ASHESI UNIVERSITY

IMPROVING ACCOUNTING PRACTICES OF SMALL AND MEDIUM SCALE ENTERPRISES IN GHANA THROUGH APPLICATION SOFTWARE

APPLIED PROJECT

B.SC. MANAGEMENT INFORMATION SYSTEMS

KWAKU KWAYISI BOOHENE

2020

ASHESI UNIVERSITY

IMPROVING ACCOUNTING PRACTICES OF SMALL AND MEDIUM SCALE ENTERPRISES IN GHANA THROUGH APPLICATION SOFTWARE

APPLIED PROJECT

Applied Project submitted to the Computer Science and Information Systems (CSIS) Department, Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Management Information Systems.

KWAKU KWAYISI BOOHENE

2020

DECLARATION

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date:

……

I hereby declare that the preparation and presentation of this Applied Project were supervised in accordance with the guidelines on supervision of Applied Project laid down by Ashesi University.

Supervisor's Signature:

……

Supervisor's Name:

……

Date:

……

Acknowledgements

I would like to thank God for the ability to complete this project. I would also like to exhibit my appreciation to everyone who helped in the undertaking and the completion of this project. I would like to thank my supervisor Mr David Sampah for his support and his feedback. Finally, I would like to exhibit my appreciation to the Computer Science and Information Systems Department of Ashesi University which has exposed me to the different aspects of Computer Science and equipped me with sufficient knowledge to complete this project.

Abstract

Small and Medium-scale businesses in Ghana contribute to the growth and development of the Ghanaian economy. It is, therefore, important that these businesses optimise their performance to generate more revenue and boost productivity to help the country develop. Generating and keeping accounting information is an essential way to track and improve a business' performance.

This paper presents a detailed design and implementation of an application which aids Small and Medium-scale Sole Proprietorships (businesses which are wholly owned and usually managed by a single individual) in generating and keeping accounting records and information for the analysis of business performance and, other business and financial purposes.

Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Background

Accounting refers to the process of recording and providing information about a firm's financial performance and its assets and liabilities for various and different users [12].

The information from accounting can be used by managers of a firm for controlling planning and decision making. Generally, there are two groups of users of accounting information; these are the external and internal users[12]. The external users like banks, investors and suppliers use accounting information to determine if they want to get into business with a firm or not. They also use it to evaluate how much they would put or invest in a firm. The internal users, like managers, use accounting information to plan, direct and control business affairs. For these reasons, it is important that every firm takes account of their business engagements and transactions. According to the Partnership Act, 1962 and the Companies Act, 1963, companies and partnerships are required by law to take accounts of their books and transactions and to perform general external audits. However, the Business Act,1962 does not require sole proprietorships to do the same, and so most of these businesses avoid taking accounts of their books and transactions. A case study performed by [3] proves that about 65% of Small and Medium Scale Enterprises (SMEs) do not keep any accounting records or perform any form of accounting for their business. Out of this 65%, about 90% of the businesses selected were sole proprietorships. As such, these businesses do not have access to records and figures which they can use to analyse their business' performance and plan for their business based on the results gained from these records. Some of the reasons highlighted for the poor accounting practices in Ghana, according to [3] are:

- Accounting requires technical knowledge,
- Accounting is time-consuming, and
- Accounting requires additional staff (the hiring of an accountant)

1.2 Proposed Solution

To attempt to resolve the problem of poor accounting practices by SMEs describe above, a software specifically designed for the Ghanaian entrepreneur, which simplifies the collection and the presentation of accounting information for small-scale sole proprietorships in Ghana was built. Functionality and specifications were based on the requirements gathered from target users during the requirements analysis phase.

1.3 Related Work

The need for accounting information and tools in the running of a business is ever more pressing as studies show that over 60% of SMEs fail in their first five years of operation [3].

In Ghana, before an SME can access a loan from a financial institution like a bank, the financial institution usually audits the business' financial information and position [19]. The financial institutions do this is to convince themselves that these businesses will be able to pay the loan back with interest [19]. The banks however, find that most SMEs do not prepare or keep financial statements and hence have no information that they can audit [19]. In situations where they do prepare these statements, they are not prepared well and seem to be unreliable as some transactions are hard to trace since they were not recorded [19].

In a study conducted by Amidu and Effah [2] to evaluate E-Accounting practices in Ghana, a sample of 58 businesses was selected. Out of these 58 businesses, 76% of

businesses had Chief Executive Officers (CEOs) with university degrees or certificates higher education. And out of this 76%, only 22% of these CEOs had professional training in finance and accounting. This validates the research done by Amoako [3], which concludes that most SMEs in Ghana do not keep accounting records because they feel there is no need to. According to Amoako [3], other reasons people do not keep accounting records are because accounting is time-consuming, expensive, requires technical knowledge, and exposes your financial position.

The purpose of the proposed accounting app (which will be subsequently referred to as "My Accounts") built through this project is to make accounting less time consuming, less expensive, easy to perform without any technical knowledge. The app also seeks to improve the financial literacy of the users of the system, who already realise the benefits of accounting to their business.

A study by Amidu, Effah and Abor [2] states that the most common accounting software used by small and medium scale enterprises in Ghana is Microsoft Excel, which is spreadsheet software [9] that was not purposely created for accounting purposes but can be used as such. The study also concludes that most of these SME's hire an external accountant to generally take accounts of the business' affairs.

There are some accounting systems available to the general market, like QuickBooks[26]. Accounting software can be quite difficult to utilise as they have a lot of functionalities and can be quite complex. According to Lu, Fu, Gu et al.[22] current accounting software on the market is quite complex and would usually require a professional accountant to use and navigate. Oze [27], a fintech start-up in Ghana has created an Oze app for accounting purposes, but primarily only focus on drawing up invoices and tracking of expenses and sales of businesses. The app does not produce any financial statements like an income statement or a statement of financial position which makes it easier to analyse and assess

business performance [12].

1.4 Project Summary

In summary, the goal of this project is to design and build an accounting software for sole proprietors and small business owners in Ghana. The proposed accounting software is therefore supposed to help businesses

- Keep track of their expenses, revenue and profit
- Create reports and financial statements for the user.
- Keep track of whom they owe and who owes them.

Chapter 2: Requirements

This chapter provides a detailed overview of the different requirements of the proposed software. This chapter also describes the target users of the proposed software and the different use case scenarios with respect to the proposed software.

2.1 Project Overview

The accounting software would run on web platforms. The system will allow users to easily input, view, manage and analyse their accounting information on any device which has an internet browser.

2.2 User Characteristics

The accounting software will be suitable for a user who has the following attributes:

- The user is a sole proprietor or a freelancer who does not have access to enough capital to hire an accountant to perform bookkeeping and other accounting functions.
- The user has basic high school education in order to understand the words and terms that he or she engages within using the system to enhance usability.
- The user has access to a smartphone, personal computer or any device with internet access and a web browser in order to use the system.
- The user has prior knowledge of the importance of accounting and how it can benefit their business. The user is also opened to learning more about accounting through facts and tips, which may help the user flourish in their business.

2.2.1 Use-case scenarios and Diagrams

- *A business owner (tracking his expenses)* - A business owner has been struggling to keep track of his expenses. He usually records his expenses in an exercise book, but due to the cluttered nature of his workspace, this book tends to get lost often. He opens the myaccounts app, creates an account and stores his expenses using the app. He uses the graphs to see if he is spending more money on his business this month in comparison to previous months and can identify what is responsible for the differences in monthly expenses.
- *Business owner (Going for advice from a business consultant)* - A business owner is looking to expand his business and so goes to a business consultant for advice. The business consultant asks the business owner for his business' accounting records to paint an accurate picture of the performance of the business. The business owner then opens the "My Accounts" web app and opens his most recent income statement to show the business consultant. The business consultant then proceeds to ask the business owner follow up questions to get more information on the business and to understand the income statement more accurately. The business consultant proceeds to give well-informed advice to the business owner on what he can do to grow his business.
- *Business owner (Going for a loan from a savings and loans company)* - A business owner is looking for money to expand his business. He goes to a savings and loans company which requests for any accounting information in order to grant him a loan for him to expand his business. He opens the "My Accounts" app and shows his most recent income statement as proof that his business is profitable and will be able to pay back the loan.

Below is a use case diagram (Figure 2.1) to illustrate the different use cases for this accounting app.

