Assignment Cover Sheet

Programme : Bachelor in Business Analytics

Subject Code and Title : WEB DESIGN AND DEVELOPMENT (BAA3114)

Project Title : **Group Assignment**

Lecturer : Dr. Ibrahim T. Nather Khasro

Assignment Due Date : 5th January, 2024

Student's Declaration

1. I hereby declare that this assignment is based on my own work except where acknowledgment of sources is made.
2. I also declare that this work has not been previously submitted or concurrently submitted for any other courses in Sunway University of College or other institutions.

No	Name	Student ID	Student Imail
1	Aarogya Banepali	21014527	21014527@imail.sunway.edu.my
2	Tai Yong Xin	19098201	19098201@imail.sunway.edu.my
3	Yoon Yen Wei	19118074	19118074@imail.sunway.edu.my
4	Low Kah Keng	19049170	19049170@imail.sunway.edu.my
5	Siow Zi Ting	20096731	20096731@imail.sunway.edu.my

APPROVAL FOR LATE SUBMISSION OF ASSIGNMENT (If Applicable)

If an extension is granted, what is the revised due date? :

Signature of Lecturer: Date:

Marker 's Comments:

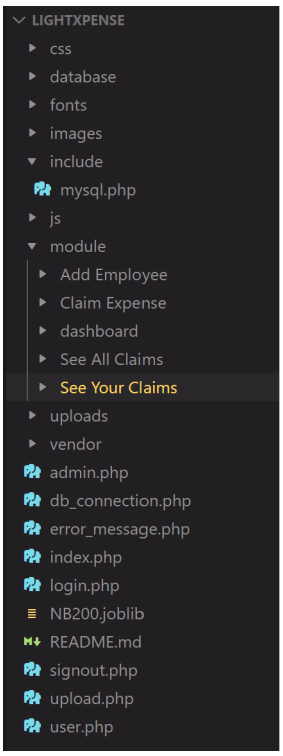
Marks and/or Grade Awarded:

Table of contents

Project directory management and file paths....2
Directories....2
Files....3
Front-end codes....4
1. Landing Page and Login Screen....4
a. Navigation bar....4
b. Home content....6
Landing Section....6
About Section....9
How It Works Section....11
Our Team Section....13
c. Footer....14
d. Login screen....16
2. Admin Account....18
a. Admin Dashboard page....26
b. Add Employee page....32
c. See all employee claims & edit page....37
3. Employee Account....39
a. Employee Dashboard page....45
b. Claim Expense page....51
c. See all personal claims....59
Back-end codes....60
1. PHP with SQL Integration:....60
Login Page....61
i. Admin Dashboard page....62
ii. Add Employee page: Register Form....67
iii. See all employee claims & edit page....69
Employee account:....71
i. Employee Dashboard page....71
ii. Claim Expense....72
2. PHP with Python Integration through shell_exec():....75
Responsiveness....77

Project directory management and file paths

The development of the LightXpense system has been completed through the files in the following format. A brief description of these files and the directories, how they relate with each other and how it comes together to be a complete solution is as follows:



Directories

1. CSS

- Contains CSS files for web page design.

- Standardized design using main.css for the landing page and dashboard.css for dashboards.

- Includes supporting CSS files for icons, fonts, and base HTML codes.

2. Database:

- Houses the SQL export file for easy import into a SQL-based database.

3. Fonts:

- Holds files for the fonts used in the design.

4. Images:

- Stores static images utilized across the website.

5. Include:

- Contains PHP files establishing database connections (currently set for localhost MySQL).

6. JS:

- Houses all JavaScript files responsible for adding responsiveness to the website.

7. Module:

- Specialized modules adding functionality to the dashboard:

    a) Add Employee: Adds the "Add Employee" section in the admin dashboard.

    b) Claim Expense: Codes enabling employees to claim expenses through a single receipt.

    c) Dashboard: The primary dashboard page for both employees and admins.

    d) See All Claims: Adds the "See All Claims" feature for admins.

    e) See Your Claims: Adds the "See Your Claims" feature for employees to view past claims.

8. Uploads:

- Enables users to upload receipt images, temporarily storing them here.

9. Vendor:

- Facilitates external links to integrated solutions such as Bootstrap and jQuery.

Files

1. admin.php:

- The base file enabling the admin dashboard page after login.

2. db_connection.php:

- Establishes a connection with the database before the dashboard/logged-in state.

3. error_message.php:

- Contains codes for error handling.

4. index.php:

- The base file for the landing page.

5. login.php:

- The base file for the login page, handling the login system.

6. signout.php:

- The file manages sign-out handling after a logged-in state.

7. upload.php:

- The file handling user receipt image uploads.

8. user.php:

- The base file for the user dashboard after employees login.

9. NB200.joblib:

- The machine learning model capable of taking text and outputting an accounting category.

Front-end codes

The front-end development of our academic project was a crucial aspect in ensuring a user-friendly and efficient interface for both employees and administrators. We utilized a combination of HTML, CSS, JavaScript, and the Bootstrap framework to create a responsive and visually appealing user experience. The front-end development for each section of the website and how it flows is described below:

1. Landing Page and Login Screen

The user journey begins with a well-designed landing page that succinctly describes the functionalities of our solution. It has been divided into several sections that make the landing page complete and are easily readable as each starts with a comment and uses section tags such as <header> <section> and <footer> to help keep track.

a. Navigation bar

```
<!-- header
================================================== -->
<header id="header">
  <div class="header-logo">
    <a href="index.html">LightXpense</a>
  </div>

  <nav id="header-nav-wrap">
    <ul class="header-main-nav">
      <li class="current">
        <a class="smoothscroll" href="#home" title="home">Home</a>
      </li>
      <li><a class="smoothscroll" href="#about" title="about">About</a></li>
      <li>
        <a class="smoothscroll" href="#howitworks" title="howitworks"
          >How it works</a
        >
      </li>
      <li>
        <a class="smoothscroll" href="#ourteam" title="ourteam"
          >Our Team</a
        >
      </li>
    </ul>

    <a href="login.php" title="sign-up" class="button button-primary cta"
      >Login</a
    >
  </nav>

  <a class="header-menu-toggle" href="#"><span>Menu</span></a>
</header>
<!-- /header -->
```

The navigation bar and header section contain quick access links to the same page, as well as the login screen for users and admin to access the main content. It has been created using an unordered list with anchor tags for the links. The header, identified by the <header> element, serves as a cohesive container for the various components contributing to user navigation.

