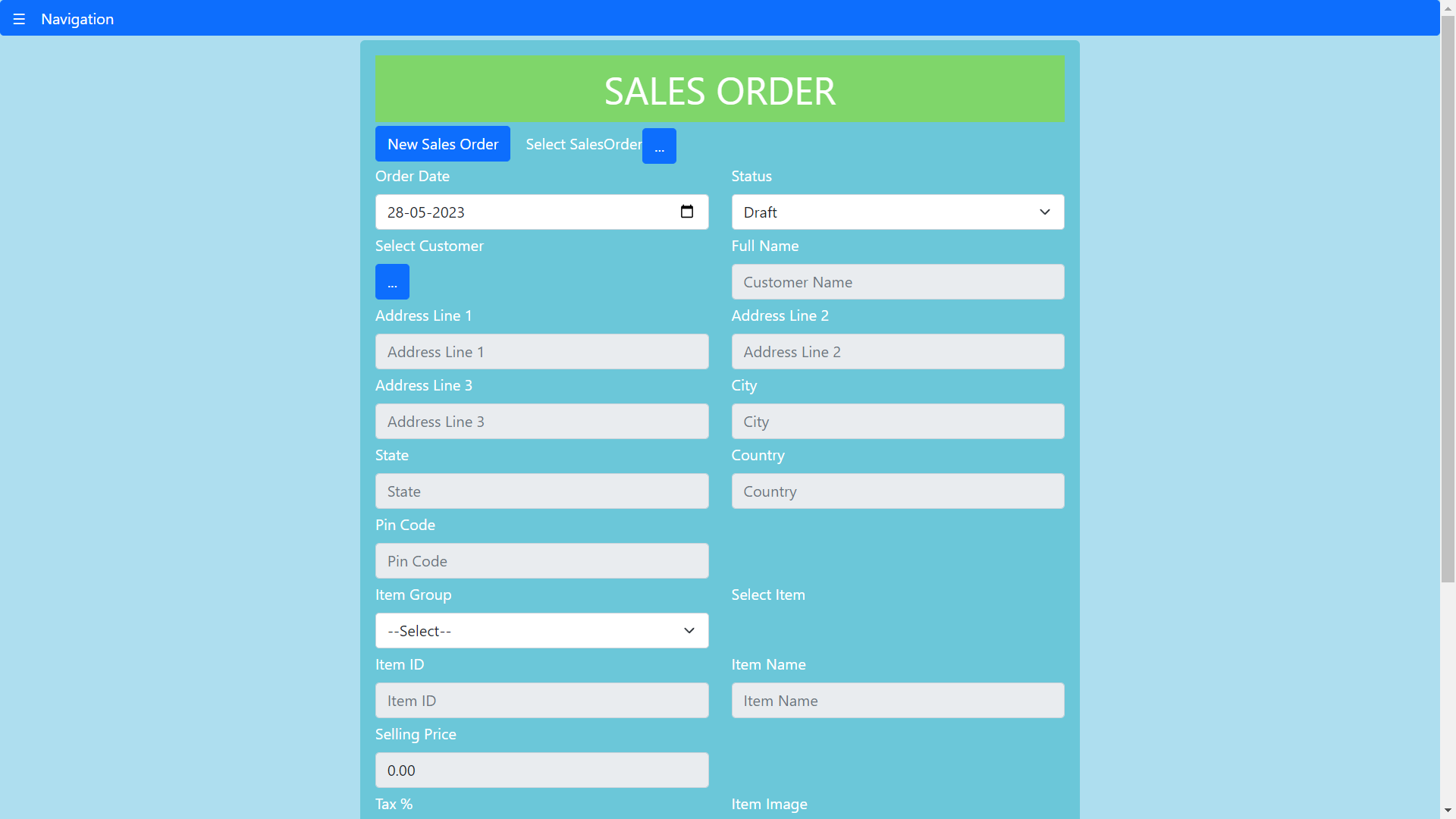
**(b) Sales Order**

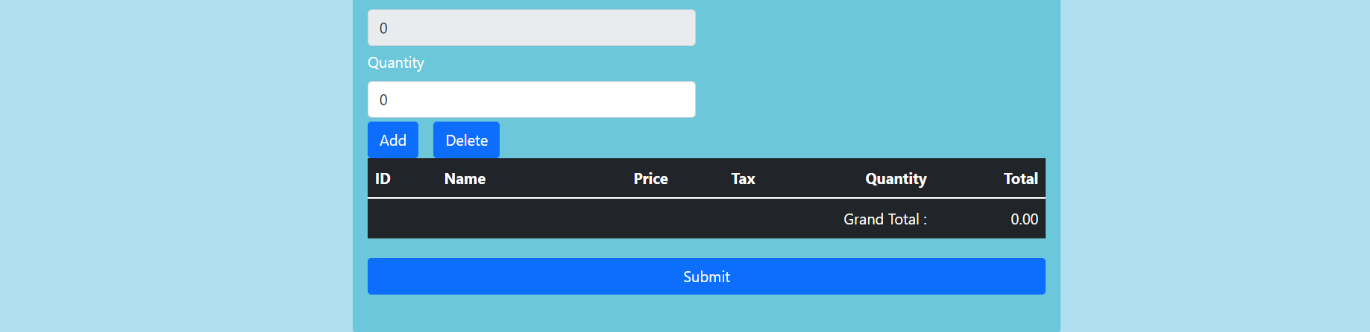
A sales order is a commercial document -- prepared by a seller and issued to a customer -- confirming the sale of goods or services involved in a given transaction. The document contains details about the sale, including the quantity, quality, and price of any goods or services exchanged. The data can be edited if the sales order status is in draft. . Status of sales order include draft, confirmed and closed. It is possible to view the sales order in pdf format which can be sent to the customer.

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. salesOrderID | Int32 |
| 1. salesOrderDate | date |
| 1. customerID | Int32 |
| 1. items | Array of documents |
| 1. status | string |

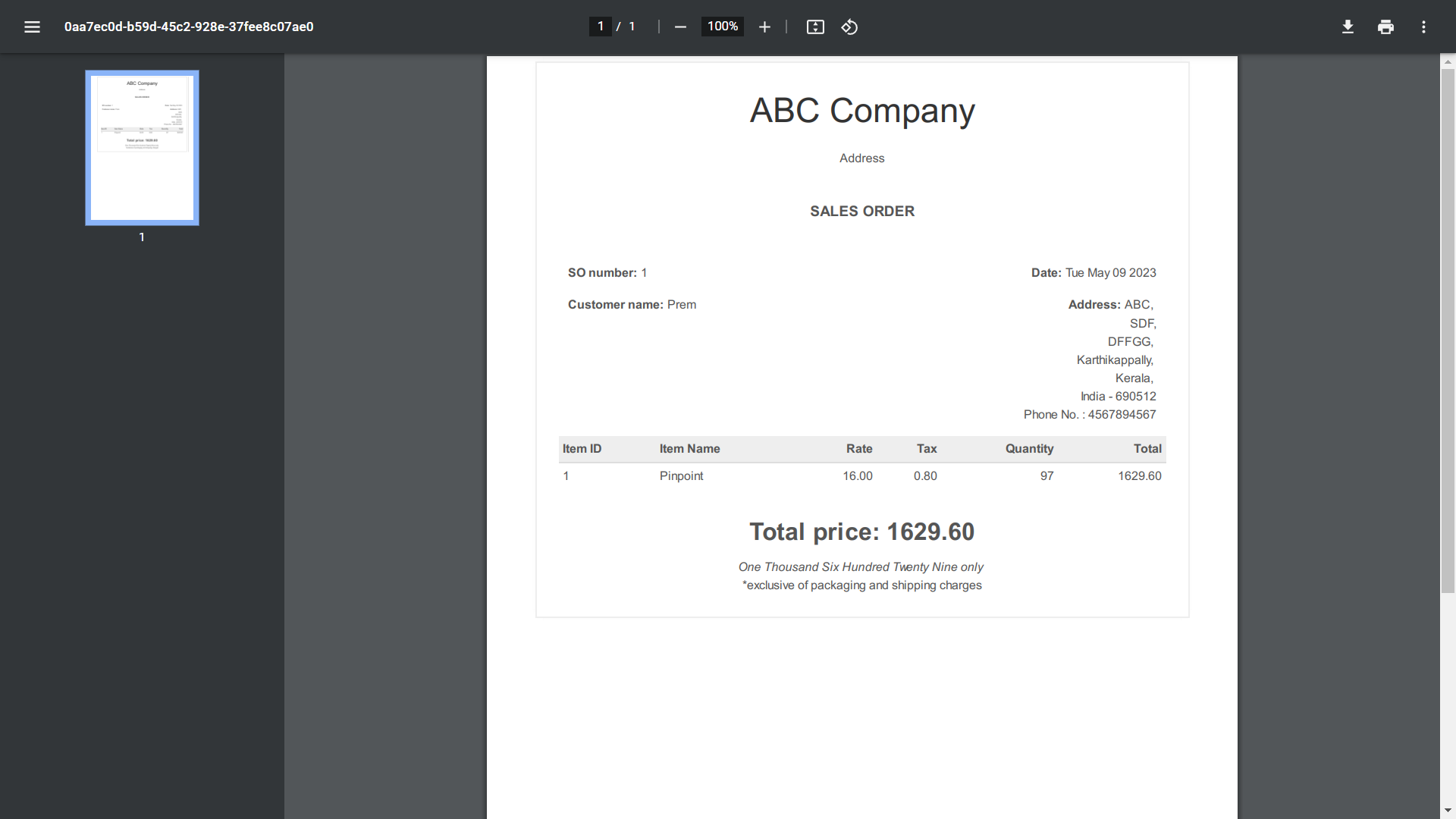
**Schema of Array of documents – items**

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. itemID | Int32 |
| 1. itemName | string |
| 1. price | Decimal128 |
| 1. tax | Decimal128 |
| 1. quantity | Decimal128 |
| 1. total | Decimal128 |