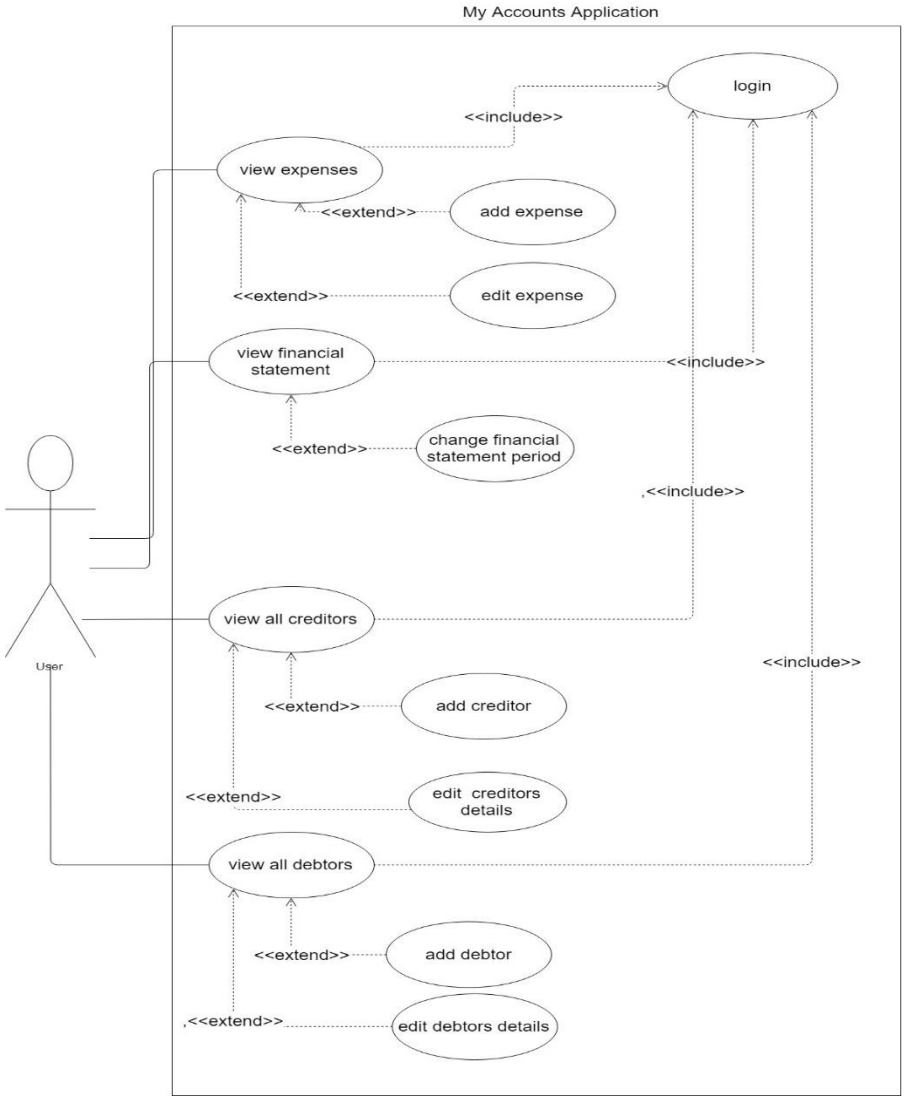

Figure 2.1: Use-case diagram of the accounting system

2.3 System Requirements

Below is a list of features and functions that the accounting software will perform. These features and functions have been divided into functional and non-functional requirements.

2.3.1 Functional Requirements

- The system will allow users to input the names of their debtors and their creditors, which is essentially those they owe as a result of the business and those who owe them. The system would essentially track the amounts they owe, the amounts they are owed and when all these payments are due. In accounting, individuals or businesses who owe the business are recorded as account receivables and businesses or individuals, who a business is in debt to are recorded as account payables[12].
- The system will allow users to input business transactions and keeps track of these transactions, and classifies these transactions under sales, expenses and profits. The system would graph these business transactions to help users analyse the performance of their business.
- The system produces an income statement for users for a given period.

2.3.2 Non-Functional Requirements

- The system should be secure. As the system contains the accounting information of users, these details should be encrypted and stored securely.
- The system should be user friendly and easily usable by the target users.

Chapter 3: Architecture and Design

In this chapter, a detailed overview of the architecture and the design of the proposed application is given. Diagrams which describe how the application's architecture will also be illustrated and explained in this chapter.

3.1 General System Overview

The choice of system architecture design types proposed for the application is influenced by the following reasons:

- Security: The selected system architecture should enhance the application's security and should protect the data of users. The architecture should make it difficult for individuals with malicious intent to perform system attacks such as database injections.
- Modularity: The selected system architecture should be made of different layers or modules. This should make it easier for further improvement of the different aspects of the system It would make scaling easier, should there be the need.

In considering these factors, the type of system architecture type proposed for the design of the system is a 3-tier client-server architecture; Reasons for this would be further elaborated in the forthcoming sections of the chapter as these architecture types are explained.

3.2 3-Tier Client-Server Architecture

A 3-tier client-server architecture pattern breaks down a computer system into three different layers, each with a different responsibility [13]. These are:

- The presentation layer, which formats and displays data to user of the system.

- The application layer, which handles the business logic of the system. (The business logic of a system refers to the possible sequences and ways in which the system uses, manipulates and processes data [15])
- The data layer, which stores data and information being used by the system [13].

In the proposed system, the presentation layer can be accessed through a web browser. The web app would be made progressive to adapt to different screen sizes as the app may be used by users who have smartphones, tablets or/and traditional personal computers like desktop and laptop computers.

The application layer, which controls how data in the system is created, stored, used and manipulated will be built in the form of a Representational State Transfer (REST) Application Programming Interface (API). An API is an interface which allows the interaction and communication of two different software applications [18]. Representational State Transfer (REST) is an architectural style for creating Web service software and technology [17]. In this system, the RESTful API will take user data inputs keyed in through the presentation layer and upon verification and interaction with the data layer, will return a response in the JavaScript Object Notation (JSON) format which will be formatted for display by the presentation user.

The data tier, which handles the storage of data, would be built using a relational database. A relational database is a type of database which stores data in tables, columns and rows [33]. In a relational database, data items have predefined relationships between them [33]. Relational databases are ideal for storing data in the form of text (that is letters, numbers, and characters) as opposed to multimedia and documents (pictures, videos and gifs) [7]. The numeric nature of accounting data makes this type of database ideal for data storage in this system. The accounting details also have relational data. For example, your expenses

and sales must be tied to your user id. This type of data is stored in a more appropriate format in relational databases [7].

Below is a diagram (Figure 3.1) illustrating the 3 Tier Architecture of this proposed web application.

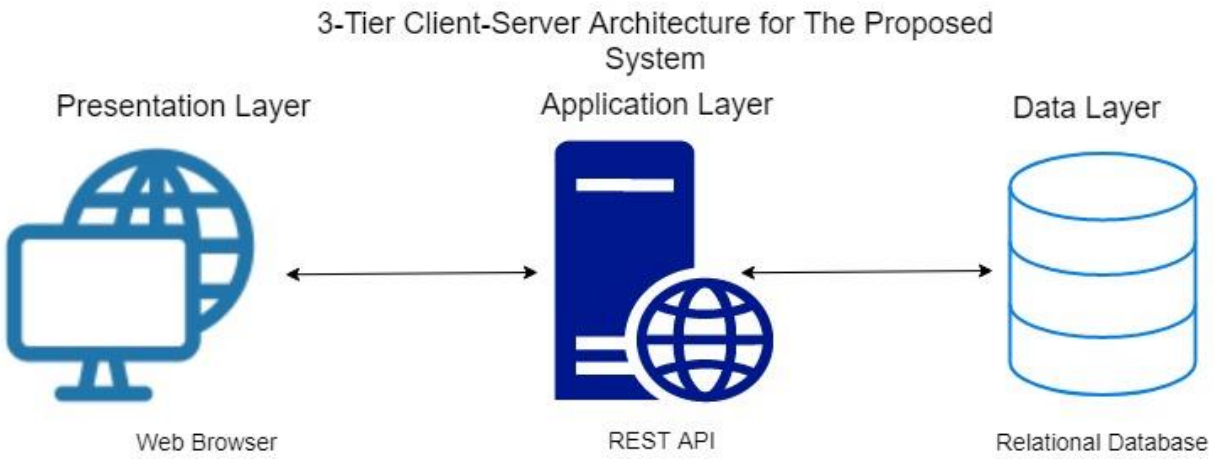

Figure 3.1: Diagram of the 3-Tier Client-Server architecture for the accounting system

3.3 Database Design

The proposed relational database design for the system comprises of six tables, namely: “expenses”, “expenses_type”, “sales”, “creditors”, “users” and “debtors”. Below is a table illustrating the different tables in the proposed database and a description of what kind of data it stores:

Table 3.1: Table showing the structure of the relational database.

Table	Description
users	This table stores the user's details and information. Details include the user’s email, password, first name and last name.
expenses	This table stores the details of the expenses that a user has logged.

Expense_type	This table stores the different type of expenses in order to classify expenses. The classification of expenses into production expenses, administrative expenses, selling expenses and miscellaneous expenses makes it possible for the system to produce an income statement for the user at the end of a given period.
sales	This table stores the details of every sale that a user logs in.
debtors	This table stores the details of clients and businesses that owe the user money.
creditors	This table stores the details of partners and suppliers which the user owes money to.

Every table aside the “expenses_type” table is connected to the “users” table through a user_id foreign key which references the id primary key of the users table. The expenses table has an additional foreign key expense_type which references the expense_type table. Below is an Entity Relational (ER) Diagram of the proposed database design for the system which gives details on, the attributes and datatypes of the columns in the different tables of the database. It also gives details on the type of relationship which the different tables have with each other.