The encompassing <header> element is assigned the id header to facilitate potential styling and scripting adjustments. It establishes the framework for the logo, navigation menu, responsive menu toggle, and login button. Within the header, a distinct with the class header-logo encapsulates the application logo. This logo, represented by an anchor <a> element, acts as a clickable entity redirecting users to the home page, fostering seamless navigation. The navigation menu resides within a <nav> element, marked by the id header-nav-wrap. It comprises an unordered list (<ul>) with the class header-main-nav. Each list item (<li>) signifies a specific navigational link such as "Home," "About," "How it works," and "Our Team." The application of the smoothscroll class enables smooth transitions when users interact with these links.

To cater to varying screen sizes, an anchor <a> element with the class header-menu-toggle facilitates a responsive menu toggle. Upon activation, this toggle displays a menu icon labeled "Menu," enhancing user experience, particularly on smaller screens. This is handled through @media command on the CSS file to show when the screen-size is lower than a predefined max screen size.

The login button, featured as an anchor <a> element with the class button button-primary cta, directs users to the "login.php" page. This button is strategically styled to attract attention, serving as a prominent call-to-action. Emphasis on responsive design is evident in the inclusion of the responsive menu toggle (header-menu-toggle). This feature ensures optimal user interaction and navigation on devices with limited screen real estate.

Through the implementation of the code defined above, it looks like this on the front-end.

b. Home content

Landing Section

```
<section
  id="home"
  data-parallax="scroll"
  data-image-src="images/hero-bg.png"
  data-natural-width="3000"
  data-natural-height="2000"
>
  <div class="overlay"></div>
  <div class="home-content">
    <div class="row contents">
      <div class="home-content-left">
        <h3 data-aos="fade-up">Welcome to LightXpense</h3>

        <h1 data-aos="fade-up">
          One click solution <br />
          to reimbursement <br />
          processing
        </h1>

        <div class="buttons" data-aos="fade-up">
          <a href="#ourteam" class="smoothscroll button stroke">
            <span class="icon-circle-down" aria-hidden="true"></span>
            Preview
          </a>
          <a
            href="https://youtu.be/O7Xq6C-AVPQ"
            data-lity
            class="button stroke"
          >
            <span class="icon-play" aria-hidden="true"></span>
            Watch Video
          </a>
        </div>
      </div>

      <div class="home-image-right">
        <img
          src="images/iphone-app-470.png"
          srcset="
            images/iphone-app-470.png 1x,
            images/iphone-app-940.png 2x
```

```
            images/iphone-app-470.png 1x,
            images/iphone-app-940.png 2x
          "
          data-aos="fade-up"
        />
      </div>
    </div>
  </div>
  <!-- end home-content -->

  <div class="home-scrolldown">
    <a href="#about" class="scroll-icon smoothscroll">
      <span>Scroll Down</span>
      <i class="icon-arrow-right" aria-hidden="true"></i>
    </a>
  </div>
</section>
<!-- end home -->
```

This HTML and CSS code snippet defines the structure of the landing section within the LightXpense web application. The landing section, designated by the <section> element, incorporates visually engaging content, including a background image, welcoming text, and interactive buttons. This academic analysis dissects each HTML element and class, elucidating their functionalities and contributions to the overall design and user experience.

1. Landing Section Structure:
The <section> element, marked by the id home, constitutes the landing section. Employing the data-parallax attribute, it incorporates a parallax scrolling effect for a visually dynamic experience. A popular technique on websites, the parallax scrolling effect, gives the impression of depth and three dimensions by having the background content scroll at a different rate than the foreground text. This is further supported through using data-image-src attribute, which specifies the background image ("images/hero-bg.png"), the hero-bg.png file inside the images folder, contributing to the aesthetic appeal.

Within the landing section, a element with the class overlay serves as an overlay. This overlay enhances text legibility against the background image, ensuring optimal readability. Essentially, it helps seamlessly overlay text and other content on top of the background image.

2. Content:
The main content of the landing section resides within a element with the class home-content. It encompasses welcoming text, buttons, and an illustrative image for a comprehensive visual experience.

The left section of the home content features a with the class home-content-left. It includes a dynamically presented welcome message (<h3 data-aos="fade-up">Welcome to LightXpense</h3>) and a prominent heading (<h1 data-aos="fade-up">One click solution to reimbursement processing</h1>). The data-aos attribute applies a fade-up animation for a smooth and engaging entrance effect.