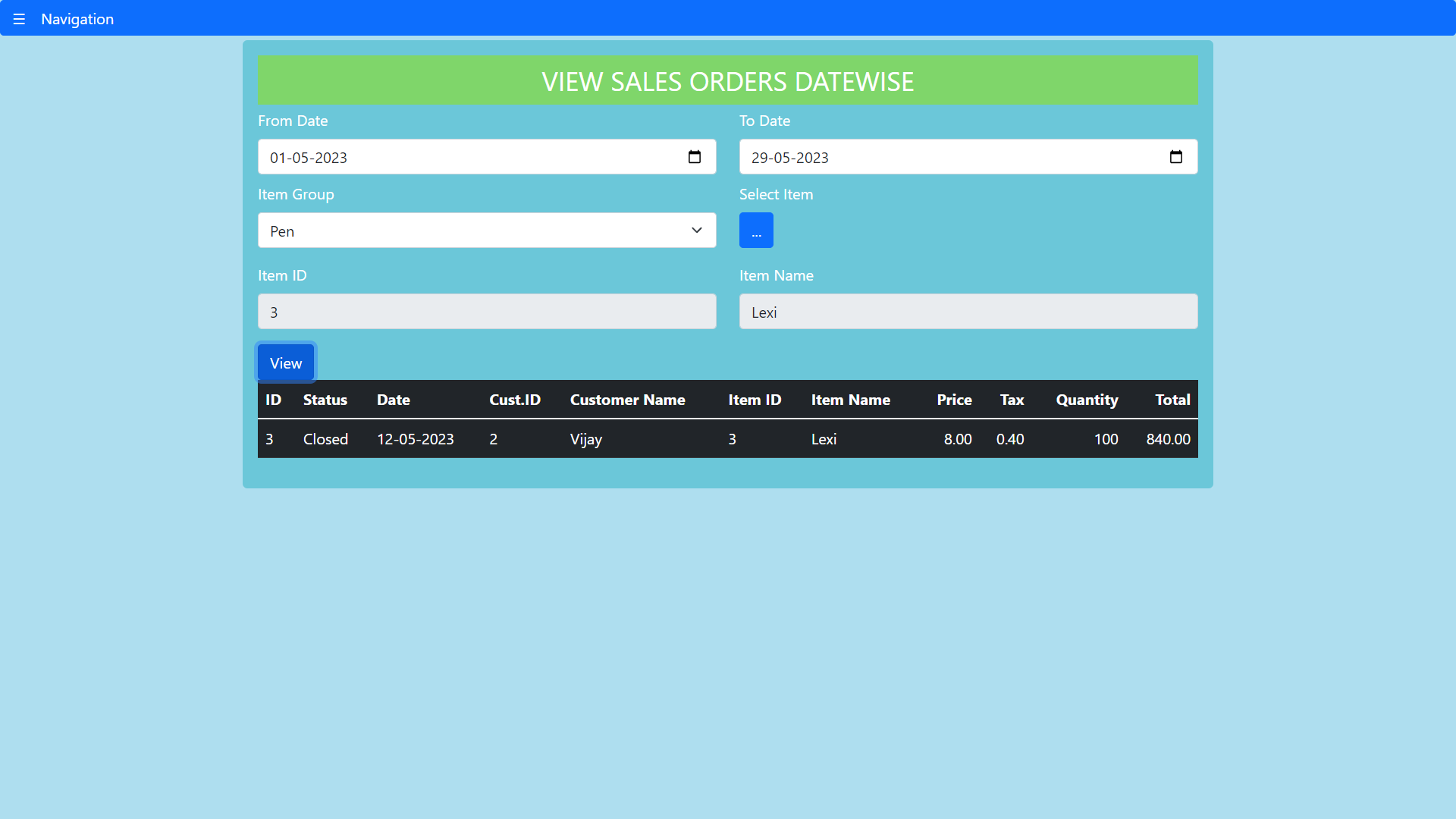


****

**Fig.10 – Sales order data entry form**



**Fig.11 – Sales Order in pdf format**



**Fig.12 - Sales orders during a period**

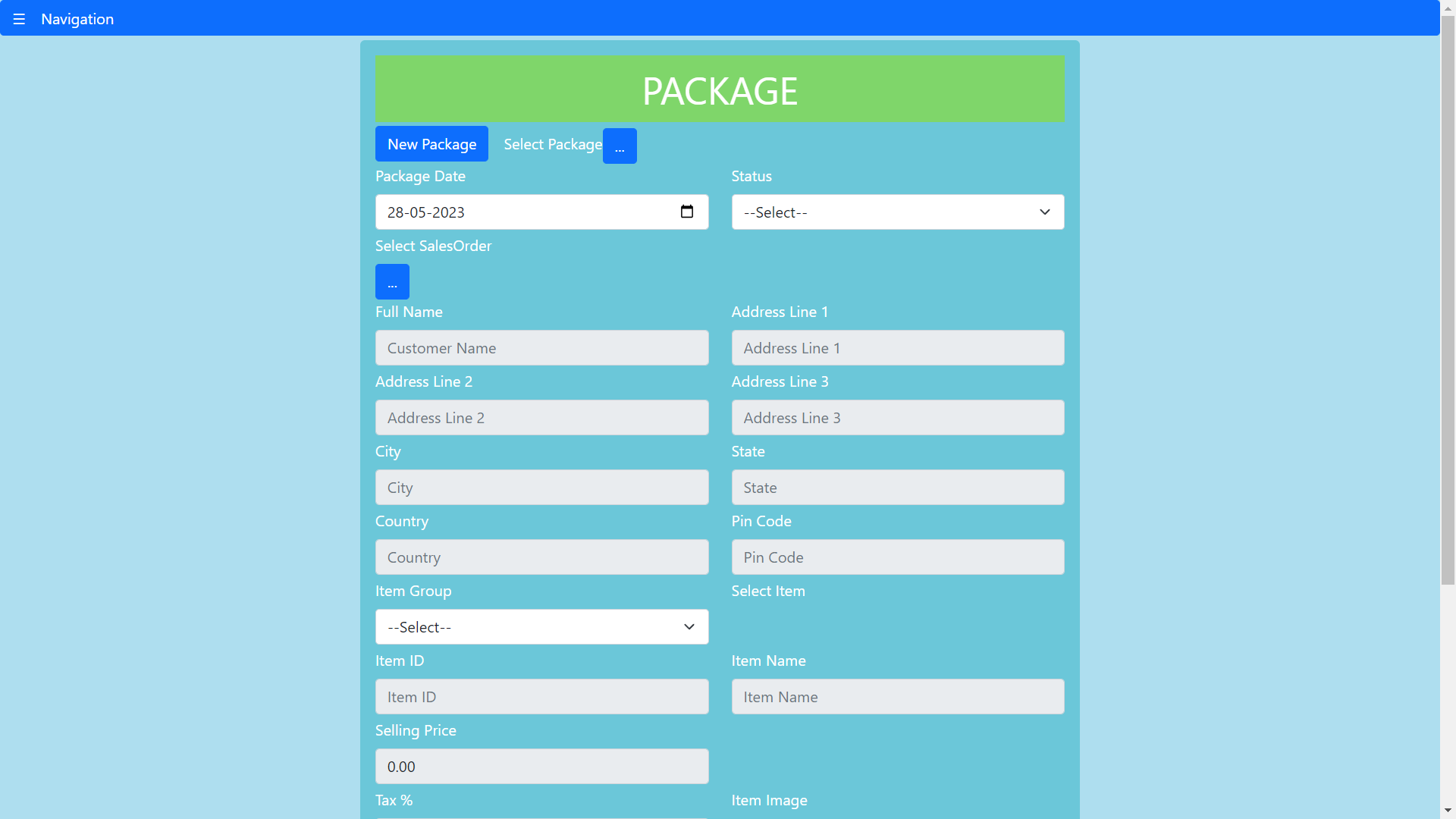
**(C) Package**

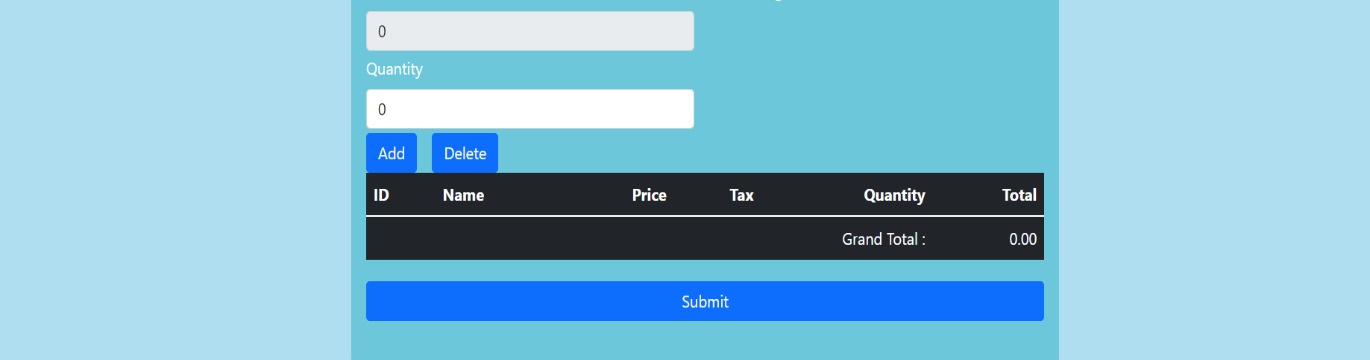
After a sales order is confirmed and invoice is generated, they have to reach the customer. For reaching the customer via courier these sales items are packed in packages. This form is used to enter details of packages thus made for shipment. The status includes not shipped, shipped, delivered and returned.

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. packageID | Int32 |
| 1. packageDate | date |
| 1. salesOrderID | Int32 |
| 1. status | string |
| 1. customerID | Int32 |
| 1. items | Array of documents |

Here items is having same schema as in sales order collection.

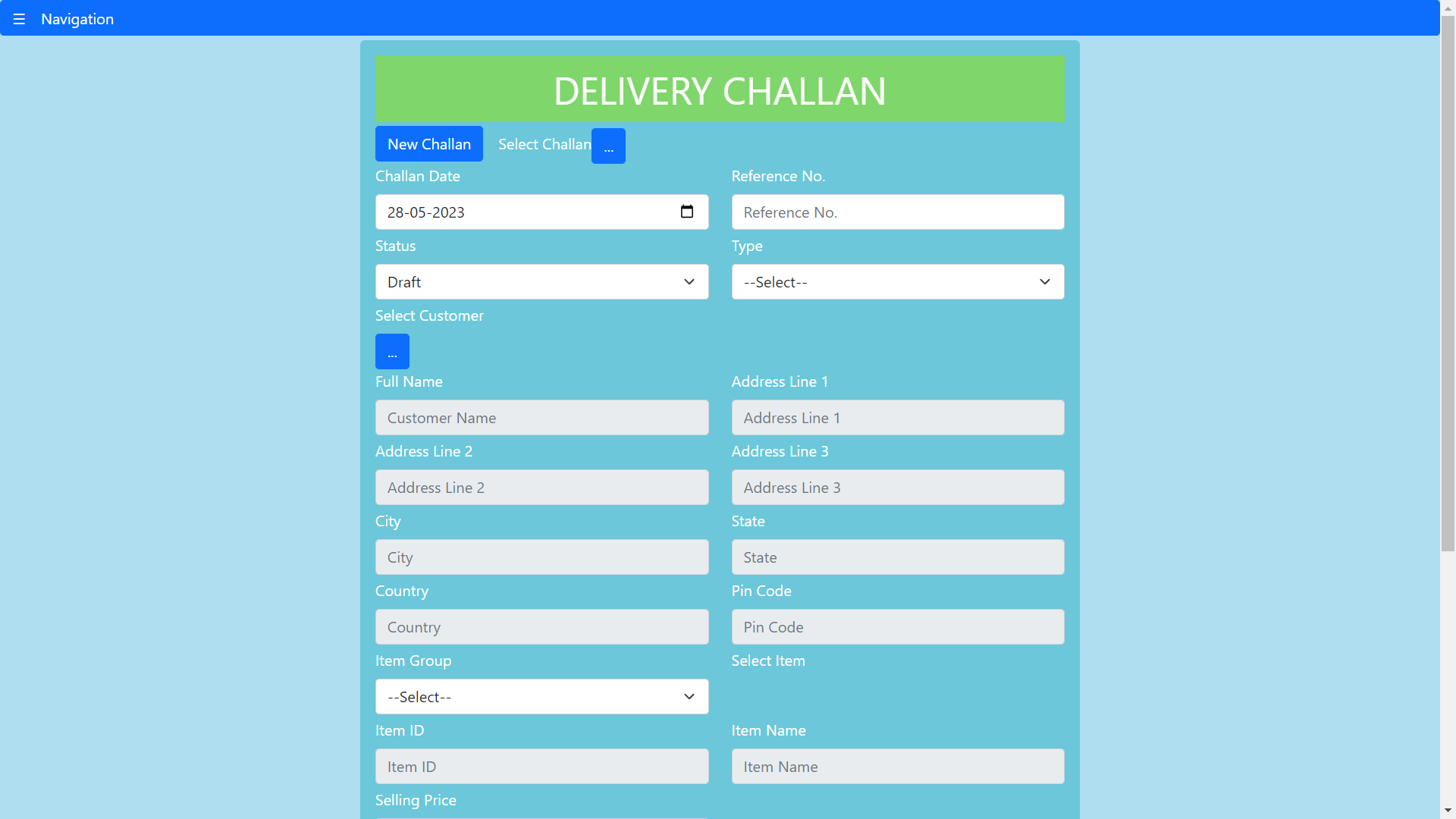


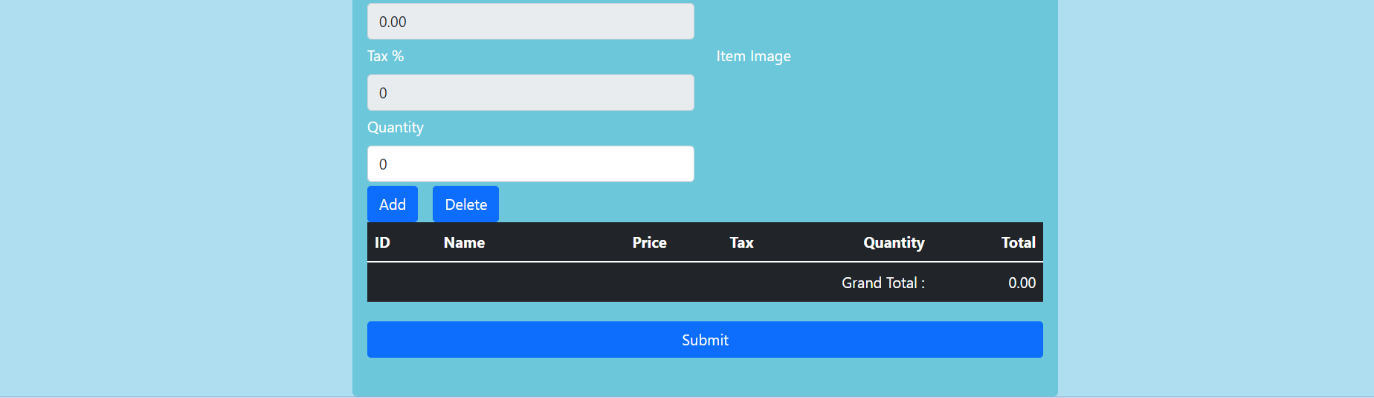


**Fig.13 – Package data entry form**

**(d) Delivery challan**

A delivery Challan is a formal document that is created in situations where goods are being transported from one place to another which may or may not result in sales. The data entry form for delivery challan is as shown in figure where data can be edited if status is draft. The type contains supply of goods on approval, job work, semi assembled goods and others. Status includes draft, open, delivered and returned.





**Fig.14 – Delivery challan data entry form**

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. challanID | Int32 |
| 1. challanDate | date |
| 1. challanType | string |
| 1. customerID | Int32 |
| 1. items | Array of documents |
| 1. refNo | string |
| 1. status | string |

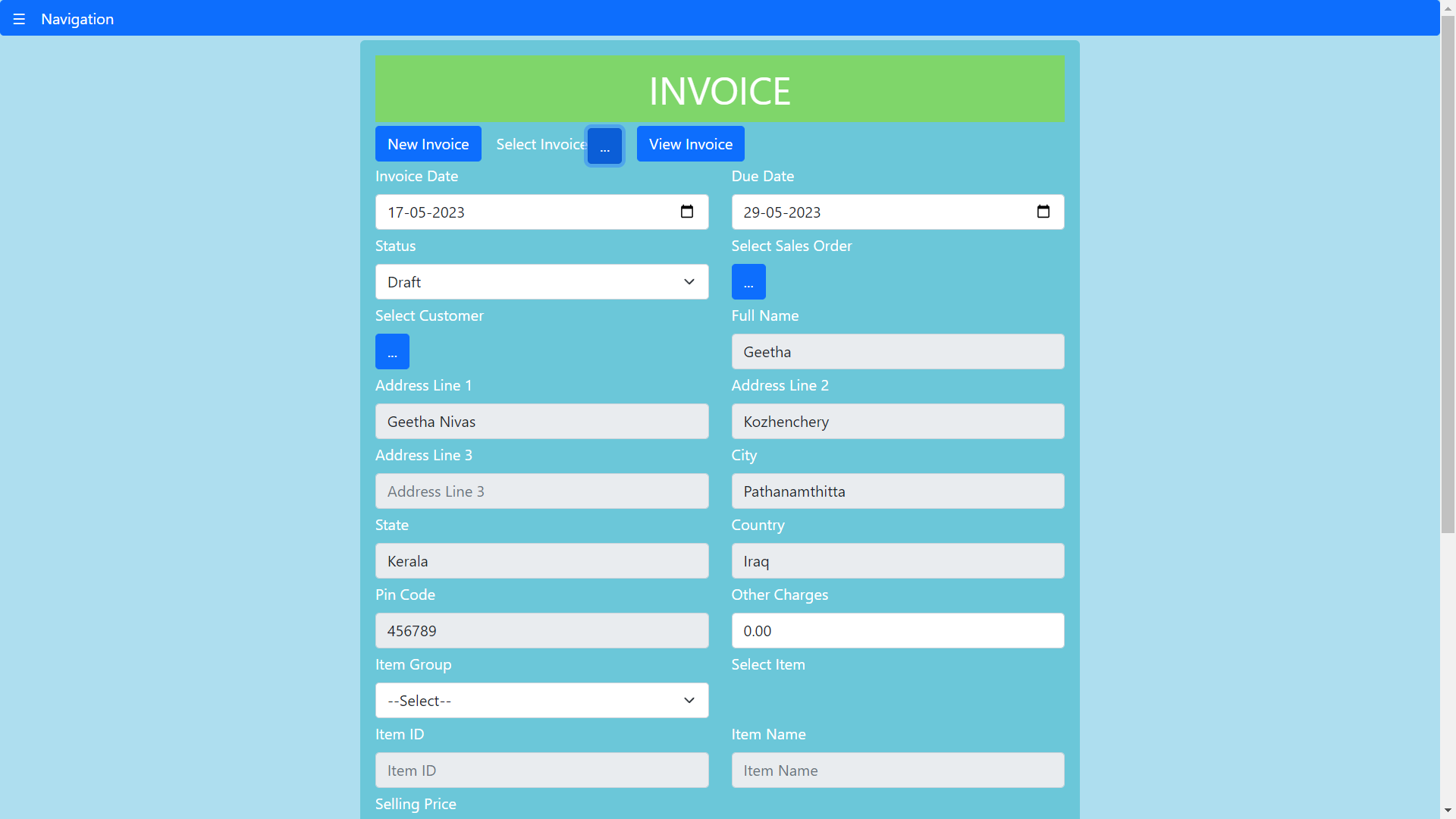
Here items is having same schema as in sales order collection.