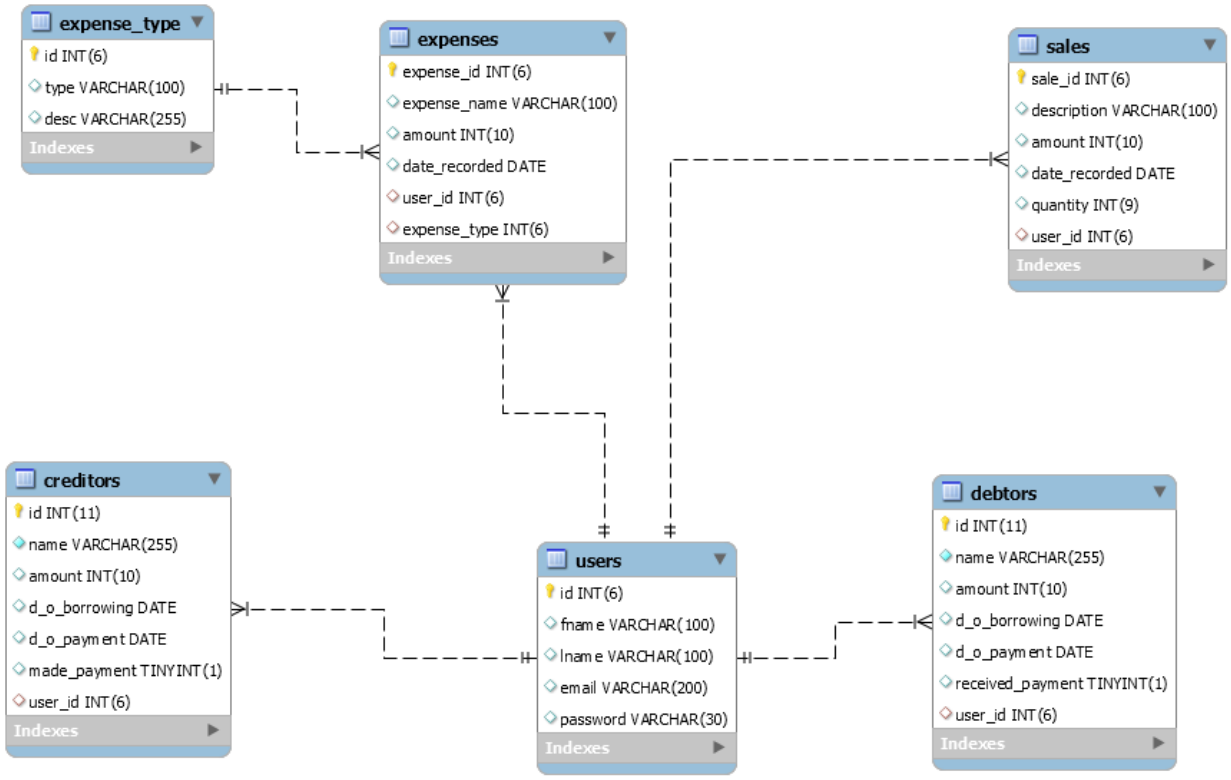

Figure 3.2: ER diagram illustrating the database design for the accounting system

3.4 Activity Diagrams

The diagram below shows the different activities a user would go through to view and manage his expenses.

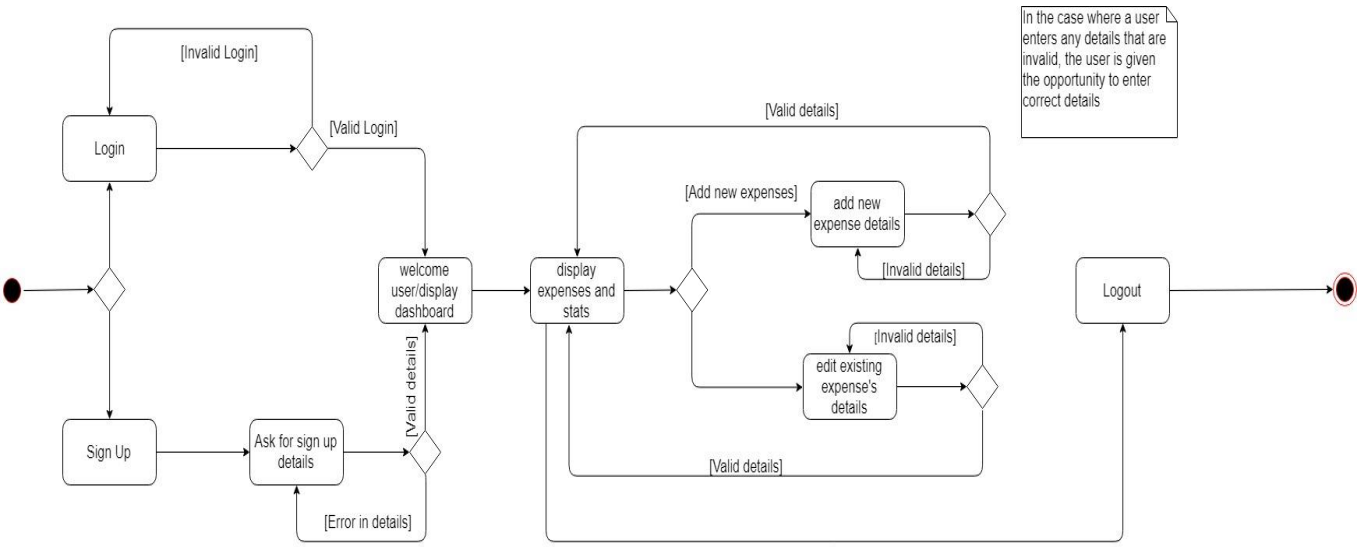

Figure 3.3: Activity diagram illustrating different activities involved in users managing their expenses with the accounting system.

The user undergoes a similar series of activities to manage his sales, list of debtors and list of creditors. Below is another diagram which illustrates the steps a user undergoes to view his income statement.

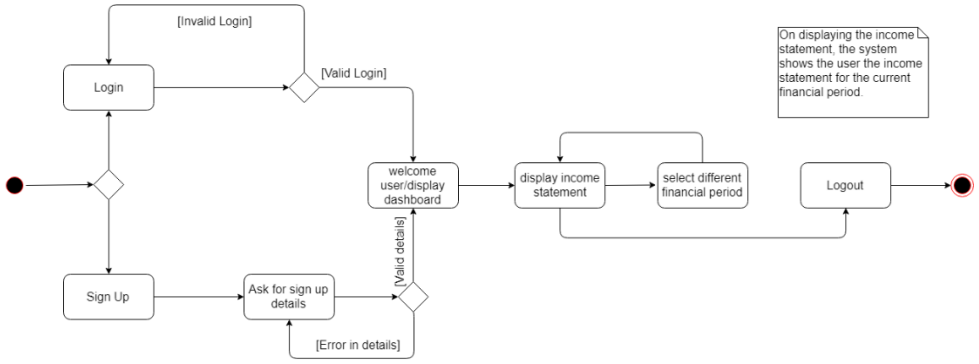

Figure 3.4: Activity diagram illustrating different activities involved in viewing their income statements with the accounting system

Chapter 4: Implementation

In this chapter, a description of the implementation technique, as well as a description of the different tools, libraries, APIs, frameworks and components used, will be detailed. Evidence of implementation in the form of screenshots of the code written and the different interfaces built, will also be presented.

4.1 Technology, Libraries and Tools Used

- JavaScript: JavaScript is a web programming language that allows for the dynamic display of information on web sites [10]. In the modern development of web applications, JavaScript can be used for both server-side and client-side programming as the language continuously evolves [10]. In the implementation of this application, JavaScript was used to build both the client-side (with React. Js) and the server-side (with Node.js and Express.js).
- React.js: React is an open-source JavaScript web framework created by Facebook to make it easier for web developers to create dynamic web apps, as some find interacting with the Document Object Model of HTML (HyperText Markup Language) pages unfriendly and difficult [6]. The implementation of a Virtual DOM in React optimises the overall performance of web applications [1]. For example, components which have been rendered are not re-rendered unless changes have been made to those components[1]. Hence if you load a header in a web application, that header would not need to be loaded again hence saving time and space.
- Cascading Style Sheets (CSS): "CSS is one of a series of standards formulated by the W3C (Worldwide Web Consortium) in December 1996" [16]. Web developers use CSS to beautify and organise the content of web pages. In this application, CSS was used to format and beautify the different components of the application.

- Bootstrap: Bootstrap is a CSS framework created by Twitter, which provides a set of JavaScript functions and CSS classes to make web development easier and faster [4]. It allows developers to make cross-browser websites and web applications [4]. In this application, bootstrap was used to create a standardised interface across different browsers and devices.
- Axios.js: Axios is a JavaScript library which can be used to make a HyperText Transfer Protocol (HTTP) request to an API's endpoints in order to send and receive data from the API [32]. In this application, Axios.js was used to make HTTP requests to the API, for interaction with the database.
- Chart.js: Chart.js is a JavaScript library for data visualisation in the form of different charts and diagrams on a web page [8]. Chart.js was used in the application to create chart components through which a user's accounting data can be visualised.
- Node.js: Node.js is a runtime environment that executes JavaScript code outside a web browser [20]. This serves as a platform to build networking applications like APIs using JavaScript. Server-side programming for this application was written in JavaScript code and executed using Node.js.
- Express.js: Express.js is a JavaScript web framework that can be used to build server-side web applications and APIs for Node.js [29]. In this application, Express.js was used to build an API with REST endpoints for manipulation of data.
- MySQL: MySQL is a relational database management system that stores data using tables with rows and columns[14]. This system uses SQL queries to create, read, update and delete data stored in the relational database[14]. For data storage in this application, a MySQL database was set up and used.
- WAMPserver (WAMP): WAMP is a software development stack package that allows software developers using personal computers running the Windows operating system

to create a virtual web server for the developer to build their application on their client computer [5]. This package contains Apache, PHP and MySQL software.[5]

- Postman – Postman is an application for testing the different endpoints of APIs [31]. With Postman, developers can test HTTP requests and see the responses they get from an API [31]. Postman was used to test the different endpoints of the REST API.
- Git: Git is a version control system designed to store and track changes in code that developers make in the process of creating an application [21] . In this project, Git was used to track any changes and to store and back up all programming files on an online repository on github.com.
- Visual Studio Code: Visual Studio Code is an Integrated Development Environment (IDE) produced by Microsoft for programming and software development [11]. It has extension and plugins that aid in the programming by highlighting syntax errors and making corrections in code [11]. It also has plugins which can integrate language compilers and runtime environments for more efficient programming [25]. It was used to write and build both the client-side and server-side of this application.
- MySQL WorkBench: This is a database design tool suited for the creation and manipulation of MySQL databases [23]. It has an in-built IDE which was used to write SQL queries to construct the database manipulate data in the database.