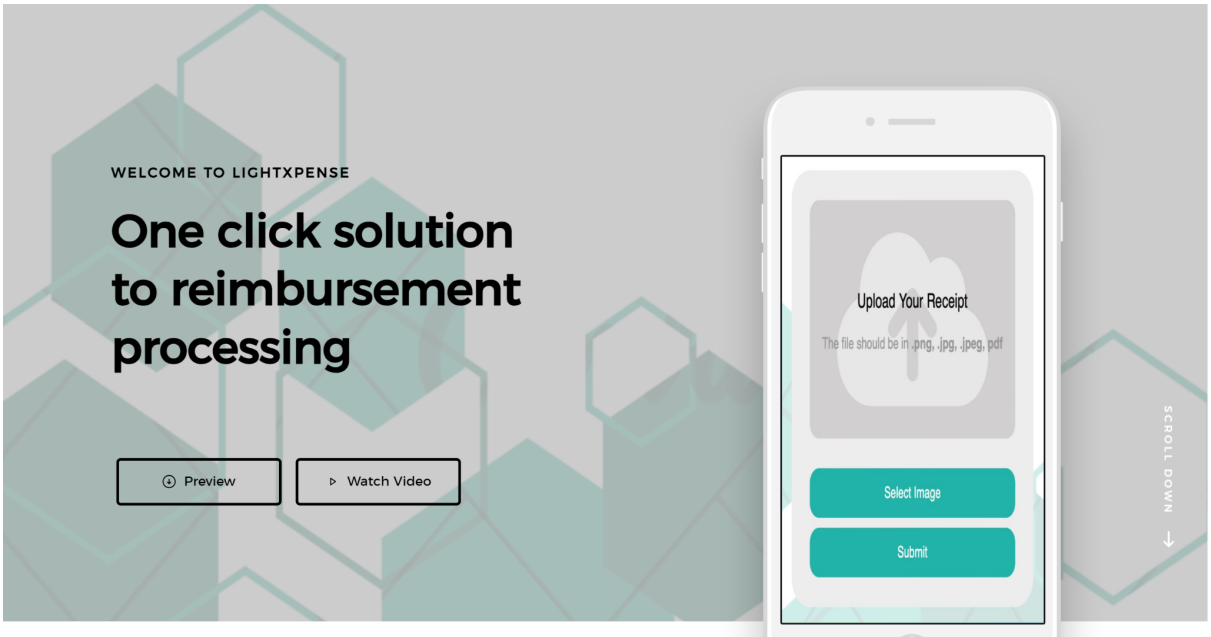
Interactive buttons within the landing section are encapsulated in a with the class buttons. Two buttons are present, each facilitating a different interaction. The first button, a preview link (<a href="#ourteam" class="smoothscroll button stroke">), encourages users to

explore further sections. The second button, triggering a video popup link (<a href="https://youtu.be/O7Xq6C-AVPQ" data-lity class="button stroke">), directs users to watch an explanatory video of the solution. Both are incorporated using anchor tags, one linked to different sections of the same landing page while another is hyperlinked externally to a YouTube video.

The right section of the home content integrates an with the class home-image-right. It includes an image () representing the LightXpense application on an iPhone. The srcset attribute facilitates responsive image loading, upon scroll down to add animation.

A scroll-down icon is positioned at the bottom of the landing section, encouraging users to explore further. This icon is embedded within a with the class home-scrolldown. The accompanying link (<a href="#about" class="scroll-icon smoothscroll">) facilitates smooth scrolling to the next section.

All the described code above comes together to build this front-end for the landing page which gives a quick introduction to LightXpense, alongside key elements such as preview and watch video.



About Section

```
<!-- about
================================================== -->
<section id="about">
  <div class="row about-intro">
    <div class="col-four">
      <h1 class="intro-header" data-aos="fade-up">About Us</h1>
    </div>
    <div class="col-eight">
      <p align='justify' class="lead" data-aos="fade-up">
        Our journey began with the simple idea that technology could
        revolutionize the way employees interact with their expenses, making
        reimbursement a breeze. We aim to re-invent how reimbursement is
        processed and facilitate seamless integration between HR, finance
        and accounting modules while keeping costs at a minimum.
      </p>
    </div>
  </div>

  <div class="row about-features">
    <div class="features-list block-1-4 block-m-1-2 block-mob-full group">
      <div class="bgrid feature" data-aos="fade-up">
        <span class="icon"><i class="fa-solid fa-camera-retro"></i></span>

        <div class="service-content">
          <h3>Snap & Claim</h3>

          <p align='justify'>
          Simplify claims submission by only submitting one photo and increases compliance.
          </p>
        </div>
      </div>
      <!-- /bgrid -->

      <div class="bgrid feature" data-aos="fade-up">
        <span class="icon"><i class="fas fa-robot"></i></span>

        <div class="service-content">
          <h3>RPA Supported</h3>

          <p align='justify'>
          RPA-supported platform is meticulously designed to seamlessly classify and extract grand totals, eliminating the need for extensive HR involvement.
          </p>
        </div>
      </div>
      <!-- /bgrid -->

      <div class="bgrid feature" data-aos="fade-up">
        <span class="icon"><i class="fa-solid fa-legal"></i></span>

        <div class="service-content">
          <h3>Chart of Account Compliance</h3>

          <p align='justify'>
          Our system derives Human Resources classifications in accordance with the Chart of Accounts (COA), ensuring a standardized approach across your organization. This alignment eliminates the need for Finance teams to conduct time-consuming reclassifications when transitioning information from HR to Finance.
          </p>
        </div>
      </div>

      <div class="bgrid feature" data-aos="fade-up">
        <span class="icon"><i class="fa-solid fa-chart-pie"></i></span>

        <div class="service-content">
          <h3>Visualisation</h3>

          <p align='justify'>
          Integrate claims submission with dashboard view, enabling managers to identify trends, assess risk, and make informed, precise decisions.
          </p>
        </div>
      </div>
      <!-- /bgrid -->

    </div>
    <!-- end features-list -->
  </div>
  <!-- end about-features -->
```