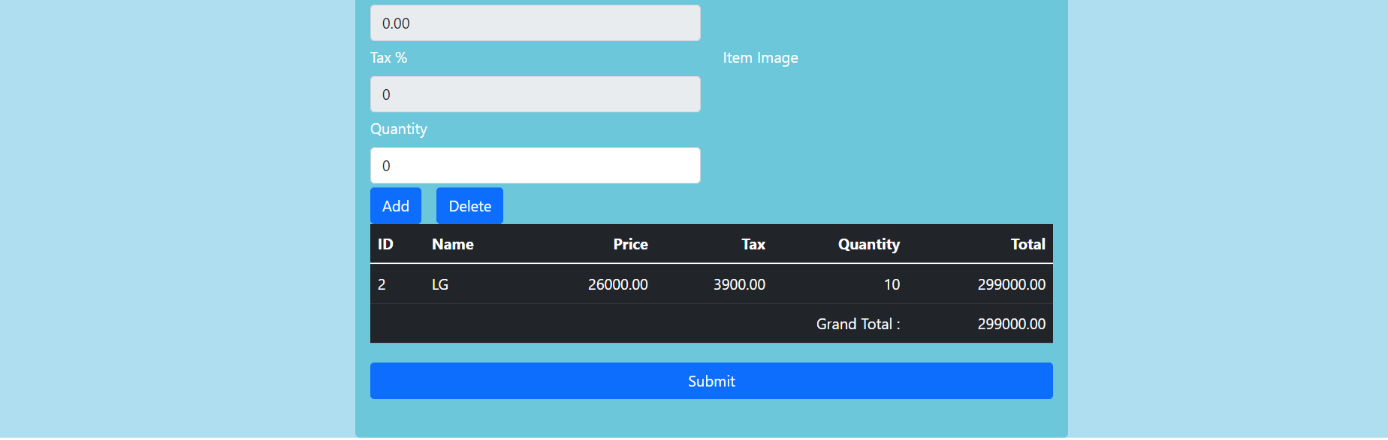
**(e) Invoice**

An invoice is a commercial document issued by a seller to a buyer relating to a sale transaction and indicating the products, quantities, and agreed-upon prices for products or services the seller had provided the buyer. The data entry form of invoice is as shown in figure. The status includes draft, sent, partially paid, paid, due and void. When invoice status changes from draft to any status other than void, stock on hand is decreased as it is considered as sales and stock out. It is possible to get invoice in pdf format by clicking the view invoice button.

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. invoiceID | Int32 |
| 1. invoiceDate | date |
| 1. customerID | Int32 |
| 1. dueDate | date |
| 1. items | Array of documents |
| 1. salesOrderID | Int32 |
| 1. otherCharges | Decimal128 |
| 1. status | string |

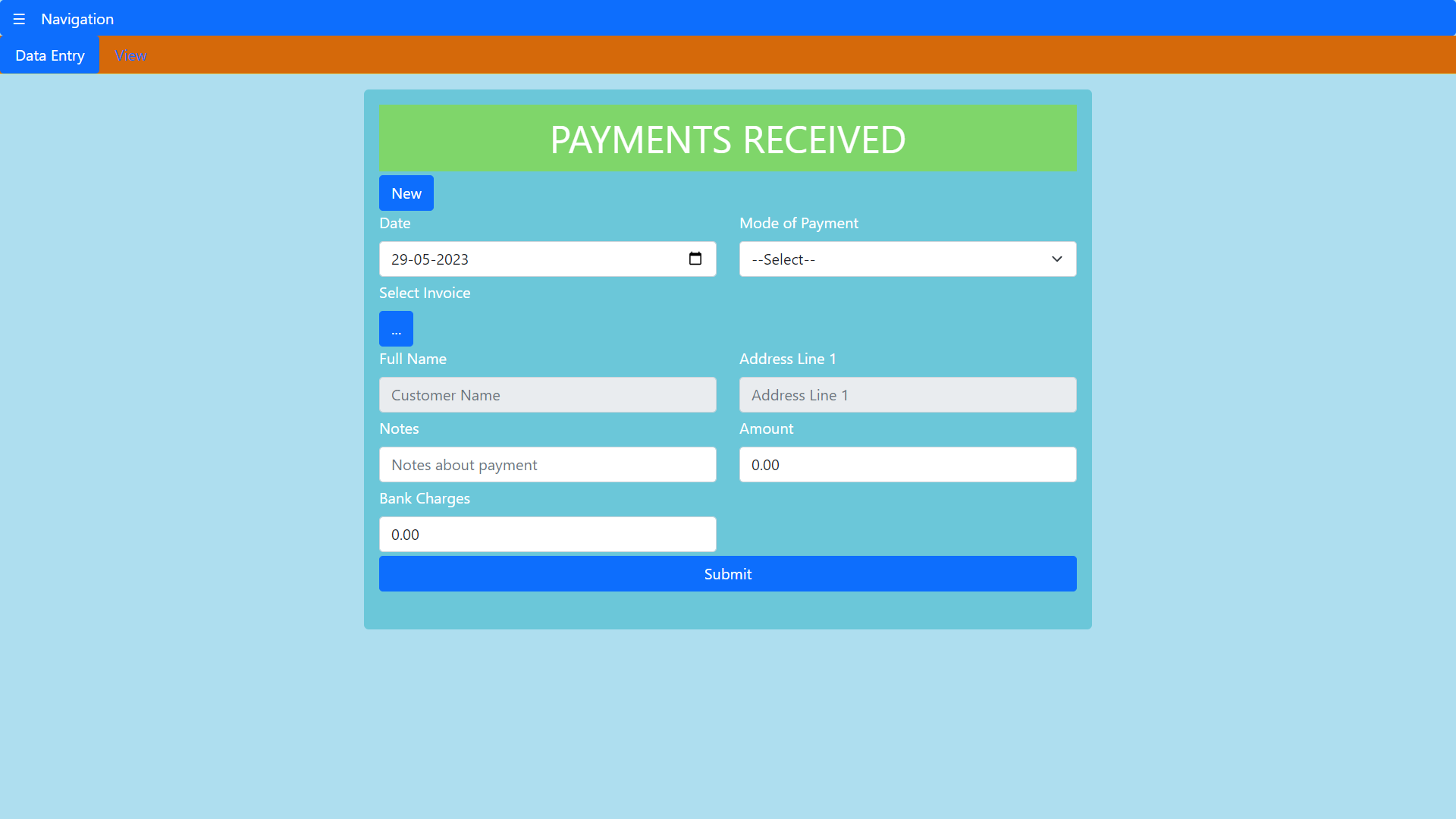




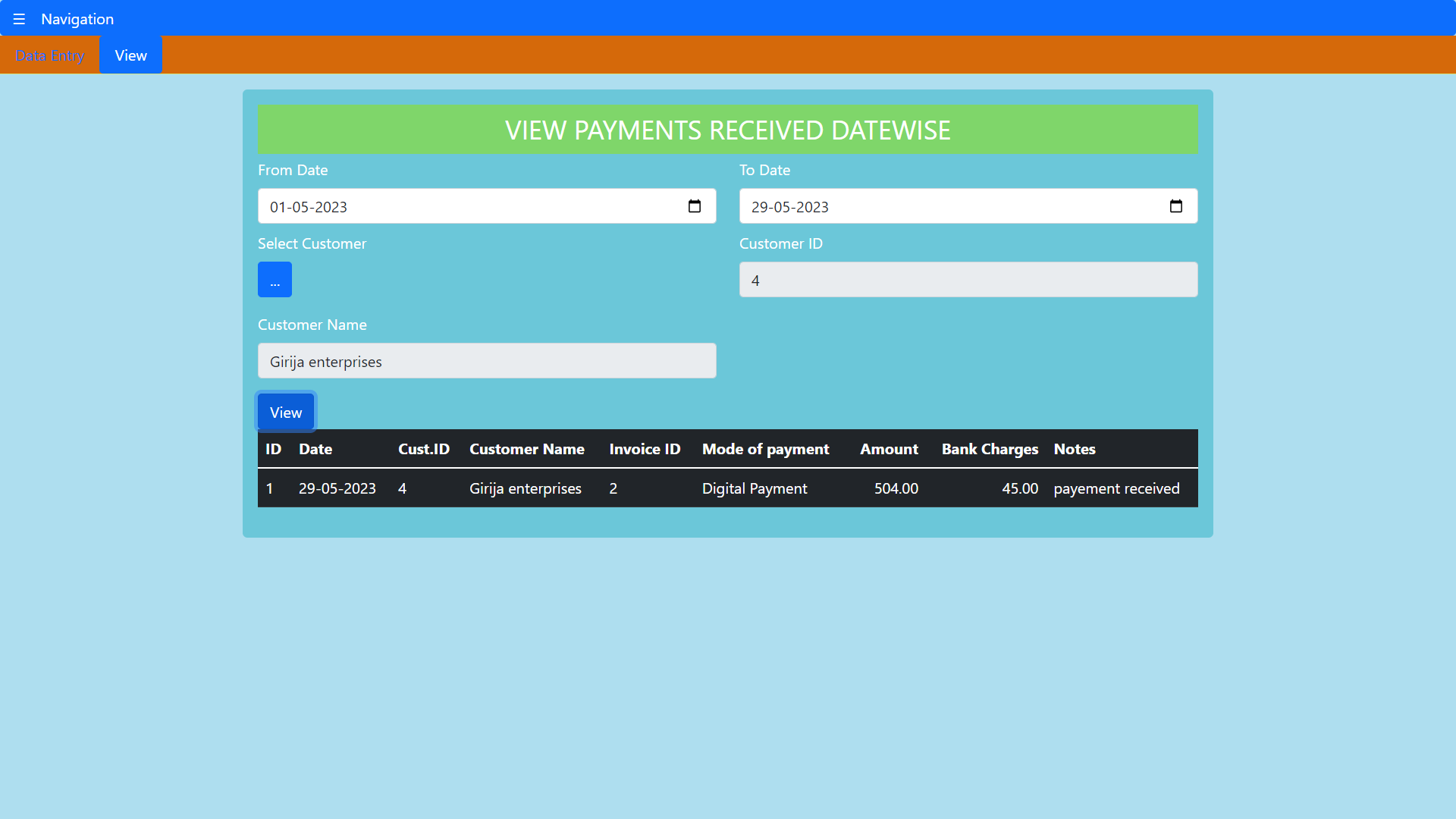
**Fig.15 – Invoice data entry form**

**(f) Payments Received**

This form is used to record receipts from customers for invoice. The invoice can be selected from the picklist and the details like amount is automatically filled. Mode of payment includes by cash, cheque, debit/credit card and digital payment. The payments received during a period can be seen in a table. Also these payments can be filtered customerwise.



**Fig.16 – Payments received data entry form**



**Fig.17 – Payments received during a period**

*Database Schema*

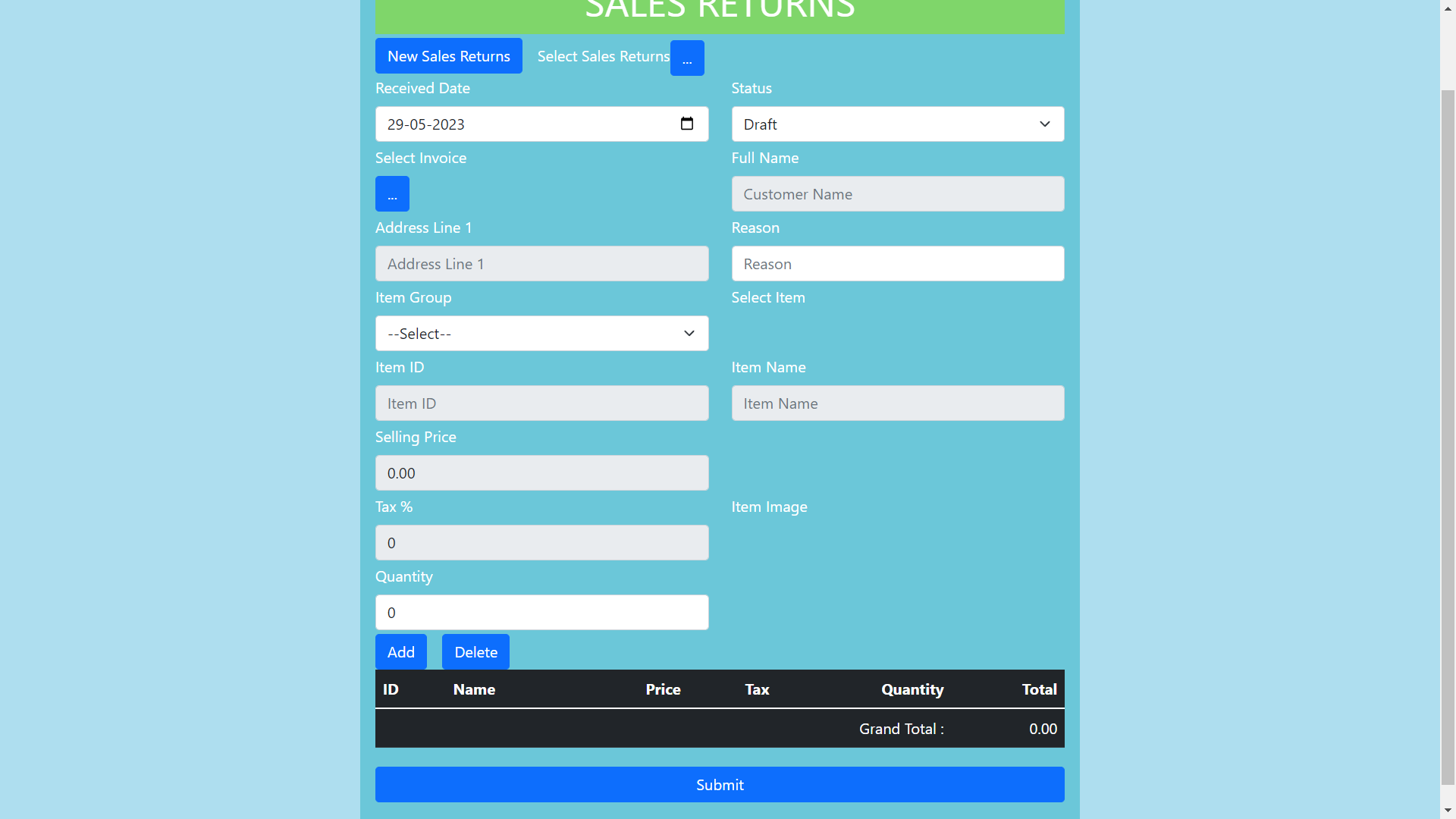
|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. paymentRecID | Int32 |
| 1. paymentRecDate | date |
| 1. customerID | Int32 |
| 1. invoiceID | Int32 |
| 1. amount | Decimal128 |
| 1. bankCharges | Decimal128 |
| 1. modeOfPayment | string |

**(g) Sales Returns**

A sales return is merchandise sent back by a buyer to the seller. The return is usually because an excess quantity was either ordered or shipped, or due to defective goods. The status in the form consists of draft, accepted and declined. The returned goods are checked before accepting, so they will be recorded as draft first. If they are accepted, then status accepted is set and stock on hand is increased.