4.2 Hardware and Software

In order to utilise the app, the user should have a device with a web browser which supports React.js web applications. Examples of such web browsers are Internet Explorer 9 and above. Microsoft Edge, Google Chrome, Firefox and Opera browser. The application layer of the app was also stored on a web server which supports Node.js. Node.js has compatibility with Windows, Linux and Macintosh Operating systems.

4.3 Implementation Techniques

In this section of the chapter, the different techniques and programming styles used in the implementation of the application will be detailed.

4.3.1 Component-based Development

Component-based development is a style of software development that focuses on the use of existing components to build new applications [17]. These components may be made in house or may be used from an existing library. Component-based development allows for components to be re-used in different views and aspects of an application and as such is a development style under re-use software engineering. React.js, which was used to build the presentation tier of the application, is a component-based library [1]. Hence to create a web app using React.js, developers need to build components or import pre-existing components from other libraries.

In the proposed web app, every web page is a separate component made up of different React components. Some of these components, like the header, the side navigation bar, and the modal, were specially made and developed in the process of making the application. Some of the other components, like the charts, were imported from other JavaScript libraries. The charts were imported from the Chart.js Library. Below is a screenshot of the dashboard with the side navigation component and the header component *(Figure 4.1)* as well as snippets of code used to build the header component

(Figure 4.2).

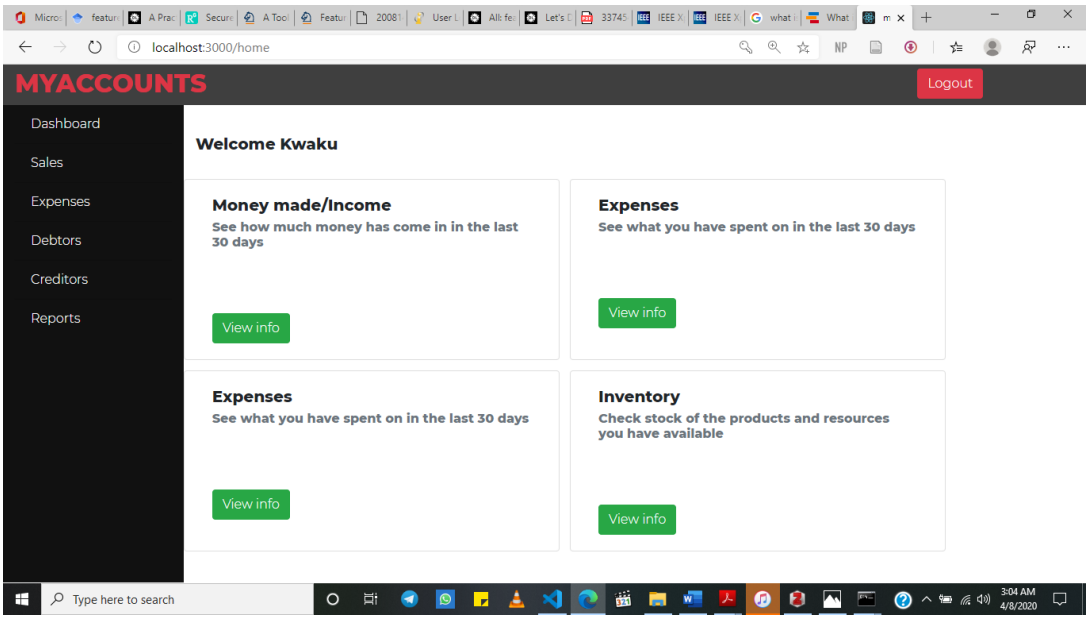

Figure 4.1 The side Navigation component and the header Component on the dashboard

```
export default class Dashboard extends React.Component{
    constructor(props) {
        super(props);
        this.state = { redirect : false, nextpage : ""}
        this.redirectTo = this.redirectTo.bind(this);
    }
   redirectTo = e => {
       this.setState({redirect : true,nextpage: e.target.name,})
   }
    render(){
        return(
            <div>
                <Header startPage="/home" />
                <div className="">
                    <div className="row">
                        <SideNav/>
                            <section className="homepage-main col col-sm-10
container"><br/>
                            <h5>Welcome
{localStorage.getItem('fname')}</h5><br/>
                            <div className="row">
                                <div className="card col-xs-12 col-sm-5
col-lg-5" id="homepage-grid-item">
                                    <div className="card-body">
                                        <h5 className="card-title">Money
```

Figure 4.2 Snippet of the code for the header component

4.3.2 Feature Driven Development

Feature driven development is an agile software development technique that focuses on the development of the different functionalities [28] of a system. The system was developed functionality by functionality. The order in which the system was developed was as follows:

- Develop the landing page
- Develop the sign up and login process
- Develop the dashboard
- Develop the processes and interface needed to manipulate and view expenses
- Develop the processes and interface needed to manipulate and view sales
- Develop the processes and interface needed to manipulate and view debtor and creditor information
- Develop the processes and interface needed to view income statements for different periods of time

4.4 Project File Structure

The project was stored in a folder named “myaccounts’. This folder had two sub-folders and an SQL file titled "myaccounts.sql” which can be imported into an SQL database management software and run to set up the SQL database. The two subfolders in the myaccounts folder were “frontend” and “myaccounts_api”. The frontend folder was created by running the “create-react-app *project_name*” command. This command automatically creates an instance of the React framework on the computer being used for development [1] . The default file structure for a React app created with the create-react-app command is

- `project_name`

- node_modules
- public
- src
- .gitignore
- package-lock.json
- package.json
- README.md

The “node_modules” folder contains all the JavaScript libraries and modules which have been preconfigured to allow the react application to run. It also contains all the additional modules and libraries which were installed to provide additional react components and functionalities, like the chart.js library and the axios.js library.

The “public” folder contains the HTML file through which the react application is rendered. The “src” folder contains all the source code for the React app. Most of the files created in the development process were placed in different folders in the src folder. Below is the final structure for the src folder of this project.

- src
  - assets
  - bootstrap
  - components
  - css
  - images
  - pages
    - creditors
    - debtors
    - expenses
    - general
    - reports
    - sales
  - App.js
  - App.test.js
  - index.css
  - index.js
  - routes.js
  - serviceWorker.js
  - setupTests.js

The pages folder contains each of the different pages. Each subfolder except the “general” folder contains components that would be used to manipulate related data. For example, the “sales” folder contains the different components used to add, edit and view sales

details. The “general” folder contains the landing page interface, the dashboard, and the “login” and “sign up” components. The file structure of the pages folder is given below.

- pages
  - creditors
    - add-creditor.js
    - all-creditor.js
    - creditors-page.js
    - edit-creditor.js
  - debtors
    - add-debtor.js
    - all- debtor.js
    - debtors-page.js
    - edit- debtor.js
  - expenses
    - add-expense.js
    - all- expenses.js
    - expenses-page.js
    - edit- expense.js
  - general
    - dashboard.js
    - landing-page.js
    - login-page.js
    - signup-page.js
  - reports
    - income-statement.js

The “my_accounts_api” folder contains all files which make up the RESTful API used in the application layer of the system. The file structure for this folder is given below:

- my_accounts_api
  - database
    - db.js
  - node_modules
  - routes
    - creditors.js
    - debtors.js
    - expenses.js
    - income.js
    - sales.js
    - users.js
  - index.js
  - package-lock.json

The files in the “routes” folder contain the different endpoints through which the react application can communicate with the API to make changes in the database. The “db.js”

file contains JavaScript code which creates a connection to the SQL database.

4.5 Functionality and Graphical User Interface

On opening the web application, the user is presented the landing page from which he can decide to Login or sign up.