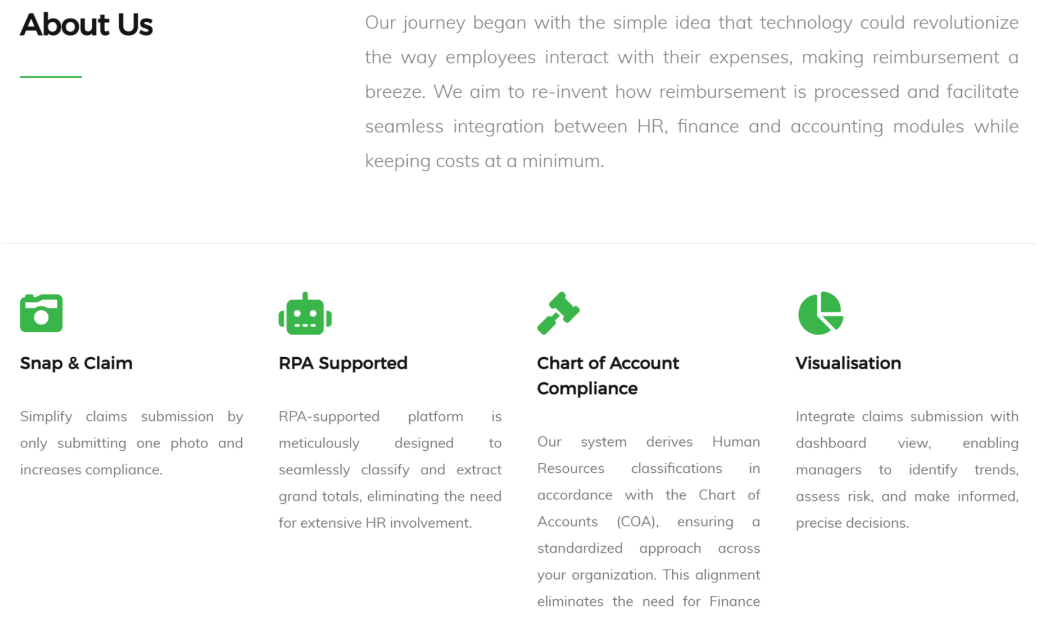
The "About" section, housed in a <section> element, provides an introduction to the LightXpense journey, vision, and key features. The about section starts with a main

paragraph, and another section which has been divided into four columns, with each column showcasing a specific feature that our solution accomplishes. Each of these columns have been designed as a bgrid, which is technical and developer friendly to amend.

The introductory content is organized in a with the class row about-intro. It consists of two columns - a four-column layout ( ) housing an impactful header (<h1 class="intro-header" data-aos="fade-up">About Us</h1>), and an eight-column layout ( ) presenting a paragraph of text ( ) aligning the text justified for readability. The paragraph within the eight-column layout narrates the journey of LightXpense.

Each feature is encapsulated in a feature block ( ). These blocks include an icon () denoting the feature, and a content section ( ) containing a heading (<h3>Snap & Claim</h3>) and a description (Simplify claims submission by only submitting one photo and increases compliance.). Icons within feature blocks, such as the camera icon, visually represent each feature for quick user comprehension.

Through the codes above, the about section is developed as shown below. The introductory paragraph on the top, divided into four blocks of information below it through the described HTML elements and CSS class.



How It Works Section

```
<section id="howitworks">
    <div class="row about-how">
        <h1 class="intro-header" data-aos="fade-up">How It Works?</h1>

        <div class="about-how-content" data-aos="fade-up">
          <div class="about-how-steps block-1-2 block-tab-full group">
            <div class="bgrid step" data-item="1">
              <h3>Sign-Up</h3>
              <p align='justify'>
                HR administrators take the initiative to create registered accounts for employees. Employees are then required to log in using the
                provided credentials, streamlining the process and ensuring secure access to the LightXpense platform.
              </p>
            </div>

            <div class="bgrid step" data-item="2">
              <h3>Upload</h3>
              <p align='justify'>
                To claim reimbursement, employees capture the essence of their transactions by taking photos of the receipts. The simplicity of the
                process lies in the ability to effortlessly upload these images onto the LightXpense platform.
                This step ensures that the necessary documentation for reimbursement is promptly submitted in a convenient and user-friendly manner.
              </p>
            </div>

            <div class="bgrid step" data-item="3">
              <h3>Create</h3>
              <p align='justify'>
                Upon successful upload of receipts, employees can delve into the LightXpense platform to create and access detailed information
                related to their reimbursement claims. This includes the ability to input and review relevant details, fostering transparency and
                accountability in the reimbursement workflow.
              </p>
            </div>

            <div class="bgrid step" data-item="4">
              <h3>Publish</h3>
              <p align='justify'>
                Notifications regarding the status of reimbursement claims are automatically generated within LightXpense. A centralized dashboard
                facilitates the collaborative effort among employees, the HR department, and the Finance department. This shared platform allows
                stakeholders to publish, review, and manage claim information collectively, ensuring a streamlined and efficient process for all
                involved parties.
              </p>
            </div>
```