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. salesReturnID | Int32 |
| 1. receivedDate | date |
| 1. customerID | Int32 |
| 1. invoiceID | Int32 |
| 1. items | Array of documents |
| 1. reason | string |
| 1. status | string |



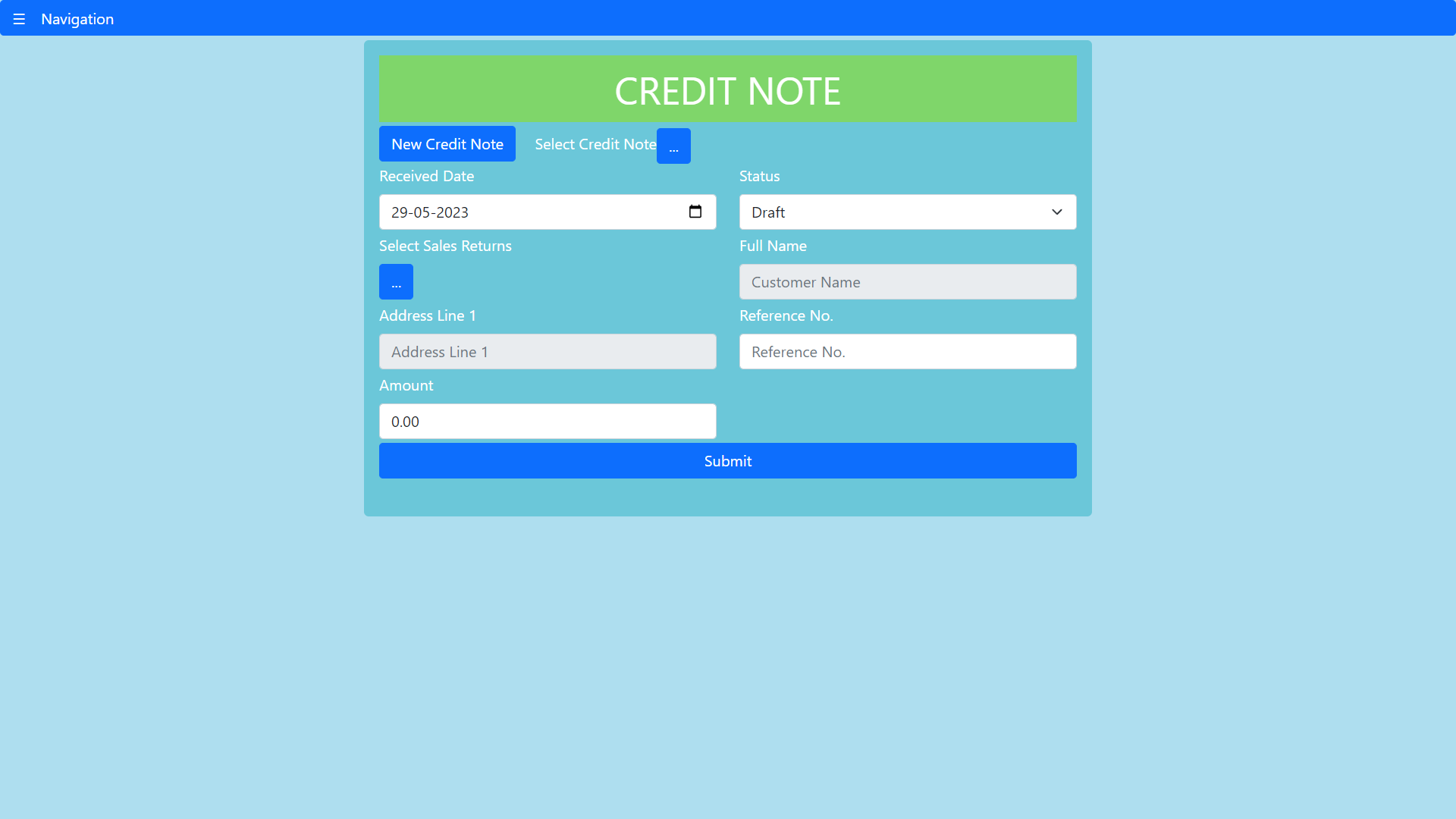
**Fig.18 – Sales returns data entry**

**(h) Credit Note**

A credit is a commercial document issued by a seller to a buyer. Credit notes act as a source document for the sales return journal. In other words, the credit note is evidence of the reduction in sales. So this form is used to create credit notes for customers on sales returns. The credit note is closed when the customer is refunded. Status includes draft, open and closed.

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. creditNoteID | Int32 |
| 1. creditNoteDate | date |
| 1. customerID | Int32 |
| 1. invoiceID | Int32 |
| 1. salesReturnID | Int32 |
| 1. amount | Decimal128 |
| 1. refNo | string |
| 1. status | string |



**Fig.19 – Credit note data entry form**

**(iii) Purchase Sub module**

**(a) Vendor**

Purchases of goods/services are made from vendors. The vendor form is used to enter the details about the vendor which includes contact information, the goods and services they sell and about the point of contact. The vendor details can also be edited by selecting the already inserted vendors using picklist.

*Database Schema*

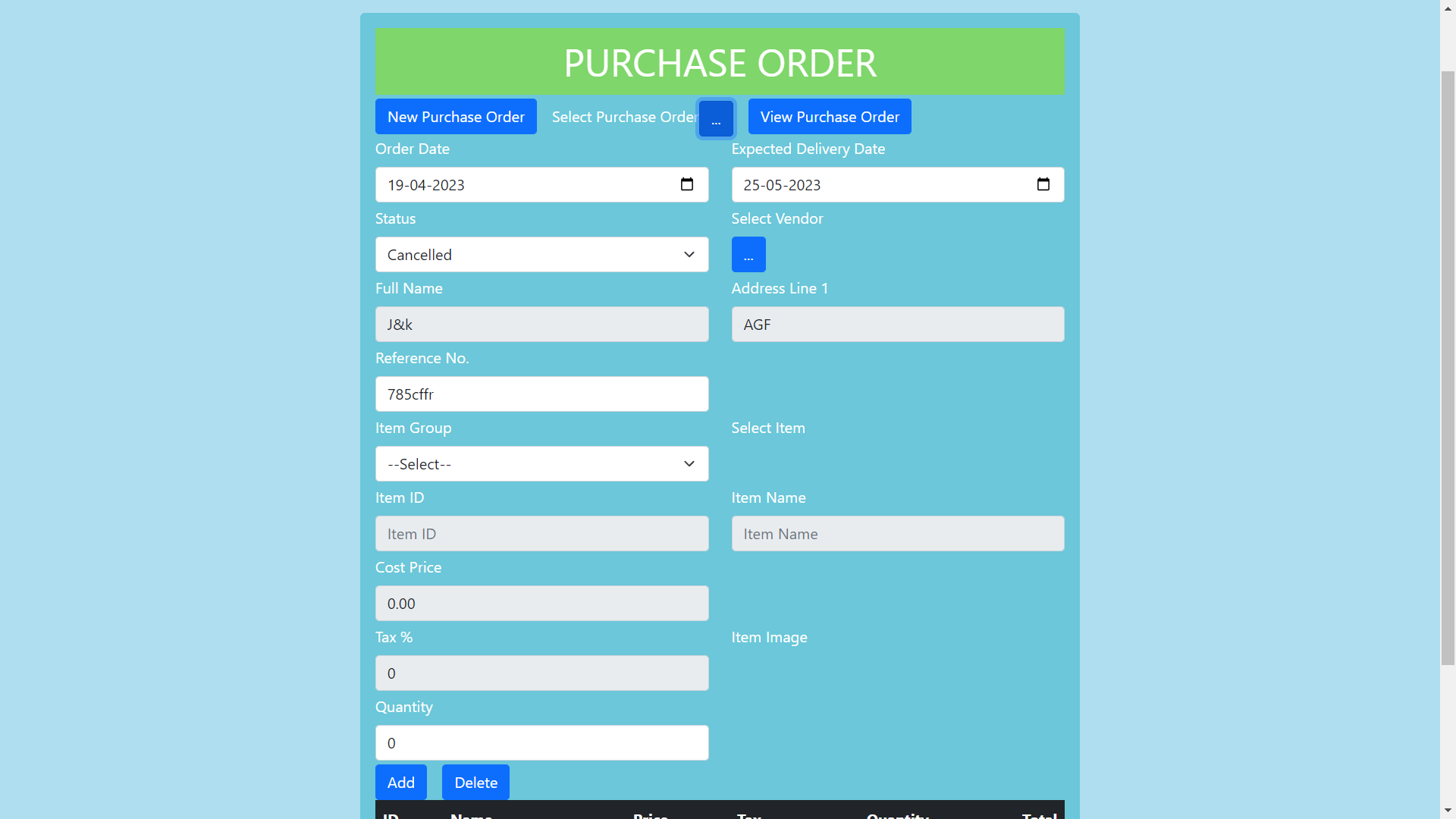
|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. vendorID | Int32 |
| 1. companyName | string |
| 1. goodsServices | string |
| 1. addressLine1 | string |
| 1. addressLine2 | string |
| 1. addressLine3 | string |
| 1. city | string |
| 1. state | string |
| 1. country | string |
| 1. emailID | string |
| 1. contactNo1 | string |
| 1. contactNo2 | string |
| 1. website | string |
| 1. pocName | string |
| 1. pocEmail | string |
| 1. pocContactNo | string |

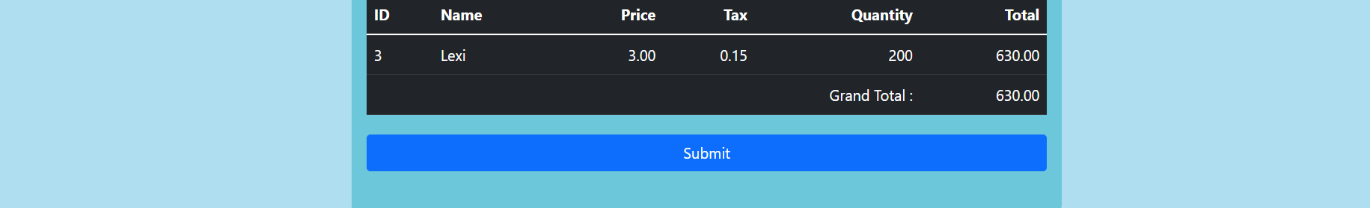


**Fig.20 – Vendor data entry form**

**(b) Purchase Order**

A purchase order (PO) is a commercial document and first official offer issued by a buyer to a seller, indicating types, quantities, and agreed prices for products or services. It is used to control the purchasing of products and services from external suppliers. In the data entry form, the vendor can be selected and also the items selected using the item selector component. The status of purchase order are draft, issued, received and cancelled.