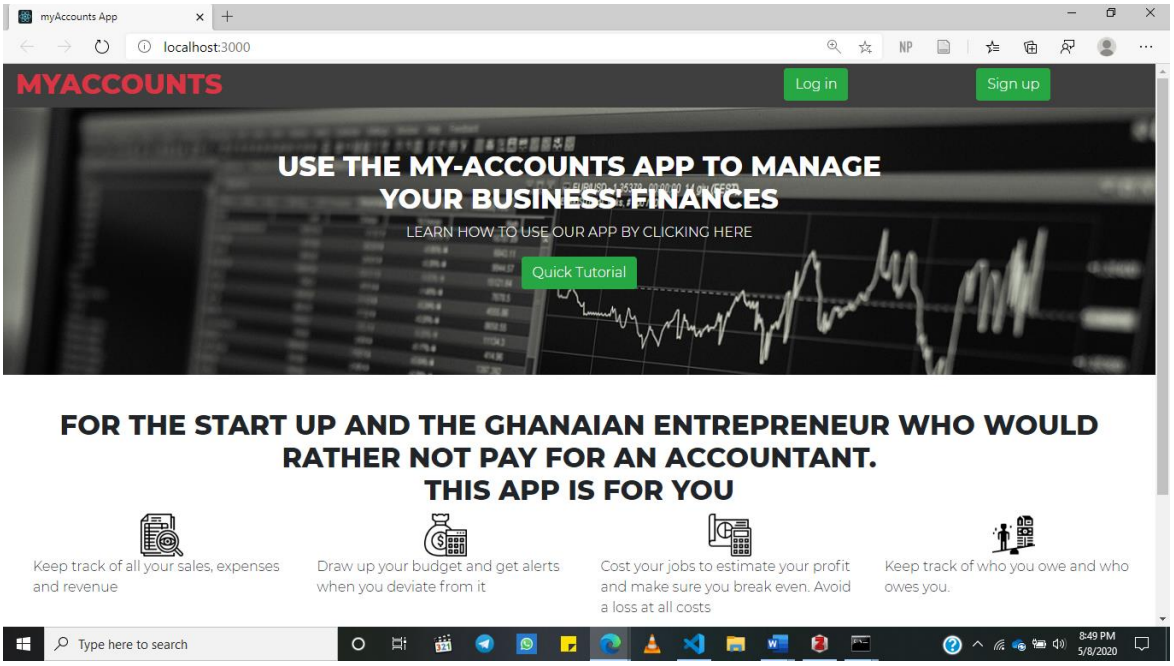

Figure 4.3 Screenshot of the Landing page of the accounting system

If the user has an account, the user proceeds to click the login button and is met with a form which demands his/her email address and password. On submitting the user's login details, the React app makes an API call using the axios library to the database to check if any email address and password of that form exist in the database.

Once it does, the system shows the user their dashboard. Below is a screenshot of the User Interface when the user enters an invalid email or password during the log in process.

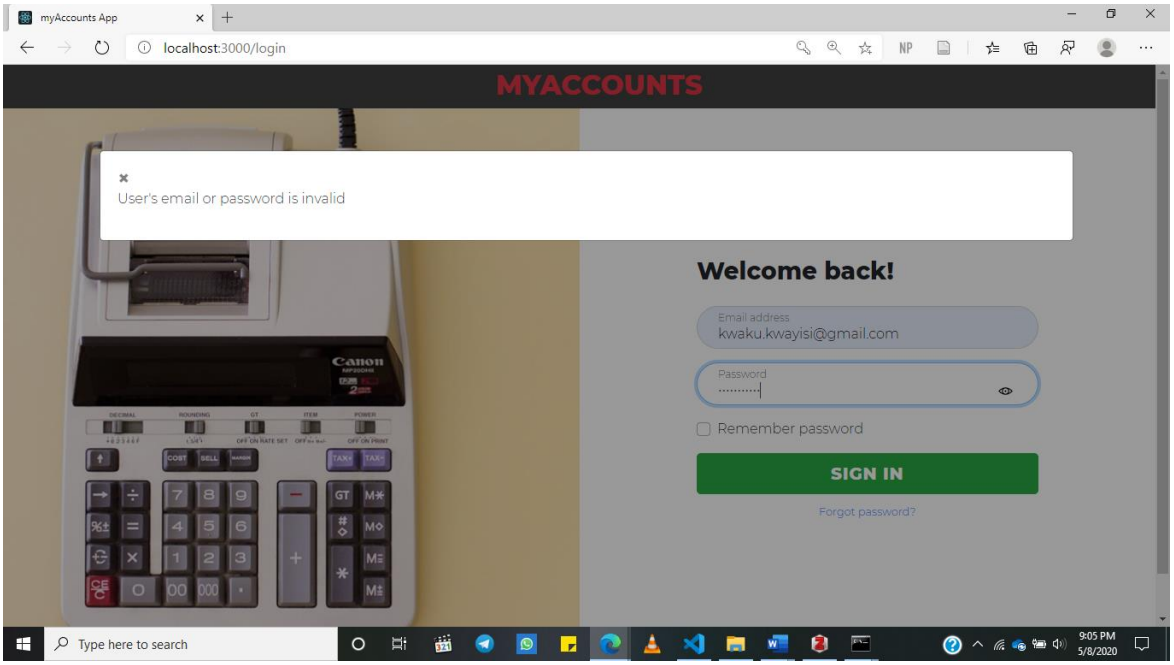

Figure 4.4 Login component when the user enters an invalid email address or password

Once the user gains access to the dashboard, the user can decide to manage his expenses, sales, creditors details, debtors details and view his income statement in reports (can be seen in Figure 4.1) .

The four categories of details “sales”, “expenses” , "debtors" and "creditors" have 3 main functionalities. For these categories, you can

- add a new entry: This means you can enter in details of a new expense, a new sale the user has made, a new debtor, or a new creditor that the user has.
- Edit an existing entry: This means you can edit the details of an existing entry that has previously been added in case some of the details were wrong.
- Display information in the form of graphs: this means that there is a graphical representation of the expense, sale, debtor and creditor details on the page for easy analysis.

On selecting the “expenses” option, the interface displays a bar graph showing the total sum of expenditure for the past 6 months and lists the details of the most recent expenses

(Figure 4.5) . These details and figures used to draw the graph are received from an API call by the axios library using an API endpoint which response with the expense details of a user in the JavaScript Object Notation (JSON) format.

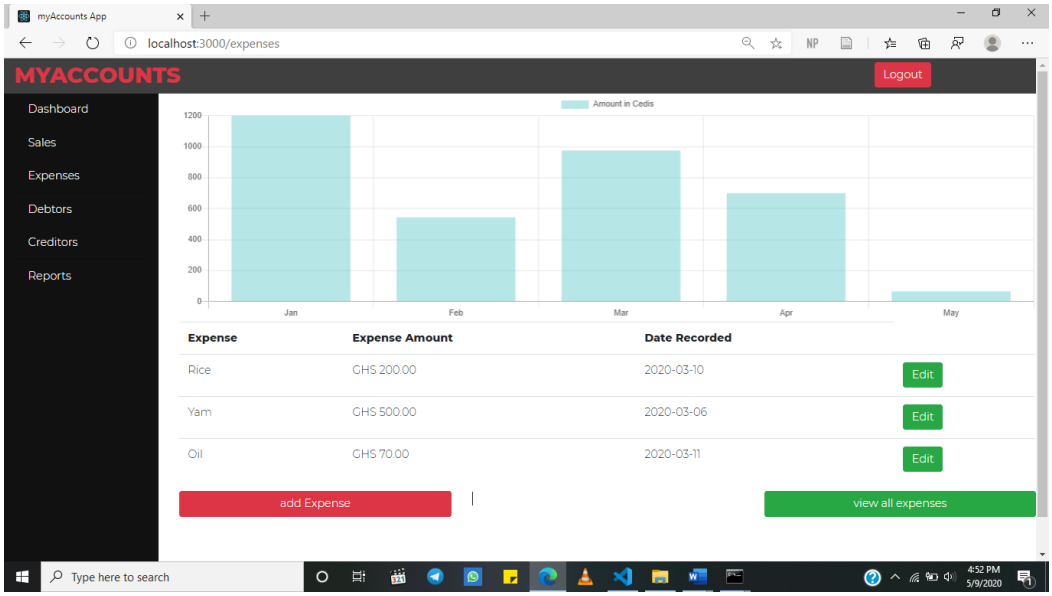

Figure 4.5 Screenshot of the page which gives a summary of a user's expenses

There is an "Edit" button beside each expense listed for the user to be able to edit the details of that expense. On clicking the "Edit" button the user gets redirected to a page which contains a form with all the expense details prefilled. From there the user can edit these details.

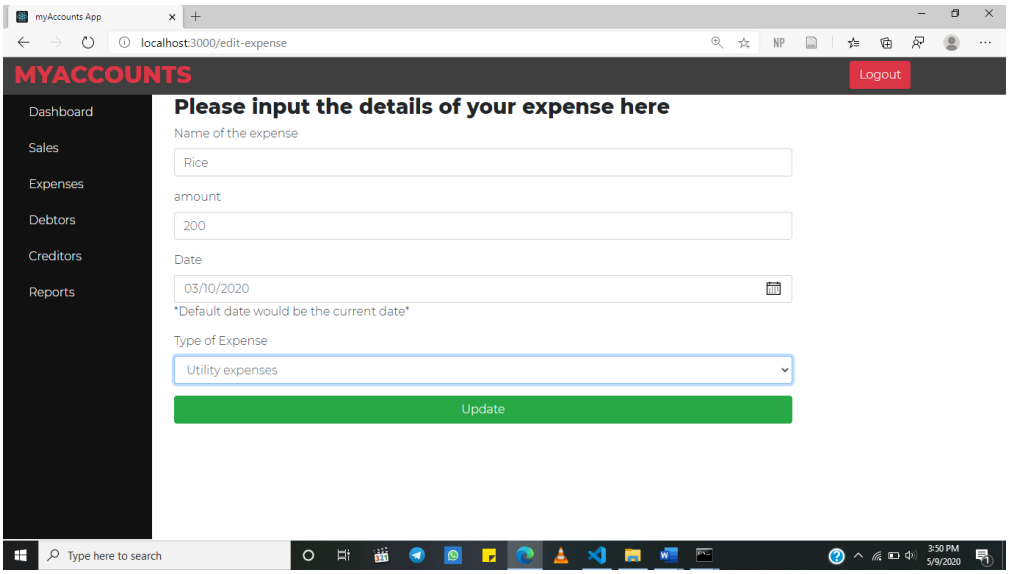

Figure 4.6 Screenshot of the page which allows users to add a new expense

On the expenses page, there is also an “add expense” and “view all expenses” button on the page. The “add expense” button allows the user to add a new expense to the system. On clicking the “add expense” button, the user is redirected to a page with a form similar to that which is above us on the “edit expense” component in figure 4.2 . Upon filling the form and clicking on the proceed button, the user is alerted that the expense has been added and is redirected to the expense page. This procedure is similar for the other “sales”, “creditors” and “debtors” categories.