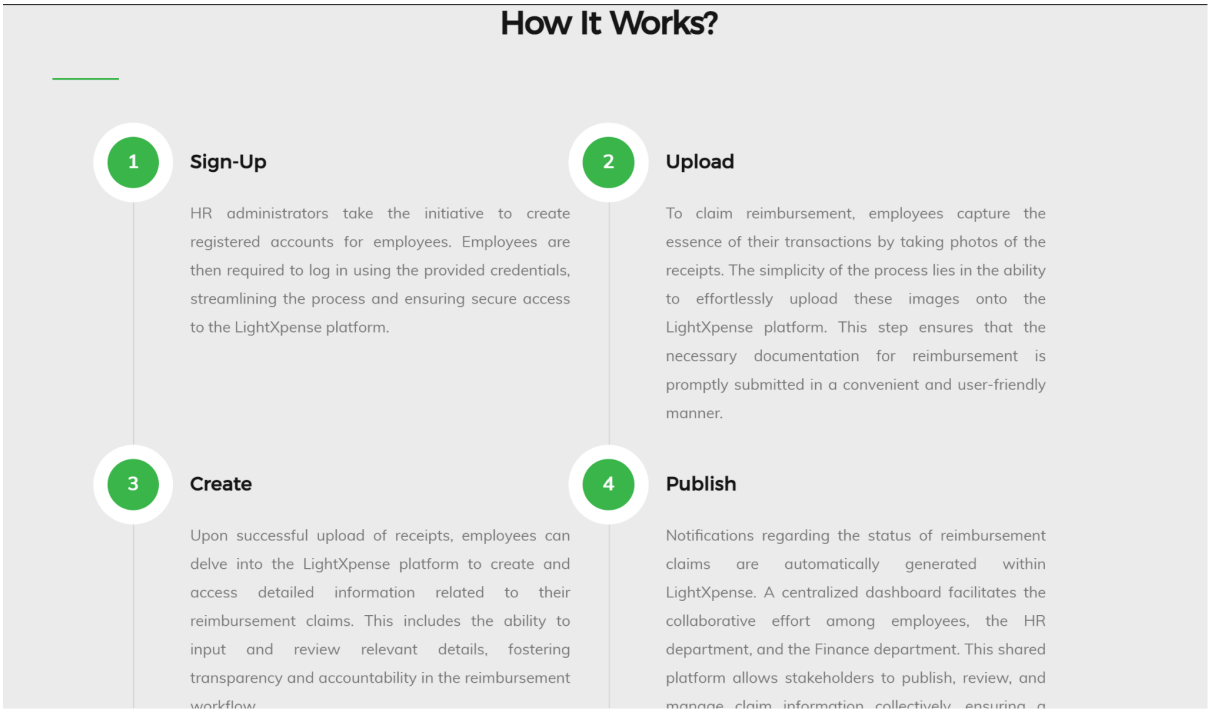
```
              </p>
            </div>
          </div>
        </div>
        <!-- end about-how-content -->
    </div>
    <!-- end about-how -->

    <!-- end about-bottom-image -->
</section>
```

This HTML and CSS code snippet defines the structure of the "How It Works" section within the LightXpense web application. The "How It Works' ' section, encapsulated in a <section> element with the id howitworks, shows the step-by-step process for users to navigate the LightXpense platform.

The entire "How It Works" section is encapsulated within a <section> element with the id attribute set to howitworks. This ID serves as an anchor point for linking within the page, as well as setting section specific design in the css files. The introduction is presented using a with the class row about-how. It includes a header (<h1 class="intro-header" data-aos="fade-up">How It Works?</h1>) with a fade-up animation effect. The content is housed in another with the class about-how-content.

The step-by-step process is organized within a with the class about-how-steps. Individual steps are represented by elements with the class bgrid step. Each step has a data attribute (data-item) for ordering. Each step component includes an <h3> for the step heading and a <p align='justify'> for the descriptive paragraph. These components are further styled with the data-aos attribute to add a fade-up animation effect. These blocks can then be duplicated to as many steps as needed, while also changing the attribute needed to make the design.

With the codes above, and the design attributes from Bootstrap, the how it works section is designed as shown below.



Our Team Section

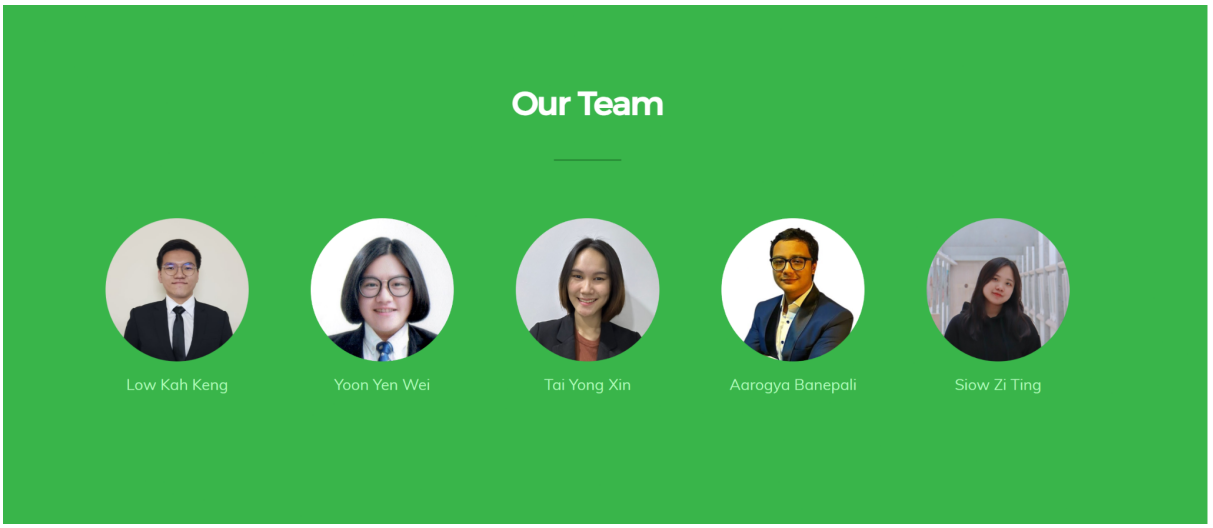
```
<!-- our team
================================================ -->
<section id="ourteam">
  <div class="row">
    <div class="col-full">
      <h1 class="intro-header" data-aos="fade-up">
        Our Team
      </h1>
      <div class="team">
        <div class="team-member">
          <div class="team-name">
          <img src="images/LKK.png">
          <p>Low Kah Keng</p>
        </div>
        <div class="team-name">
          <img src="images/YYW.png">
          <p>Yoon Yen Wei</p>
        </div>
        <div class="team-name">
          <img src="images/TYX.png">
          <p>Tai Yong Xin</p>
        </div>
        <div class="team-name">
          <img src="images/AB.png">
          <p>Aarogya Banepali</p>
        </div>
        <div class="team-name">
          <img src="images/SZT.png">
        <p>Siow Zi Ting</p>
        </div>
        </div>
      </div>
    </div>
  </div>
</section>
<!-- end our team -->
```