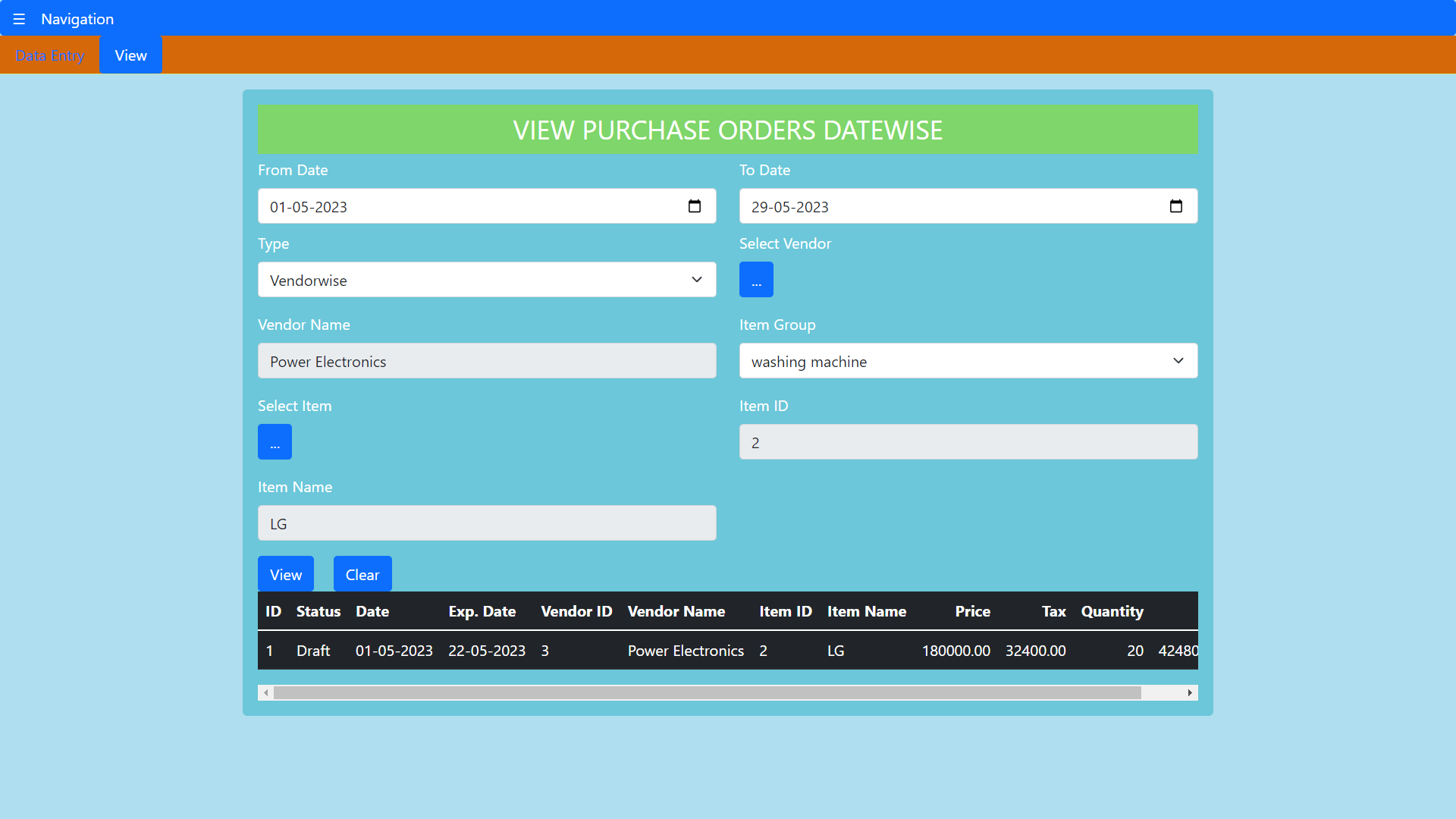




**Fig.21 – Purchase order data entry form**

*Database Schema*

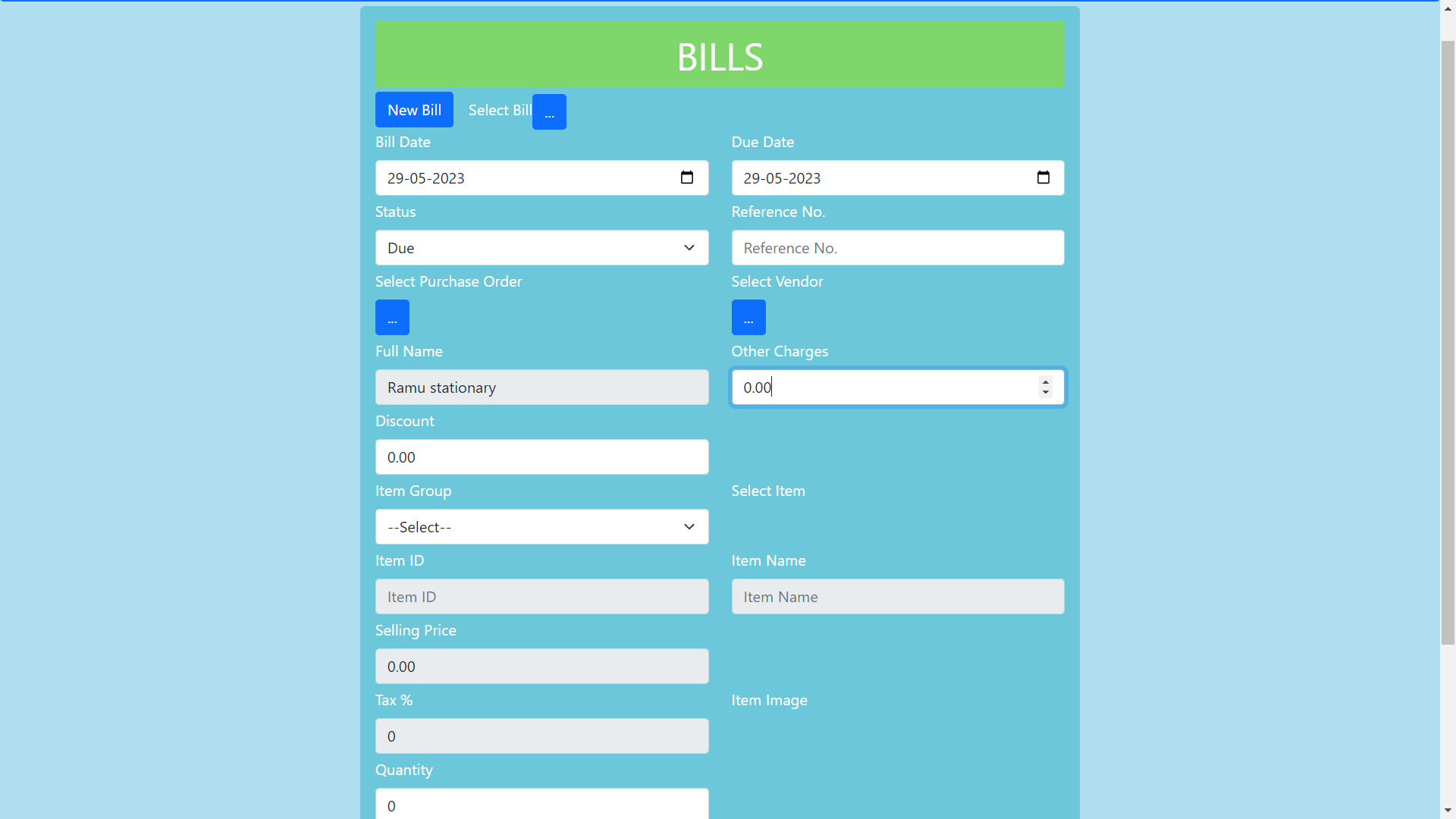
|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. purchaseOrderID | Int32 |
| 1. orderDate | date |
| 1. vendorID | Int32 |
| 1. expectedDate | date |
| 1. items | Array of documents |
| 1. refNo | string |
| 1. status | string |

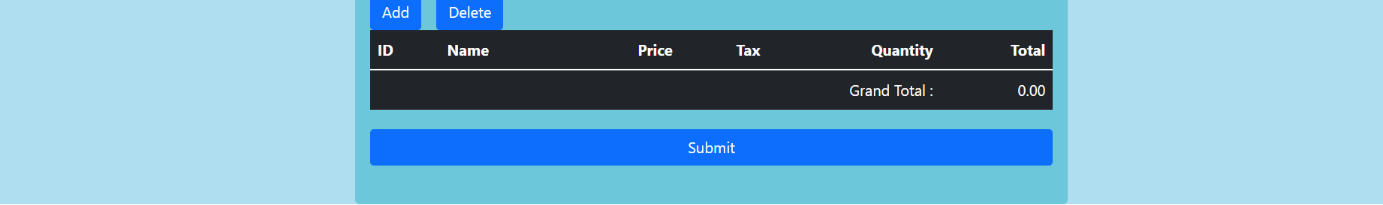


**Fig.22 – View purchase orders for a period vendorwise/itemwise**

**(c) Bills**

Vendor bills are requests for payment for products and services. Vendor bills might represent a bill for ongoing services, or they can be based on purchase orders for specific items and services. Bill details can be entered for a specific vendor by selecting vendor or for a purchase order by selecting the purchase order. Stock in is calculated using the status of bills. Bills status that can be set are draft, open, partially paid, paid, due and void. Quantity of products is taken as stock in from bills having status other than draft and void.





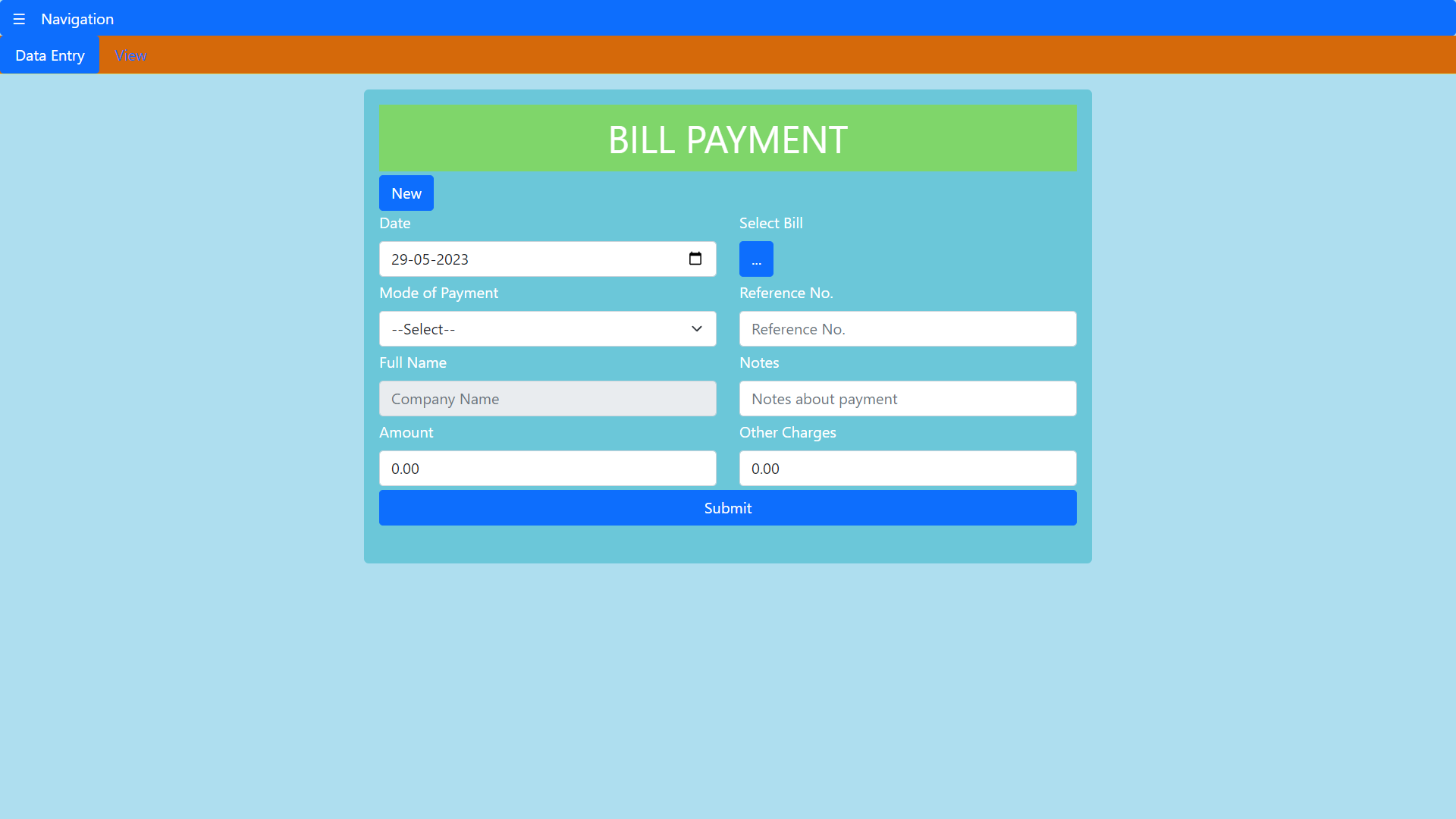
**Fig.23 – Bills data entry form**

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. billID | Int32 |
| 1. billDate | date |
| 1. vendorID | Int32 |
| 1. dueDate | date |
| 1. purchaseOrderID | Int32 |
| 1. items | Array of documents |
| 1. refNo | string |
| 1. status | string |
| 1. otherCharges | Decimal128 |
| 1. discount | Decimal128 |

**(d) Bill payment**

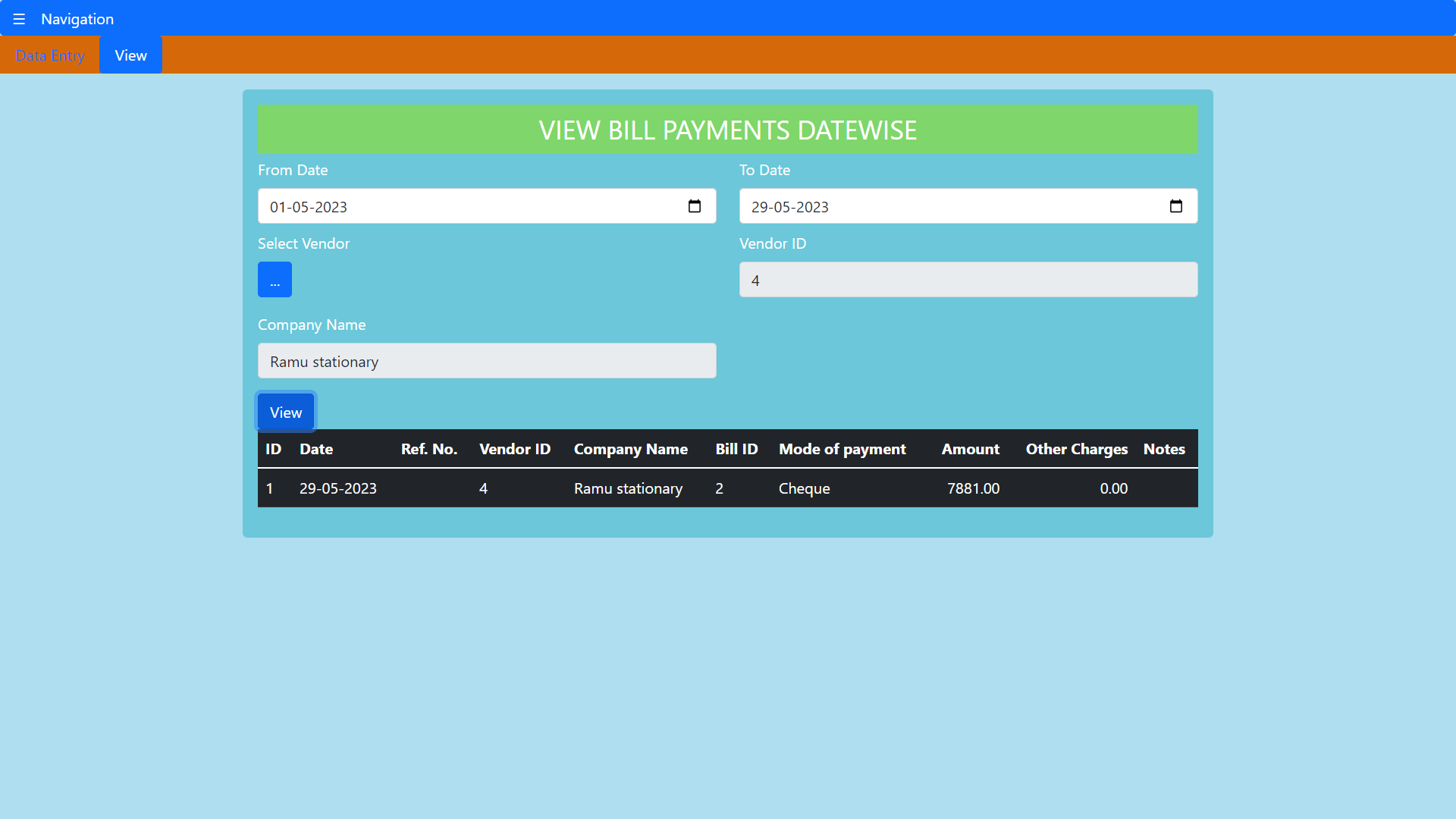
A bill payment is the last step in the purchase to pay cycle, when a company pays an outside vendor for purchased goods or services. The payment can be recorded in this data entry form. The bill has to be selected and amount will be automatically filled. Other charges can also be included like bank charges in the other charges field. Modes of payment are as same as in case of payments received form.