The “reports” page shows an income statement for the most recent calendar year. At the bottom of the “reports” page, is an HTML “select” element which allows users to select the income statement of a new calendar year. Users can click on it and select from the dropdown menu to select expenses from a different year.

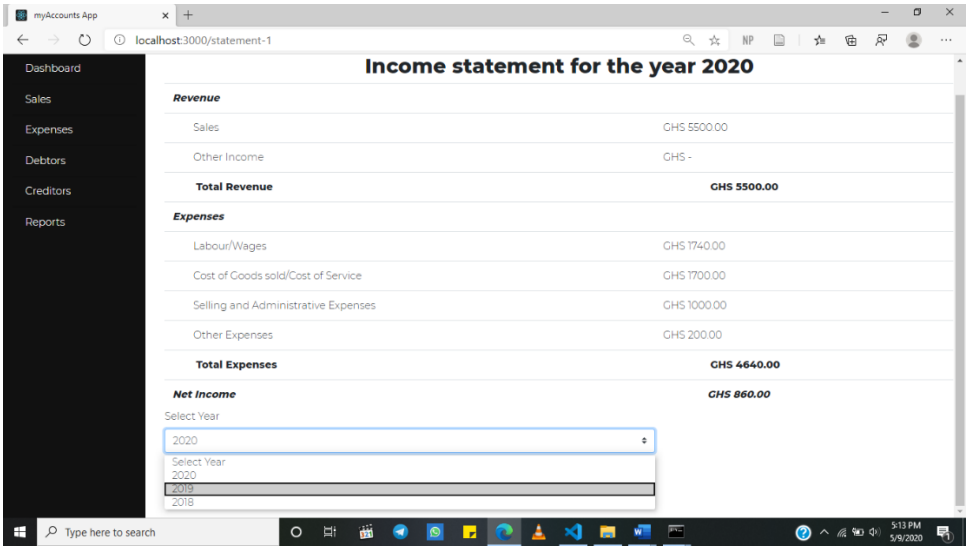

Figure 4.7 Screenshot of the page which shows users their income statement

Chapter 5: Testing and Results

In this chapter, a detailed description of the different testing procedures the system underwent to ensure full functionality is presented. Three different types of testing were performed on the system. These were component testing, system testing and usability testing. In addition to this, an analysis of the test results and a possible explanation for these results will also be presented.

5.1 Component Testing

In component testing, and different integrated program units called components are tested as a whole [30]. To perform tests on the different components, Jest, a JavaScript testing framework was used to do so.

From the presentation layer, the following components were tested:

- the Login and sign up components
- all components which control the manipulation of details for debtors, creditors, sales and expenses,
- and finally, the components which display the income statements.

A sample of the testing code and results with Jest is provided below. This was a test to check the functionality of the login component .

```
test('check if error message on login without filling any field
works', async () => {
  const { findByText, getByText } = render(
    <BrowserRouter>
      <LoginPage/>
  </BrowserRouter>);

   fireEvent.click(getByText('Sign in'))

   const modal = await getByText("Please type your email in the
required field")
   expect(modal).toBeInTheDocument();

  //Checks if user receives an error message on failed Login
});

test('check if error message without password works', async () =>
{
  const { findByText, getByText,getByLabelText } = render(
    <BrowserRouter>
'kwaku.boohene1' } })
```

Figure 5.1 Code snippet showing a JavaScript Jest test for the Login component

The results of the test are shown in this screenshot:

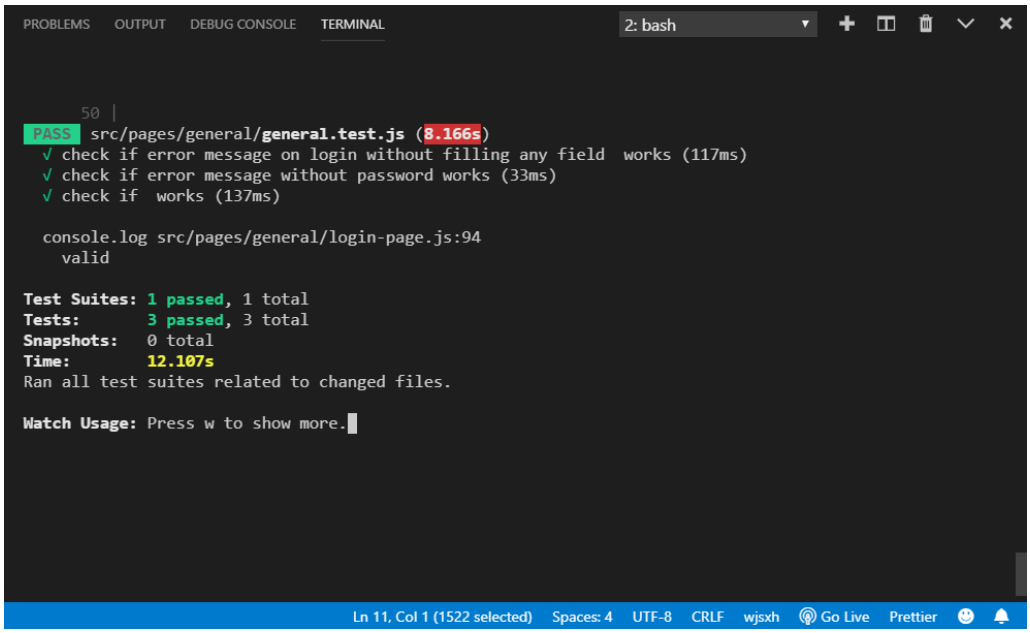

Figure 5.2 Screenshot of the results from the Jest test for the Login component

Postman was used to test the different endpoints of the API. Postman allows developers to perform tests to ensure that the API provides accurate and expected responses in different scenarios [31]. Below are some screenshots of tests taken using Postman

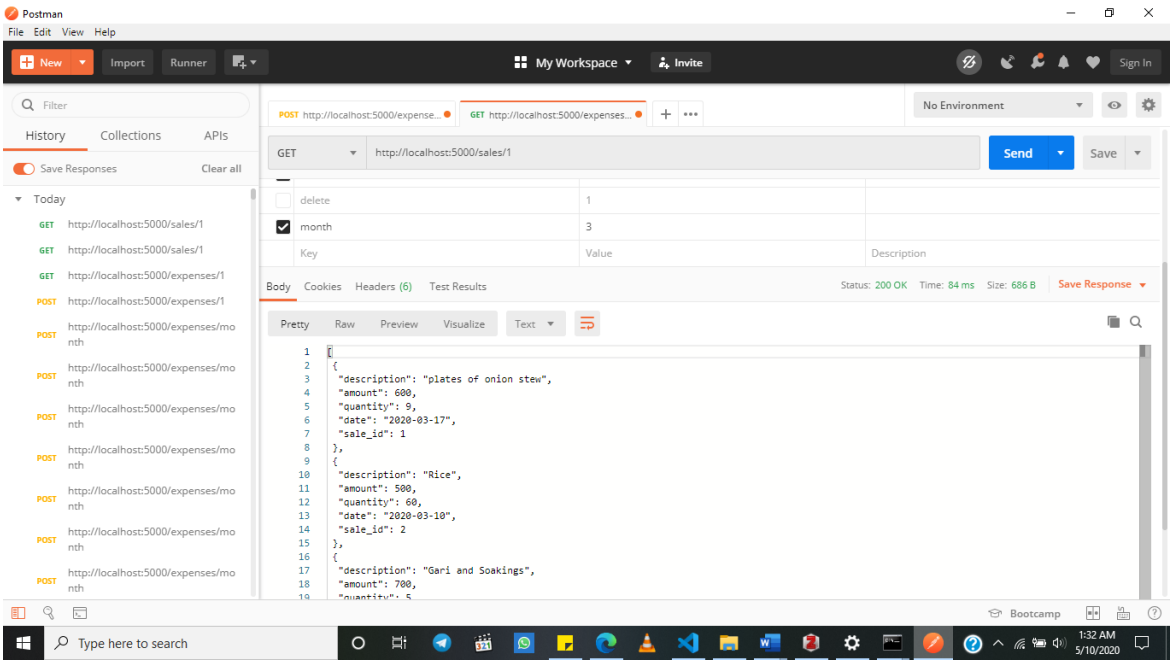

Figure 5.3 Screenshot of a Postman test of the system's API

5.2 System Testing

In system testing, some or all the components which make up the entire system are tested together. The purpose of system testing is to focus on the interactions between components and observe their results [30]. To test the accounting system, an end-to-end test was performed on the accounting system. In this test, the system was taken through sequential tests to ensure that it was fully functional. These were the following tests, results, and comments from the End-to-End testing process:

Table 5.1: A Table showing the results of the End-to-end Testing

Test	Expected Result	Actual Result	Comments
User creates account	User's details can be successfully seen in the database and the app redirects to the login page	User's details could be seen in the database, but the app redirected to another user's dashboard.	The route which was used for redirecting a user had a logical error.
User logs in	User receives welcome message in modal and is redirected to the dashboard	User received a welcome message and was redirected to the dashboard.	
User adds expense	User's new expense and its details are successfully added to the database	User's new expense and its details were successfully added to the database	
User edits an existing expense	The user is redirected to his expenditure page after being alerted the expense has been successfully updated.	The user was redirected to his expenditure page after being alerted that the expense had been successfully updated.	
User adds new sale entry	User's new sale entry and its details are successfully added to the database	User's new sale entry and its details were successfully added to the database	
User edits an existing sale entry	The user is redirected to his sales page after being alerted the expense has been successfully updated.	The user was redirected to his sales page after being alerted that the expense had been successfully updated.	
User adds details of a new debtor	The entered details are successfully added to the database	The entered details were successfully added to the database	
User edits existing details of a debtor	The user is redirected to the page which lists all debtors after the details have been successfully updated	The user was redirected to the page which lists all debtors after the details were successfully updated	
User changes income statement period	The details on the income statement page reflect the details of expenditure, revenue and income of the selected year	The details on the income statement page did not change	This was as a result of a syntax error.

All errors that were encountered during the end-to-end testing were resolved after.

5.3 Usability Testing

The application was tested by 3 business owners who fit the description of the user given in Chapter 2. They tested the application through “Team Viewer”. Team Viewer is an application that allows users to remotely control the computer of another [11]. Team Viewer was used to gain access to the computer used to develop the system by the 3 users who tested the system. They were instructed to open the application on the Google Chrome Browser on the computer used to develop the system. In the browser, they were instructed to go to “Developer mode”. “Developer mode” can be used to test a web application on different screen sizes [24] . They described the user interface as simple and easy to use and relevant as they liked the applications' functionality. They were able to manage their expenses, sales, creditor details, debtor details and monitor their business' income statement. They, however, complained about the following issues:

- The user interface was not mobile-friendly: They were instructed to resize the web browser to the size of their mobile phone device screen through the developer options available on the browser. Using the application with that interface, they concluded that using the application on a smartphone through that interface would be quite cumbersome.
- Lack of tutorials: Even though they were able to successfully use the application, they hinted that it would have been easier to use if there were more instructions to guide its usage.

Chapter 6: Conclusion and Future Works

6.1 Summary

This report contains a detailed description of the design, building and testing process of an accounting app for sole proprietors of medium and small-scale businesses in Ghana. The application allows users to keep track of the accounting information of their business without the help of a professional accountant. This project was embarked on as an effort to improve the poor accounting practices of Ghanaian businesses, as described in Chapter one. The project, however, does need improvements in terms of user interface design and security measures, should it be made available to users on the market.

6.2 Limitations

The project meets the stated functional requirements in chapter two. However, there are some limitations to this project.

- Normalisation: The database, even though functional, has not been normalised to the third normal form. In the event where the number of users increases or extra functionalities are added to the software, the system may become slower or experience failure.
- Security issues: Data entries to the database are not encrypted. The web application is also not hosted on a secure server. This opens the users of the application up to different forms of malicious attacks.
- User Experience /User Interface (UI/UX) Issues: Users complained about the user interface when accessing the application from devices of different sizes. This may make it difficult for users to adapt to the application. There is also a lack of tutorials or training to educate users and train them on the different features of the application

and how to use those features efficiently. These may contribute to user experience issues.

6.3 Future Work

Although the functional requirements for the application have been implemented, it would be advisable to improve the application before making it available to the general market. Ways in which this could be done would be to:

- Develop a native mobile application to support the current presentation layer: A native mobile interface which utilises the already built business and data layer of the application would help improve user experience and the general user interface.
- Upon further research, increase the project's scope: Additional functionalities may be added to the system to satisfy the users. The application can be improved to present other forms of financial statements aside, just income statements.
- General improvements in System security and efficiency: Non-Functional requirements like system security and efficiency may be worked on in order to generally improve the application's overall performance. For example, an encryption algorithm can be implemented to encrypt and decrypt data in the database. The database can also be normalised to third normal form to increase its efficiency.

6.4 Conclusion

Accounting is important to the day to day running of a business. As said in chapter 1, better accounting practices can essentially improve a business' output and performance. The use of technology and software development to create tools which can solve problems in society is a benefit that the world currently enjoys. As the problem of bad accounting practices is one which small businesses in Ghana face, this project is an essential step in

curbing and tackling it.

References

[1] Sanchit Aggarwal. 2018. Modern Web-Development using ReactJS. *Int. J. Recent Res. Asp.* 5, (2018), 133–137.

[2] Mohammed Amidu, John Effah, and Joshua Abor. 2011. E-Accounting Practices among Small and Medium Enterprises in Ghana. *J. Manag. Policy Pract.* 12, 4 (August 2011), 146–155.

[3] Gilbert Kwabena Amoako. 2013. Accounting practices of SMEs: A case study of Kumasi Metropolis in Ghana. *Int. J. Bus. Manag.* 8, 24 (2013), 73.

[4] Viknes Balasubramanee, Chathuri Wimalasena, Raminder Singh, and Marlon Pierce. 2013. Twitter bootstrap and AngularJS: Frontend frameworks to expedite science gateway development. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, 1–1. DOI:https://doi.org/10.1109/CLUSTER.2013.6702640

[5] R. Bourdon. 2014. WampServer, the web development platform on Windows–Apache, MySQL, PHP. *Viitattu* 25, (2014), 2014.

[6] CACM Staff. 2016. React: Facebook's functional turn on writing Javascript. *Commun. ACM* 59, 12 (2016), 56–62.

[7] Satyadhyan Chickerur, Anoop Goudar, and Ankita Kinnerkar. 2015. Comparison of Relational Database with Document-Oriented Database (MongoDB) for Big Data Applications. In *2015 8th International Conference on Advanced Software Engineering Its Applications (ASEA)*, 41–47. DOI:https://doi.org/10.1109/ASEA.2015.19

[8] Helder Da Rocha. 2019. *Learn Chart. js: Create interactive visualizations for the Web with Chart. js 2*. Packt Publishing Ltd.

[9] M. David. 2017. *Statistics for Managers, Using Microsoft Excel*. Pearson Education India.

[10] Sanja Delcev and Drazen Draskovic. 2018. Modern JavaScript frameworks: A Survey Study. In *2018 Zooming Innovation in Consumer Technologies Conference (ZINC)*, 106–109. DOI:https://doi.org/10.1109/ZINC.2018.8448444

[11] Mala Dutta, Kamal K. Sethi, and Ajay Khatri. 2014. Web based integrated development environment. *Int. J. Innov. Technol. Explor. Eng.* 3, 10 (2014), 56–60.

[12] John R. Dyson. 2007. *Accounting for non-accounting students*. Pearson Education.

[13] Eduardo B. Fernandez, Mihai Fonoage, Michael VanHilst, and Mirela Marta. 2008. The Secure Three-Tier Architecture Pattern. In *2008 International Conference on Complex, Intelligent and Software Intensive Systems*, 555–560. DOI:https://doi.org/10.1109/CISIS.2008.51

[14] Jay Greenspan and Brad Bulger. 2001. *MySQL/PHP database applications*. John Wiley & Sons, Inc.

[15] Geoff Grindrod, Oto Slavos, Saigiridhar Kodali, and Clinton Hallman. 2005. *System and method for customizing and processing business logic rules in a business process system*. Google Patents.

[16] Zheng Hao, Zhu Limiao, and Huang Hua. 2012. A Web Design Mode for Browsers to CSS Compatibility Issues. In *2012 Fourth International Conference on Multimedia Information Networking and Security*, 160–163. DOI:https://doi.org/10.1109/MINES.2012.53

[17] Florian Haupt, Frank Leymann, Anton Scherer, and Karolina Vukojevic-Haupt. 2017. A Framework for the Structural Analysis of REST APIs. In *2017 IEEE International*

Conference on Software Architecture (ICSA), 55–58. DOI:https://doi.org/10.1109/ICSA.2017.40

[18] Srikanth Jonnada and Jothis K Joy. 2019. Measure your API Complexity and Reliability. In *2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA)*, 104–109. DOI:https://doi.org/10.1109/SERA.2019.8886790

[19] Collins Owusu Kwaning, Kofi Nyantakyi, and Bright Kyereh. 2015. The challenges behind smes' access to debts financing in the Ghanaian financial market. *Int. J. Small Bus. Entrep. Res.* 3, 2 (2015), 16–30.

[20] Kai Lei, Yining Ma, and Zhi Tan. 2014. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. In *2014 IEEE 17th International Conference on Computational Science and Engineering*, 661–668. DOI:https://doi.org/10.1109/CSE.2014.142

[21] Jon Loeliger and Matthew McCullough. 2012. *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Inc.