The provided code snippet represents the "Our Team" section of a website and is composed of HTML, CSS, and JavaScript elements. In the HTML portion, a section with the ID "ourteam" is defined, encapsulating a row and column structure for layout purposes. Within this structure, there is an <h1> heading with the class "intro-header," employing the AOS (Animate On Scroll) library for a fade-up animation effect. The team members are presented using a series of elements within a container with the class "team." Each team member is further encapsulated in a "team-member" class, containing a "team-name" class that includes an image of the team member, and the team member's name.

Regarding the CSS, the code snippet includes placeholder styles for various components such as "#ourteam," ".row," ".col-full," ".intro-header," ".team," ".team-member," and ".team-name." These styles serve as a foundation for formatting and layout, and it's essential to customize them based on the specific design requirements of the website. All these classes and ids are used in the CSS file to make specific styling possible in the front-end.

For the JavaScript portion, it utilizes the AOS library for scroll-triggered animations. It initializes the AOS library with a duration of 1000 milliseconds, contributing to the fade-up animation effect associated with the team heading.

With the codes above, the following our team section in the website is accomplished.



c. Footer

```
<!-- footer
    ================================================== -->
<footer>
  <div class="footer-main">
    <div class="row">
      <div class="col-three md-1-3 tab-full footer-info">
        <div class="footer-logo"></div>

        <p align='justify'>
          We introduce a cutting-edge budgeted solution for corporate reimbursement processes. Our advanced system
          efficiently auto-classifies expenses, providing companies with a powerful dashboard for enhanced financial
          management and streamlined operational efficiency. Elevate your company's reimbursement workflows with
          LighXpense.</p>

        <ul class="footer-social-list">
          <li>
            <a href="#"><i class="fa fa-facebook-square"></i></a>
          </li>
          <li>
            <a href="#"><i class="fa fa-twitter"></i></a>
          </li>
          <li>
            <a href="#"><i class="fa fa-behance"></i></a>
          </li>
          <li>
            <a href="#"><i class="fa fa-dribbble"></i></a>
          </li>
          <li>
            <a href="#"><i class="fa fa-instagram"></i></a>
          </li>
        </ul>
      </div>
      <!-- end footer-info -->
```

The overarching structure for footer is encapsulated within the <footer> tag, and the primary content is organized under the "footer-main" class. This class further delineates the content into distinct sections, employing a grid-based layout for responsiveness. The first significant division is the "Footer Information Section," denoted by the class "col-three md-1-3 tab-full footer-info." This section accommodates a logo and a descriptive paragraph, utilizing the align='justify' attribute for justified text alignment. Additionally, social media links are

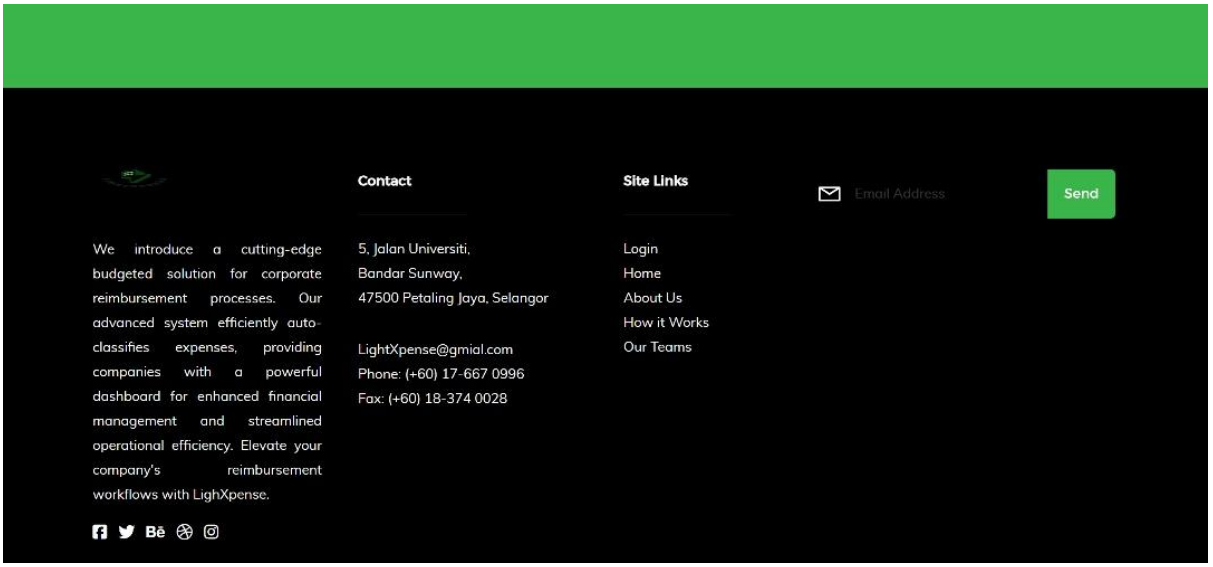
integrated into the section through an unordered list (<ul>) with list items (<li>) and anchor (<a>) elements.