**Fig.24 – Bill payment data entry form**

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. billPaymentID | Int32 |
| 1. paymentDate | date |
| 1. vendorID | Int32 |
| 1. billID | Int32 |
| 1. refNo | string |
| 1. amount | Decimal128 |
| 1. otherCharges | Decimal128 |
| 1. modeOfPayment | string |
| 1. notes | string |



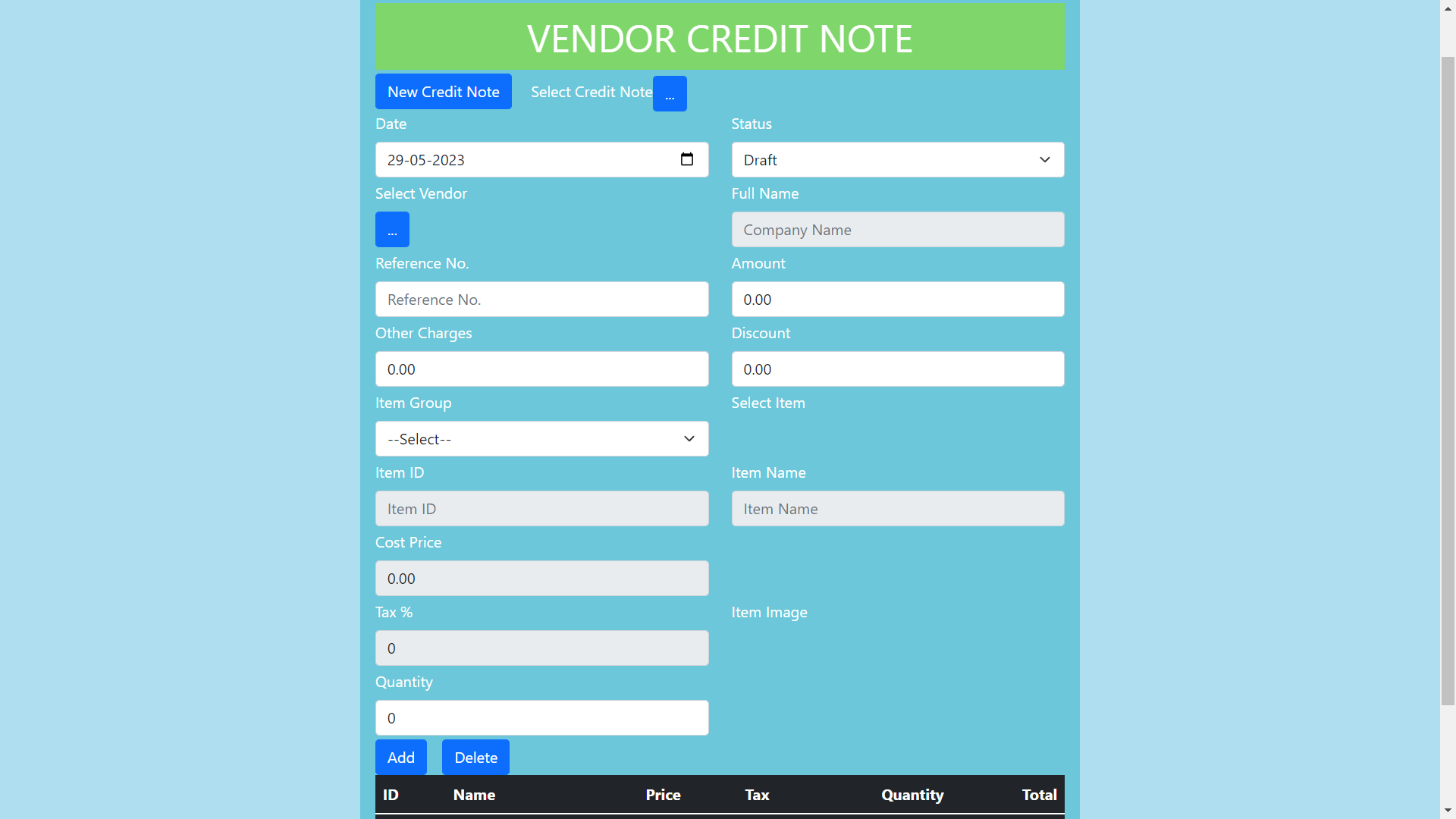
**Fig.25 – View bill payments over a period and vendor wise**

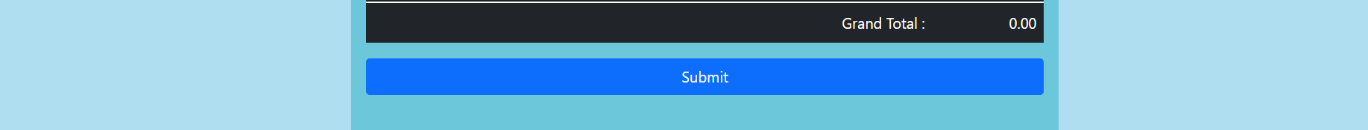
**(e) Vendor Credit Note**

A vendor credit note is an official legal document, just like an invoice or a purchase order, which suppliers provide to customers to notify the customer that credit is being applied to their account for any number of reasons. They could issue a credit note in case of purchase returns. Purchase returns can be calculated using this. The vendor credit note includes details about the returned items. The status that can be set include draft, open and closed.

*Database Schema*

|  |  |
| --- | --- |
| **Field** | **Data type** |
| 1. creditNoteID | Int32 |
| 1. creditNoteDate | date |
| 1. vendorID | Int32 |
| 1. items | Array of documents |
| 1. refNo | string |
| 1. amount | Decimal128 |
| 1. otherCharges | Decimal128 |
| 1. discount | Decimal128 |
| 1. status | string |



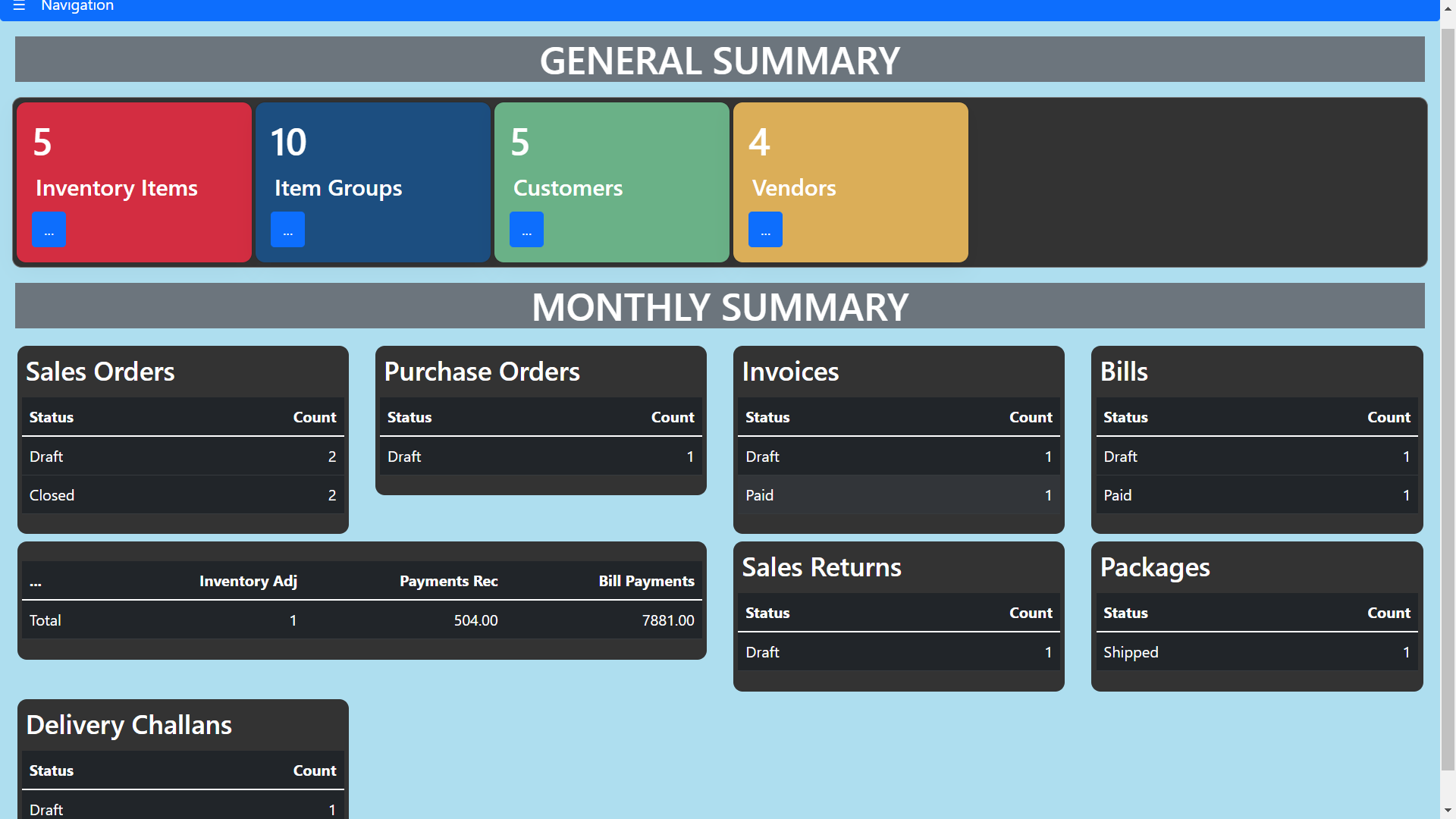


**Fig.26 – Vendor credit note data entry**

**(iv) Dashboard and Reports**

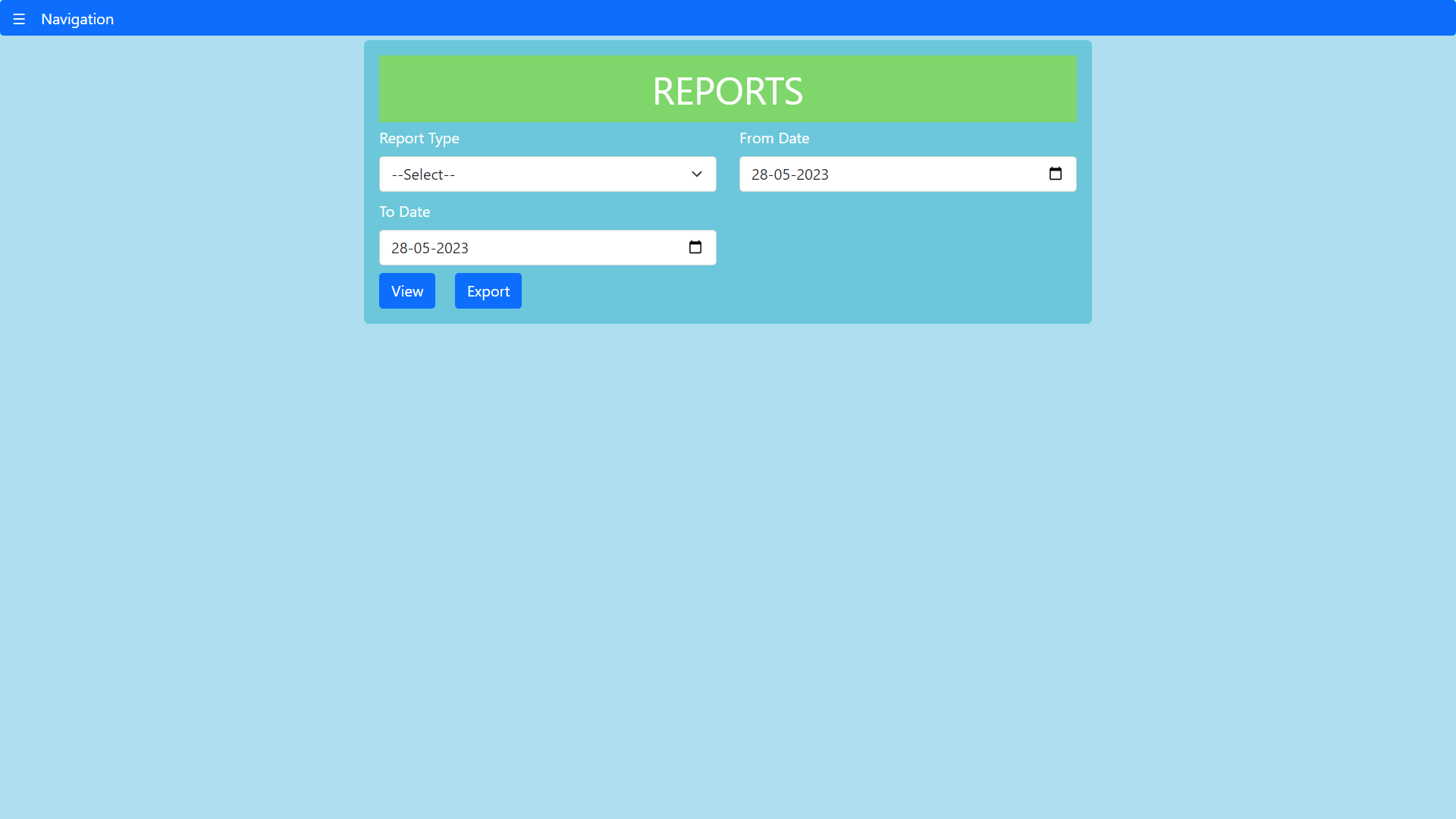
**(a) Dashboard**

This is the main landing page of this application. It shows various details about the other modules. It has got two sections namely general summary and monthly summary. General summary includes inventory items, item groups, customers and vendors. Item, vendors and customers can be selected to get their individual summaries. The monthly summary shows various numbers and statuses of sales orders, purchase orders, invoices, bills, sales returns, delivery challans, packages, inventory adjustments, payments received and bill payments. These all have links which when clicked will go to their respective data entry forms.