[22] Kan Lu, YingLi Fu, CaiDong Gu, and Liang Zhang. 2012. Problems and Solutions of Popularization of Accounting Computerization. *Phys. Procedia* 33, (January 2012), 1155–1159. DOI:https://doi.org/10.1016/j.phpro.2012.05.190

[23] Michael McLaughlin. 2013. *MySQL Workbench: Data Modeling & Development*. McGraw Hill Professional.

[24] Prateek Mehta. 2016. Introduction to Google Chrome Extensions. In *Creating Google Chrome Extensions*. Springer, 1–33.

[25] John Paul Mueller. 2006. *Mastering Web Development with Microsoft Visual Studio 2005*. John Wiley & Sons.

[26] Glenn Owen. 2007. *Using Quickbooks Pro 2004 for Accounting*. Thomson/South-Western.

[27] Ozé Inc. 2018. OZÉ | Mobile App & Software to manage and grow your business. *OZÉ | Mobile App & Software to manage and grow your business*. Retrieved September 16, 2019 from https://www.oze.guru

[28] Steve R. Palmer and Mac Felsing. 2001. *A practical guide to feature-driven development*. Pearson Education.

[29] Andrew John Poulter, Steven J. Johnston, and Simon J. Cox. 2015. Using the MEAN stack to implement a RESTful service for an Internet of Things application. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 280–285. DOI:https://doi.org/10.1109/WF-IoT.2015.7389066

[30] Ian Sommerville. 2011. Software engineering 9th Edition. *ISBN-10* 137035152, (2011).

[31] Adrian SROKA. 2016. POSTMAN– Powerful API testing tool. *Diwebsity[online]* 21, (2016).

[32] Francesco Strazzullo. 2019. HTTP Requests. In *Frameworkless Front-End Development: Do You Control Your Dependencies Or Are They Controlling You?*, Francesco Strazzullo (ed.). Apress, Berkeley, CA, 113–138. DOI:https://doi.org/10.1007/978-1-4842-4967-3_5

[33] Ling-ling Wei and Wei Zhang. 2009. A Method for Rough Relational Database Transformed into Relational Database. In *2009 IITA International Conference on Services Science, Management and Engineering*, 50–52. DOI:https://doi.org/10.1109/SSME.2009.79

Appendices

Appendix A – Requirements Gathering

1. Name of interviewer: .....
2. Place of interview: .....
3. Date of interview: .....
4. Duration Interview: .....

Questions:

1. What type of business do you own and what does your business do?
2. How big is your business?
3. What challenges or problems is your business facing?
4. Do you keep any accounting records?
5. If yes, how do you keep and update these records?
6. If no, why do you not keep accounting records?
7. Do you find accounting difficult? If yes, why?
8. Which financial details of your business would you like to keep track of?
9. Have you ever gone for a loan for your business? What was the experience and what information were you required to show?
10. Do you use a smartphone, a computer, or any device to help you run your business? How do you use these device(s) in the running of your business?

Appendix B – Component Testing

A. Automated Tests for Login Component

```
import React from 'react';
import { render,fireEvent } from '@testing-library/react';
import LoginPage from './login-page';
import Dashboard from './dashboard';
import { BrowserRouter } from 'react-router-dom';
import 'jest-localstorage-mock';




test('check if error message on login without filling any field
works', async () => {
  const { findByText, getByText } = render(
    <BrowserRouter>
      <LoginPage/>
  </BrowserRouter>);

   fireEvent.click(getByText('Sign in'))

   const modal = await getByText("Please type your email in the
required field")
   expect(modal).toBeInTheDocument();

  //Checks if user receives an error message on failed Login
});

test('check if error message without password works', async () => {
  const { findByText, getByText,getByLabelText } = render(
    <BrowserRouter>
      <LoginPage/>
  </BrowserRouter>);
  fireEvent.change(getByLabelText(/email/i), { target: { value:
'kwaku.kwayisi@gmail.com' } })
//   fireEvent.change(getByLabelText(/password/i), { target: { value:
'kwaku.boohene1' } })

   fireEvent.click(getByText('Sign in'))

   const modal = await getByText("Please type your password in the
required field")
   expect(modal).toBeInTheDocument();
```

```
  //Checks if user receives an error message on failed Login
});

test('check if  works', async () => {
    global.window = { location: { pathname: null } };
  var { findByText, getByText,getByLabelText } = render(
    <BrowserRouter>
      <LoginPage/>

  </BrowserRouter>);
  fireEvent.change(getByLabelText(/email/i), { target: { value:
'kwaku.kwayisi@gmail.com' } })
  fireEvent.change(getByLabelText(/password/i), { target: { value:
'kwaku.boohene1' } })

   fireEvent.click(getByText('Sign in'));


});
```

Appendix C – Code Snippet for Add Creditor Component

```
import React from "react";
import {Redirect} from "react-router-dom";
import Header from "../../components/header";
import SideNav from "../../components/sidenav";
import axios from 'axios';

export default class AddCreditor extends React.Component{
    constructor(props){
        super(props);
        this.state = {
            cname : "",
            amount : "",
            dBorrow : "",
            dPay:"",
            vPay:"",
            redirect: false,
            userid: Number(localStorage.getItem('userid')),

            quantity:"",
        };
        this.Change = this.Change.bind(this);
        this.addFormData = this.addFormData.bind(this);
        this.onProceed =  this.onProceed.bind(this);
        this.validate = this.validate.bind(this);
        this.getDate = this.getDate.bind(this);
    }


    Change = e => {
        this.setState({
        [e.target.name]: e.target.value
    });

    };

    getDate(){
        var today = new Date().toISOString().split('T')[0];
        var date =  String(today);
        return date;
    }

        addFormData(){
```

```
        axios

            .post('http://localhost:5000/creditors/add',
            {
                'name':this.state.cname,
                'amount':this.state.amount,
                'dBorrow':this.state.dBorrow,
                'dPay': this.state.dPay,
                'vPay': parseInt(this.state.vPay),
                'userid':this.state.userid,

            })
            .then(response =>{
                console.log(response);
                alert('sale added');
                this.setState({redirect: true})

            })
            .catch(error =>{
                console.log(error);
                alert(error);
            })


    }

    Change = e => {
        this.setState({
        [e.target.name]: e.target.value
        });
        var item = e.target.name;
        console.log(this.state);
    };

    validate = e => {
        if(this.state.name===""||this.state.amount===""){
            alert("Please fill out the name, amount and date of
Borrowing on the Form");
            return false;
        }
        if(this.state.date===""){
            this.setState(
                {date: this.getDate(),}
            )
            return true;
```

```
        }else{
            return true;
        }
    };

    onProceed = e => {
        var validate = this.validate();
        if(validate===true){
            this.addFormData();
        }
    };

    render(){
        return(
            <div className="">
                <Header/>
                <div className="">
                    <div className="row">
                        <SideNav/>

                        <div className="col col-sm-10 container">
                            <div className="row">
                                <div className="col col-sm-9">
                                    <h3>Please input the details of
your Creditor here</h3>
                                    <form>
                                        <div className="form-group">
                                            <label>Name</label>
                                            <input type="text"
                                            name = "cname" onChange =
{e => this.Change(e)}
                                            className="form-control"
placeholder="Name of the person or the Business"
                                            value={this.state.cname}
/>
                                        </div>

                                        <div className="form-group">
                                            <label>Amount /
Value</label>
                                            <input type="number" name=
"amount" onChange = {e => this.Change(e)}
                                             className="form-control"
placeholder="Value of the work done or item(s) bought on credit"
value={this.state.amount}/>
```

```
                                        </div>


                                        <div className="form-group">
                                            <label>Date
Recorded</label>
                                            <input type="date"
name="dBorrow" placeholder="YYYY-MM-DD" required

                                            className="form-control"
onChange = {e => this.Change(e)}
                                            title="Enter a date in
this format YYYY-MM-DD" value={this.state.dBorrow}/>
                                            *Default date would be the
current date*
                                        </div>

                                        <div className="form-group">
                                            <label>Deadline for
Payment</label>
                                            <input type="date"
name="dPay" placeholder="YYYY-MM-DD" required

                                            className="form-control"
onChange = {e => this.Change(e)}
                                            title="Enter a date in
this format YYYY-MM-DD" value={this.state.dPay}/>
                                            *Default date would be the
current date*
                                        </div>

                                        <div className="form-group">
                                            <label>Payment
Made</label>
                                                <div class="input-
group mb-3">
                                                <div class="input-
group-prepend">
                                                    <label
class="input-group-text" for="inputGroupSelect01">Options</label>
                                                </div>
                                                <select class="custom-
select" id="inputGroupSelect01" value={this.state.vPay}>
                                                    <option value='1'
>Yes</option>
```

```
                                                    <option selected
value="0" >No</option>

                                                </select>
                                                </div>
                                            </div>




                                            {/* <input type="submit"
value="Proceed" className = "btn btn-danger btn-block"/> */}
                                            <button type="button"
className="btn btn-success btn-block"
                                             onClick = {this.onProceed}>
                                                Proceed
                                            </button>


                                        </form>

                                    </div>
                                </div>



                            </div>
                        </div>

                    </div>
                    {this.state.redirect?<Redirect to="./all-
creditors"/>:null}

                </div>
            )
        }

}
```