Moving on to the "Contact Information Section," represented by the class "col-three md-1-3 tab-1-2 mob-full footer-contact," it includes details such as the physical address, email, phone, and fax contacts. The classes help design the front-end based on the requirement of the information to be demonstrated. The information is presented within paragraph elements (<p>), ensuring a clear and organized display.

The subsequent section, labeled "Site Links Section" and identified by the class "col-two md-1-3 tab-1-2 mob-full footer-site-links," presents a list of site links using an unordered list (<ul>) with list items (<li>) and anchor (<a>) elements. These links facilitate navigation to key sections of the website, such as login, home, about us, how it works, and our teams.

The final section is the "Subscription Section," denoted by the class "col-four md-1-2 tab-full footer-subscribe." Within this section, a subscription form is embedded, featuring an email input field, a "Send" button, and a label for potential messages. This form is designed to collect user email subscriptions, contributing to communication strategies.

The footer section then looks like this in the front-end through the codes we have developed and described above.



d. Login screen

From there, users proceed to the login screen, where they are prompted to enter their credentials. The login screen intelligently distinguishes between two user categories: employees and administrators.

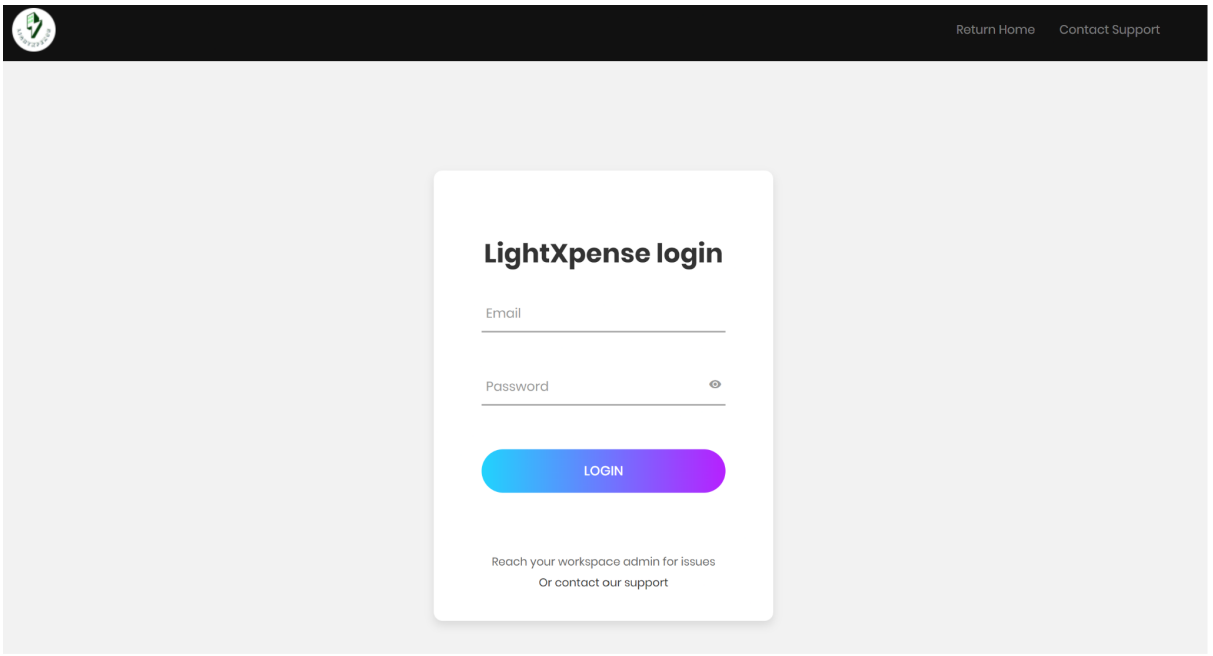
a. Employees: Regular users with specific access privileges to:
    i. View their own expense claim KPIs,
    ii. Claim a new expense by submitting an image of a receipt,
    iii. View and edit their own past expense claims

b. Admin: Administrators with elevated permissions to:
    i. View expense claim KPIs from all employees,
    ii. Add or remove employee accounts,
    iii. View and edit all expense claims from all employees.

This has been accomplished through back-end code, which will be demonstrated in the backend section below. For now, the front-end codes that handle user input and error messages are as such:

```
<div class="limiter">
  <div class="container-login100">
    <div class="wrap-login100">
      <form class="login100-form validate-form" method="POST">
        <span class="login100-form-title p-b-26"> LightXpense login </span>

        <div
          class="wrap-input100 validate-input"
          data-validate="Valid email is: a@b.c"
        >
          <input class="input100" type="text" name="email" />
          <span class="focus-input100" data-placeholder="Email"></span>
        </div>

        <div
          class="wrap-input100 validate-input"
          data-validate="Enter password"
        >
          <span class="btn-show-pass">
            <i class="zmdi zmdi-eye"></i>
          </span>
          <input class="input100" type="password" name="pass" />
          <span class="focus-input100" data-placeholder="Password"></span>
        </div>
```

```
<div class="container-login100-form-btn" style="margin-bottom:1rem">
  <div class="wrap-login100-form-btn">
    <div class="login100-form-bgbtn"></div>
    <button class="login100-form-btn" type="submit" name="submit">Login</button>
  </div>
</div>
            <!-- Error Message Display -->
            <?php if (!empty($errors)): ?>
              <div class="alert alert-danger alert-dismissible fade show" role="alert">
              <button type="button" class="close" data-dismiss="alert" aria-label="Close">
                  <span aria-hidden="true">&times;</span>
                </button>
                <?php foreach ($errors as $error): ?>
                  <strong>Error <br></strong> <?php echo $error; ?><br>
                <?php endforeach; ?>
              </div>
            <?php endif; ?>

<div class="text-center p-t-50">
  <span class="txt1">
    Reach your workspace admin for issues <br>
  </span>

  <a class="txt2" href="mailto:aarogya@banepali.com"> Or contact our support</a>
</div>
```



The input form for email and password is created as such on the front-end. Several predefined classes through jquery have been used to make the form interactive, and to show errors when email isn't the correct format, or if the password doesn't meet the requirements. Lastly, there is also a div to show error in the front-end, should there be any issues with login.