**Fig.27 – Dashboard**

**(b) Reports**



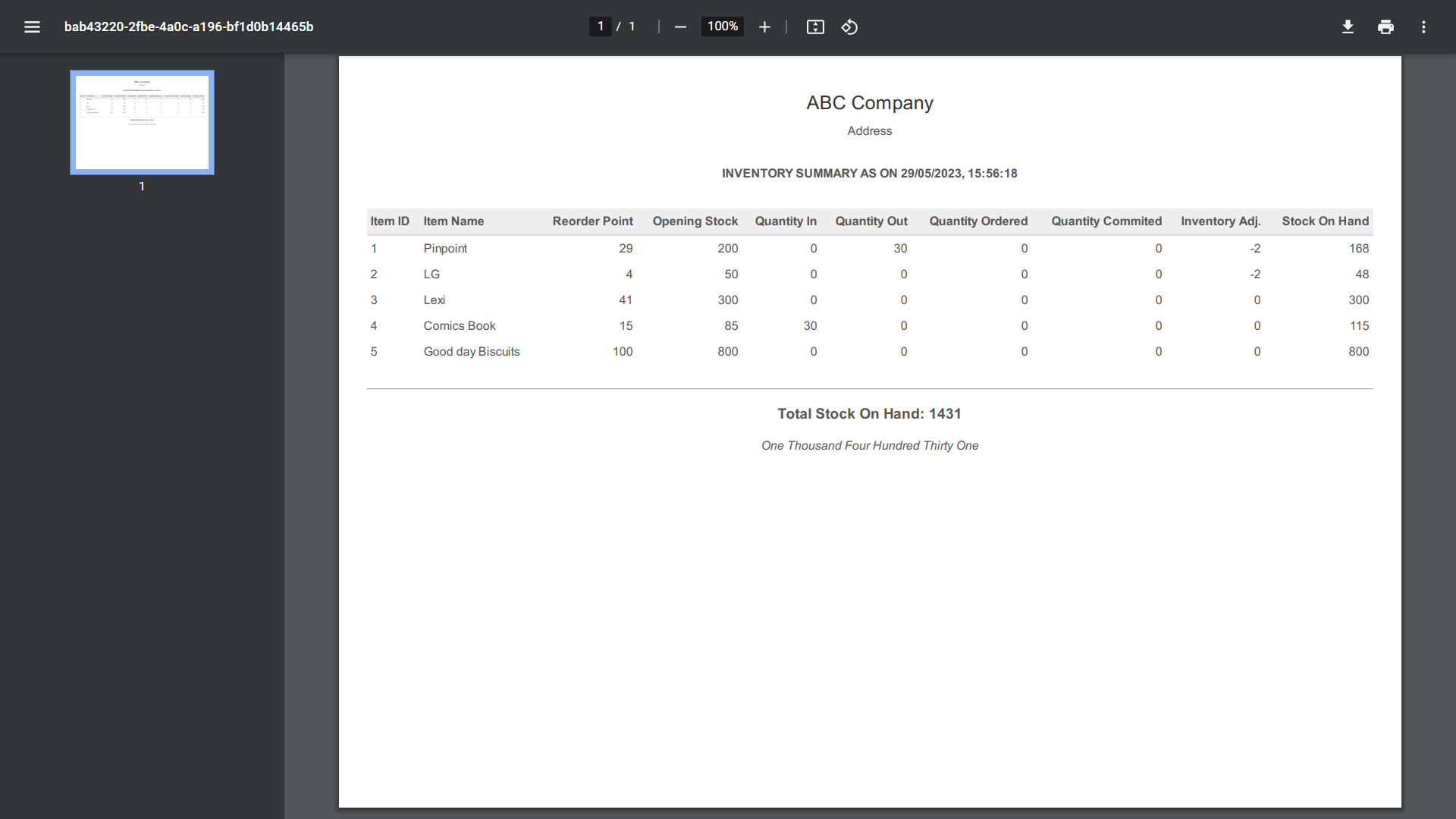
The major reports which can be viewed in pdf and data exported as excel include inventory summary, inventory aging summary, product sales report and sales by customer/item.

*Inventory summary report:*

This report gives details about all the goods sold and purchased. It also gives details about inventory adjustments, opening stock, reorder point etc. Stock on hand is calculated as

*Opening stock + Purchases + Sales returns – Sales - Purchase returns +/- Inventory adjustments.*

The inventory summary report is as shown in figure. Quantity ordered and committed are calculated based on the issued purchase orders and confirmed sales orders.

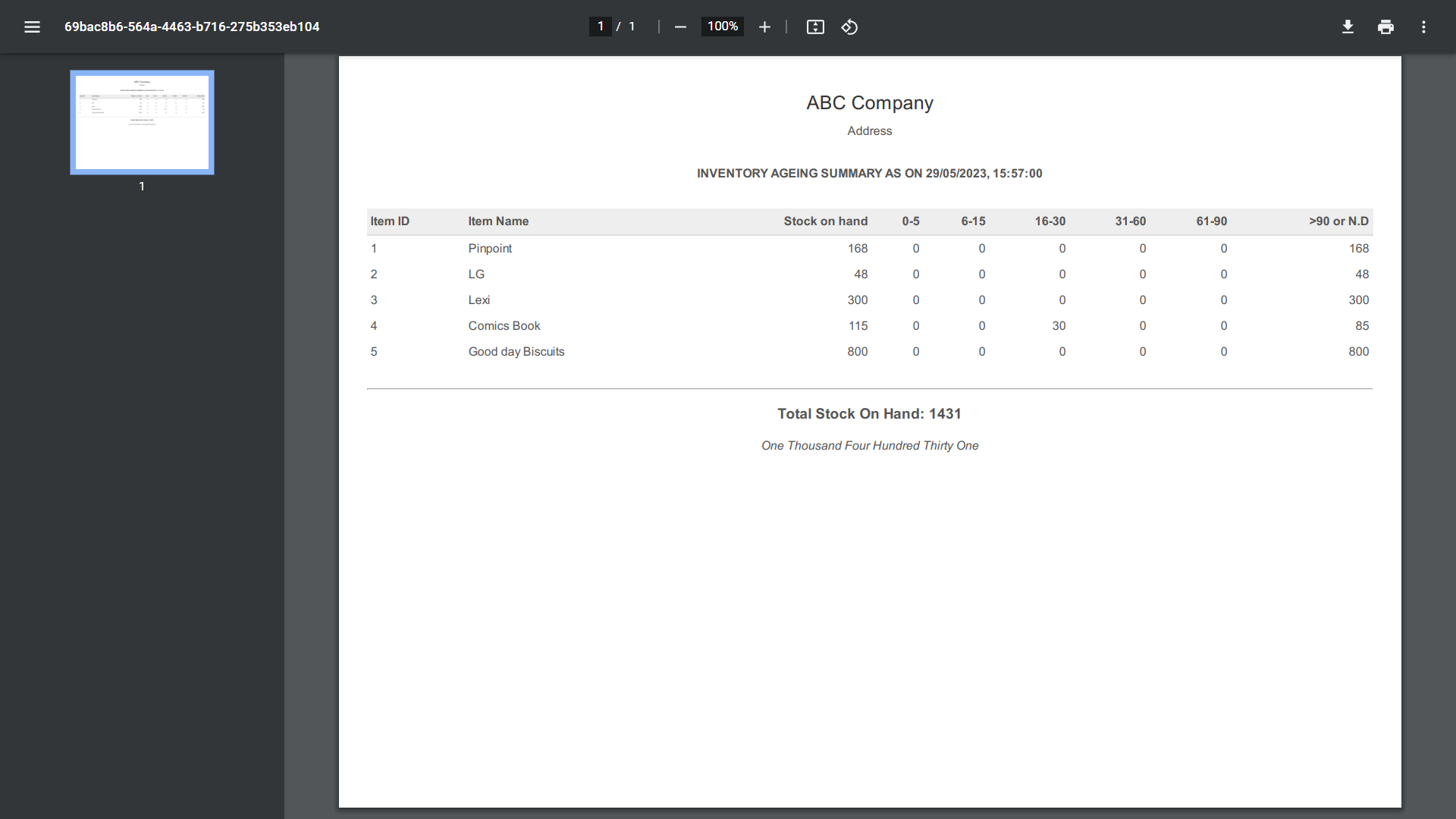


**Fig.28 – Inventory Summary Report**

*Inventory aging summary report:*

An inventory aging report, also called an aged inventory report or stock aging report, provides key metrics on how quickly your inventory moves. It's essentially a list of the items on hand grouped by the length of time in inventory. In this first stock on hand is calculated and then stock in (based on purchase bills) on each day is calculated starting backwards from today and is subtracted from the stock on hand till its value becomes zero.

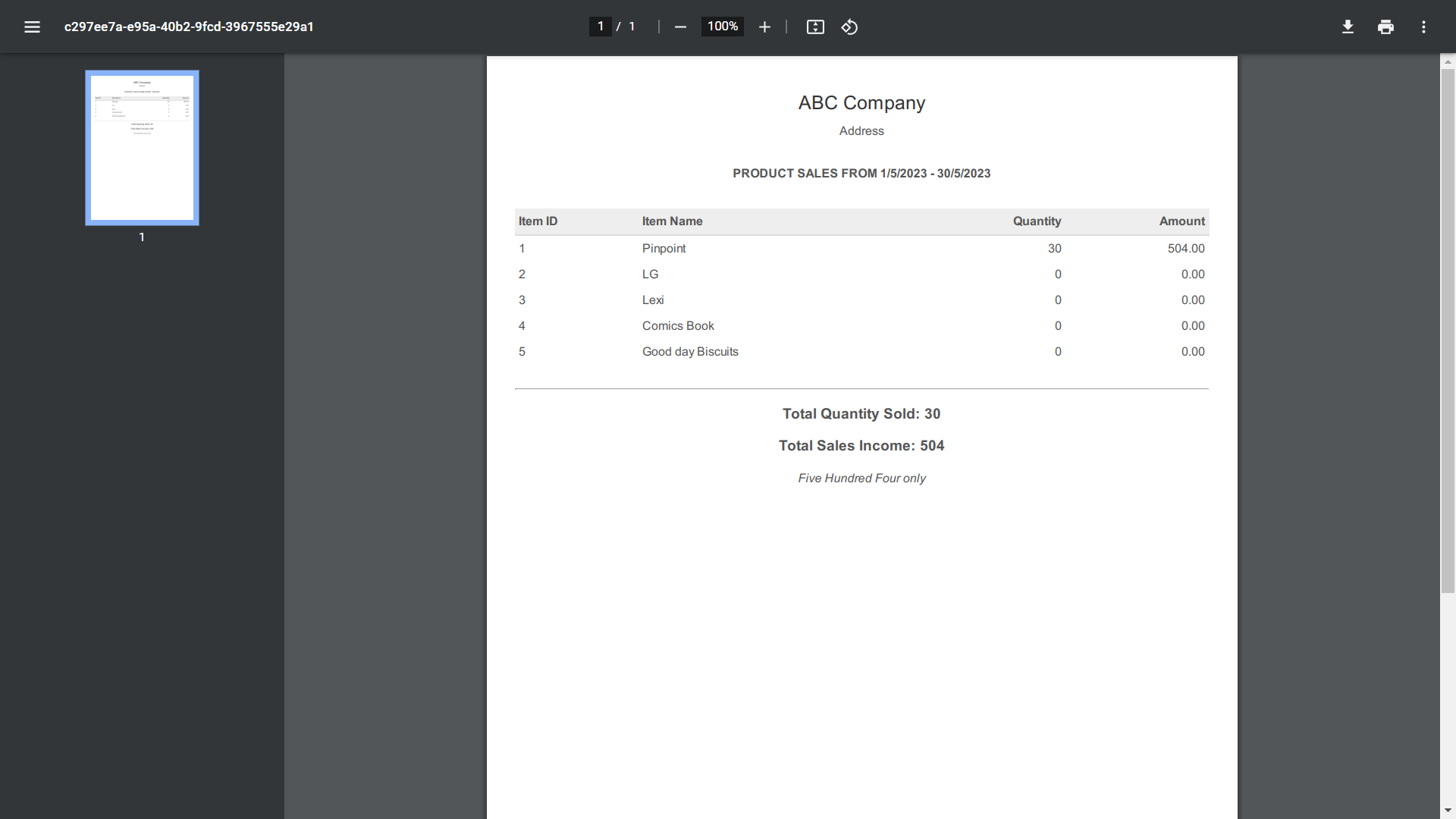
The report is as shown in the figure. Here the intervals of days are shown. ND means not determined, as there is not enough data to assess the age of the inventory.



**Fig.29 – Inventory Aging Summary**

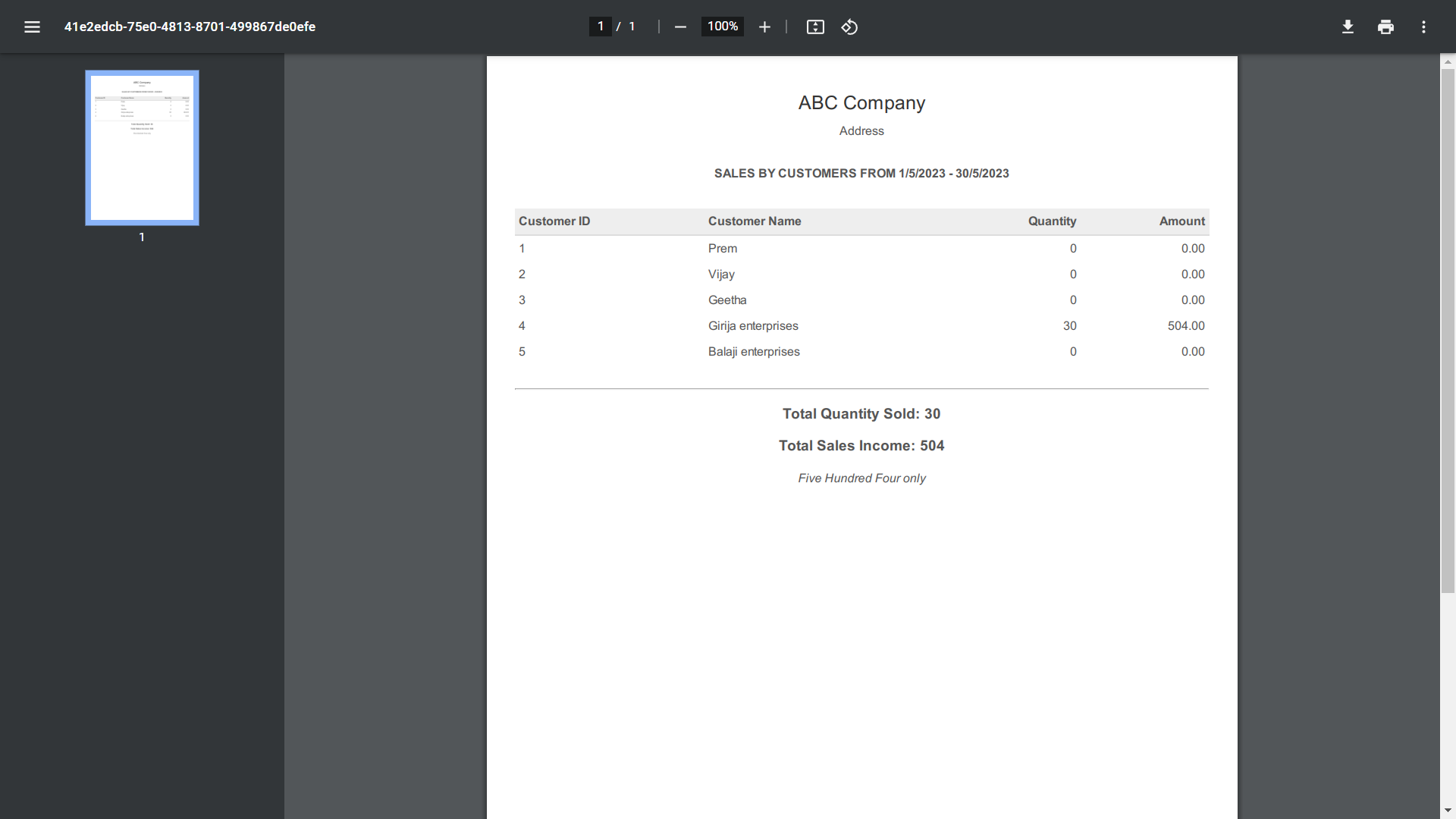
*Product sales report:*

This report shows the item sales by amount for a certain period. It is as shown in the figure below.

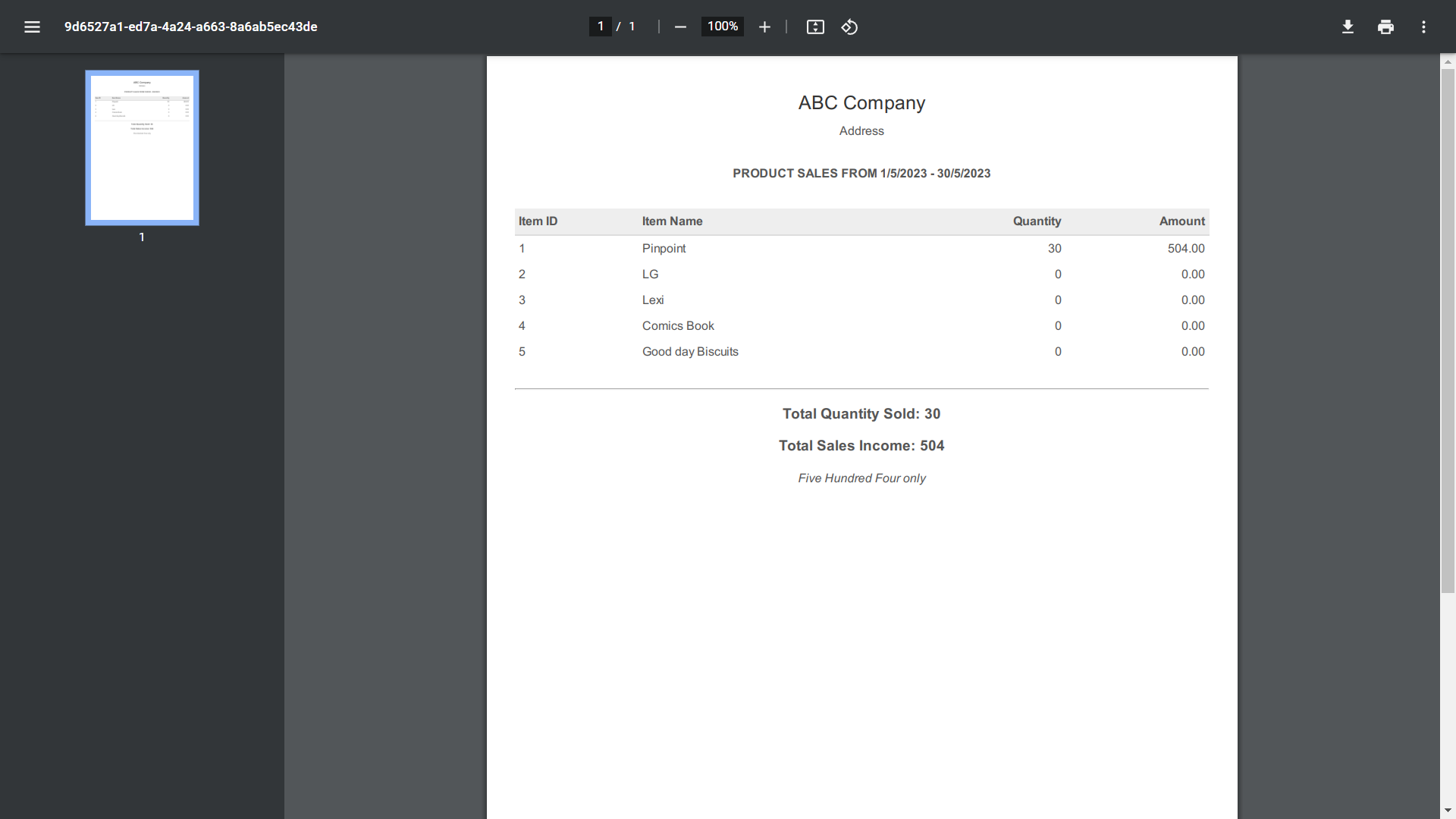
**Fig.30– Product sales report**

*Sales by customer:*

This report gives the total amount of item sales, customer wise. It lists each customer and the amount they have purchased.



**Fig.31– Sales by customer**



**Fig.31– Sales by item**

**Challenges & Opportunities**

Challenges:

1. Data Modeling: Designing an efficient and scalable data model in MongoDB for inventory management can be challenging, especially when dealing with complex relationships between entities, hierarchical data structures, and handling large volumes of data.

2. Performance Optimization: As the inventory data grows, optimizing database queries, ensuring proper indexing, and implementing caching mechanisms become crucial to maintain the performance and responsiveness of the application.

3. Real-time Updates and Synchronization: Implementing real-time updates and synchronization of inventory data across multiple devices can be complex. Handling concurrent updates, conflicts, and maintaining data consistency in a distributed environment requires careful consideration and implementation.

4. Security and Access Control: Ensuring data security, implementing secure authentication and authorization mechanisms, and managing access control for different user roles and permissions are critical challenges to address in an inventory management app.

5. Integration with External Systems: Integrating with external systems, such as payment gateways, shipping carriers, or third-party APIs, can pose challenges due to varying data formats, authentication mechanisms, and ensuring smooth communication between systems.

Opportunities:

1. Scalability and Performance: The MERN stack, when properly utilized, provides opportunities for building a highly scalable and performant inventory management app. Leveraging features like horizontal scaling, load balancing, and optimizing queries can handle increased data volumes and user traffic.

2. Real-time Analytics and Reporting: Using the MERN stack, it is possible to incorporate real-time analytics and reporting capabilities into the inventory management app. This empowers users to gain valuable insights, make data-driven decisions, and optimize inventory control and forecasting.

3. Mobile Accessibility: Leveraging the capabilities of React Native, a framework built on top of React.js, enables the development of mobile applications for both iOS and Android platforms. This provides an opportunity to extend the inventory management app's accessibility and reach to mobile users.

4. User Experience and Interface: React.js, with its component-based architecture, offers opportunities to create a rich and intuitive user interface for the inventory management app. Leveraging modern UI libraries, responsive design, and smooth interactions can enhance the user experience.

5. Ecosystem and Community: The MERN stack has a thriving ecosystem with a vast community of developers, libraries, and resources. This provides opportunities for knowledge sharing, finding solutions to challenges, and accessing pre-built components or plugins to expedite development.

By addressing the challenges and leveraging the opportunities, an inventory management app built with MongoDB, Express.js, React.js, and Node.js can deliver efficient inventory control, real-time updates, scalability, and an enhanced user experience.

**Risk Vs Reward**

**Risk:**

1. Technology Dependencies: Developing an inventory management app using a specific technology stack like MongoDB, Express.js, React.js, and Node.js involves a level of risk. As these technologies continue to evolve, there is a possibility of facing compatibility issues, version updates, or changes in the ecosystem, which may require adjustments or reworking of the app.

2. Learning Curve: If the development team is not familiar with the MERN stack, there may be a learning curve associated with understanding and implementing the technologies effectively. This could potentially lead to delays or the need for additional resources to mitigate the risk.

3. Security Vulnerabilities: With any software application, there is always a risk of security vulnerabilities. It is crucial to ensure that proper security measures, such as authentication, authorization, input validation, and secure communication protocols, are implemented to protect sensitive inventory data from unauthorized access or breaches.

**Reward:**

1. Scalability and Performance: The MERN stack, when used appropriately, offers scalability and performance advantages. MongoDB's flexible document-based model allows for efficient storage and retrieval of inventory data, while Node.js and React.js provide asynchronous and event-driven architectures that can handle concurrent requests and deliver a responsive user interface.

2. Development Efficiency: The MERN stack promotes code reusability and modular development, which can enhance the efficiency of development efforts. The ability to share code between the frontend and backend using JavaScript can streamline the development process and reduce time and effort required for implementation.

3. Real-time Updates and User Experience: The use of WebSockets or similar technologies in the MERN stack enables real-time updates and synchronization, allowing users to view and interact with inventory data in near real-time. This can enhance the user experience and improve overall operational efficiency.

4. Community Support and Resources: The popularity of the MERN stack ensures a large and active community of developers. This translates into a wide range of available resources, tutorials, and libraries, making it easier to find solutions to challenges or troubleshoot issues during the development process.

In summary, while there are potential risks associated with developing an inventory management app using the MERN stack, the rewards include scalability, performance, development efficiency, real-time updates, and access to a robust community. By addressing and mitigating potential risks, the rewards of using this technology stack can outweigh the challenges, resulting in a successful and effective inventory management application.

**Reflections on the internship**

This internship has been immensely valuable in providing hands-on experience with the MERN stack and executing industry-level projects. The project inventory module has been instrumental in enhancing my JavaScript programming skills and expanding my knowledge of crafting diverse MongoDB queries to meet specific project requirements. Although initially challenging, I enthusiastically embraced the task, delving into various online resources to acquire the necessary techniques. Overall, this internship has proven to be an enriching and worthwhile experience.

TCS iON has created an exceptional internship platform by offering invaluable references and mentorship from industry professionals. Their support has significantly contributed to the fruitful nature of this internship. The daily activity reports have fostered a sense of systematic working, enabling me to better organize tasks and meet deadlines. Consequently, this internship has instilled in me a strong work ethic and improved my time management abilities.

The experience gained throughout this internship will undoubtedly serve as a solid foundation for pursuing a career as a MERN full stack developer. I am optimistic that the insights and knowledge acquired during this program will propel my professional growth and open up new opportunities in my career advancement.

**Recommendations**

Although the internship experience has been valuable for my career advancement, I have identified areas that could have been improved to enhance the overall learning experience:

1. Increased mentor interaction: A higher level of engagement with industrial mentors would have been beneficial. Real-time communication channels such as voice chat would have allowed for more direct and immediate clarification of doubts or concerns, fostering a stronger mentor-mentee relationship.

2. Enhanced peer interaction: The level of interaction among peers during the internship was limited. Implementing group activities or discussions within the digital discussion room would have facilitated greater engagement among interns, promoting collaboration, knowledge sharing, and a sense of community.

3. Streamlined feedback communication: To improve the clarity of feedback, it would be advantageous to have a mechanism for responding to evaluators within the same format. This would enable a more comprehensive and constructive dialogue, ensuring a better understanding and implementation of the provided feedback.

By addressing these suggestions, future iterations of the internship program can create an environment that encourages robust mentorship, fosters active peer engagement, and facilitates effective feedback communication, further enhancing the learning experience and preparing interns for their careers.

**Outcome / Conclusion**

The primary goal of the industry project was to create a comprehensive automated software solution for inventory management within operations management, leveraging the power of the MERN stack. I am pleased to state that I successfully completed the majority of the essential modules outlined in the system requirement specification, employing the MERN stack effectively.

This experience has proven to be highly enriching, significantly enhancing my proficiency in utilizing the MERN stack for developing industry-grade projects. Through this project, I have gained valuable insights and hands-on experience, solidifying my capabilities as a developer in this technology stack.

**Enhancement Scope**

Here are some valuable suggestions to enhance the inventory management software:

1. User activity history: Implementing a feature that logs and tracks user activities would provide a comprehensive user activity history for future reference and analysis.

2. Multiple levels of user authorization: It would be beneficial to establish different levels of user authorization, granting individual access rights to various modules. This would enable customers to access data related to their orders, purchased products, and allow them to register complaints, enhancing their overall experience.

3. Bar code integration: Introducing bar code integration would expedite the process of entering items into the inventory list. By scanning bar codes, the system can quickly retrieve and populate item information, improving efficiency and accuracy.

4. Payment gateway integration: Integrating a payment gateway would streamline the recording of payments and receipts. This feature would automate the payment process, making it more convenient and efficient for both the business and its customers.

5. Shipping carrier integration: Collaborating with shipping carriers through integration would enhance the overall shipping experience. This integration would enable seamless communication and coordination with shipping carriers, resulting in improved service quality and customer satisfaction.

By incorporating these enhancements, the inventory management software will become more robust, efficient, and user-friendly, meeting the evolving needs and expectations of users and stakeholders.

**Research questions and responses**

**Research Questions:**

1. What are the key features and functionalities required for an inventory management application using MongoDB, Express.js, React.js, and Node.js?

2. How can the MERN stack architecture be effectively utilized to develop a scalable and efficient inventory management app?

3. What are the best practices for data modeling and database design in MongoDB to optimize inventory management operations?

4. How can real-time updates and synchronization be implemented using WebSocket or other technologies in the inventory management app?

5. What security measures and authentication mechanisms should be implemented to ensure data privacy and prevent unauthorized access in the inventory management app?

6. How can performance and scalability be enhanced in the inventory management app using caching, indexing, and query optimization techniques in MongoDB?

7. What are the available options and considerations for integrating barcode scanning functionality into the inventory management app?

8. What are the challenges and best practices for handling concurrent inventory updates and maintaining data consistency in a distributed environment?

Responses:

1. The key features for an inventory management app using MongoDB, Express.js, React.js, and Node.js may include product management, stock tracking, order management, reporting and analytics, user management, barcode scanning integration, supplier management, and sales forecasting.

2. The MERN stack architecture allows for seamless integration between the frontend (React.js), backend (Node.js and Express.js), and database (MongoDB). It enables building a single-page application with reusable components, server-side rendering, and efficient data handling.

3. Best practices for data modeling in MongoDB for an inventory management app involve denormalization, embedding related data for faster retrieval, and leveraging indexes for efficient querying. Considering the specific requirements and relationships between entities is crucial for designing an optimized database schema.

4. Real-time updates and synchronization in the inventory management app can be achieved by using technologies like WebSockets or server-sent events (SSE) to establish a persistent connection between the client and server. This enables instant updates on inventory changes across multiple devices.

5. Security measures in the inventory management app can include implementing user authentication and authorization using techniques like JWT (JSON Web Tokens), encrypting sensitive data, enforcing secure communication through HTTPS, and employing role-based access control (RBAC) to manage user permissions.

6. To enhance performance and scalability, techniques like caching (using tools like Redis), indexing relevant fields in MongoDB, and optimizing database queries can be employed. Additionally, horizontal scaling through load balancing and clustering can help handle increased traffic and ensure application stability.

7. Barcode scanning functionality can be integrated into the inventory management app by utilizing libraries and APIs that enable barcode scanning on mobile devices or by using dedicated barcode scanners that communicate with the application through appropriate interfaces or protocols.

8. Handling concurrent inventory updates and maintaining data consistency in a distributed environment can be challenging. Techniques like optimistic concurrency control, using atomic operations provided by MongoDB, and implementing proper transaction management can help ensure data integrity and prevent conflicts during simultaneous updates.

**Link to Code**

<https://github.com/PRESISPREMACHANDRAN/fsd_tcs_internship>

**Link to Demo Video**

<https://www.loom.com/share/2e925c2286644546b90422bef0e2c3c2>