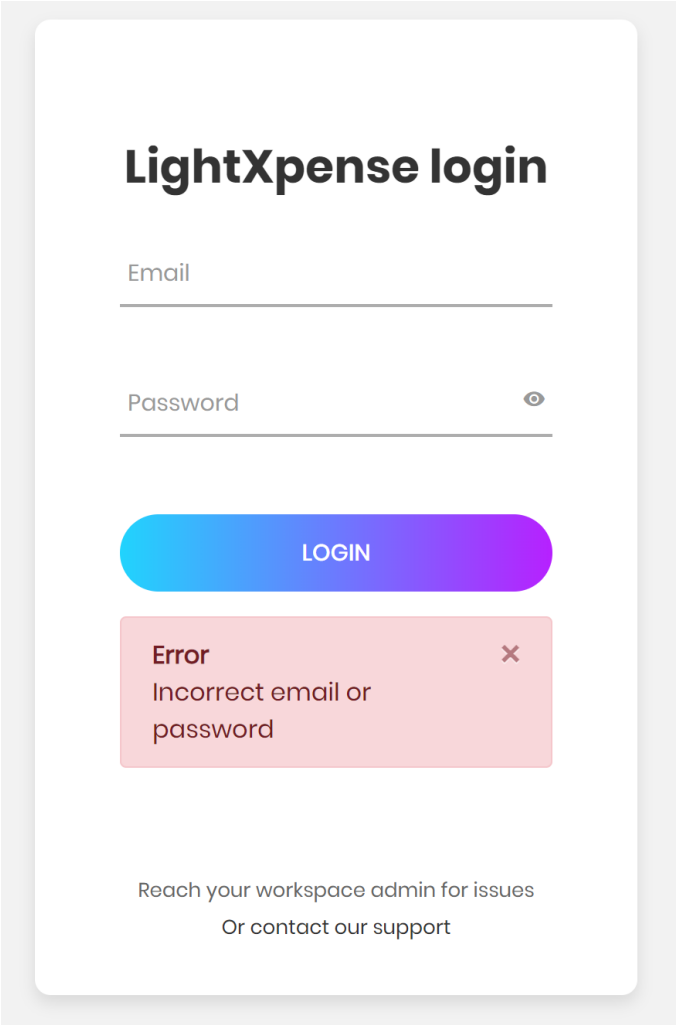
Going back to a high-level, the implementation of this has been done as such:

1. User Authentication and Redirection

   Upon entering their credentials, the front-end seamlessly communicates with the back end to authenticate the user. Clean designs and informative error messages have been

implemented to handle various scenarios, such as incorrect passwords or invalid email addresses. If the login is successful, users are gracefully redirected to their respective accounts based on their roles.

The image below shows an example of an error when someone tries to login through incorrect or non-existent details. It has been accomplished through the block underneath the <!- - error message display > comment.



2. Admin Account

The system boasts two distinct dashboard designs catering to the specific needs of administrators and employees. The base file consists of the nav bar with links to the three modules listed below. The codes for the base file are as follows:

```php
<?php

include("include/mysql.php");

session_start(); // Resume the session

// Check if the user is logged in
if (!isset($_SESSION['email'])) {
    header('location: login.php'); // Redirect to login page if not logged in
    exit();
}

// $db = new myConnection;

?>

<!doctype html>
<html Lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>LightXpense</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
    <script
        src="https://code.jquery.com/jquery-3.6.0.js"
        integrity="sha256-H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk="
        crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://cdn.datatables.net/1.12.1/js/jquery.dataTables.min.js"></script>
    <script src="https://cdn.datatables.net/1.12.1/js/dataTables.bootstrap5.min.js"></script>
    <link rel="stylesheet" href="css/dashboard.css">
    <link rel="icon" type="image/png" href="images/logo2.png"/>

```

The php section starts with include(“include/mysql.php”);, which pre-loads a php file that establishes connection with the mySQL server and creates function classes needed for the program. The mysql.php file connection alongside an example of the function defined on that file is shown below:

```php
<?php

class myConnection
{
    // __construct run after call myConnection
    function __construct(){

        $host = "localhost";
        $user = "root";
        $pass = "";
        $db = "lightXpense";

        $conn = $this->mysqli = new mysqli($host,$user,$pass,$db);

        if ($conn->connect_errno) {
            printf("Connect failed: %s\n", $conn->connect_error);
            exit();
        }


    }

    // Takes query as an argument and returns the output of the SQL query from the database connected
    function query($query){
        return $this->mysqli->query($query);
    }
```

The snippet above shows the included mysql.php file, which is linked in most php files as a way to initiate database connection and introduce base-line functions needed for the web app. The class myConnection holds the functions needed to initiate connection with the mySQL server and various functions that have been used to facilitate integration between the web-app and the database. For instance, the function __construct() initiates connection with the SQL database and holds the information such as host IP, user, password and database name needed to start the connection. Next, an example of function can be seen in the form of query(), which takes in a SQL formatted query as an argument, and outputs the result of that query from the mySQL database that has been connected. This is used in all aspects when the front-end needs to get any value from the database. A specific example can be when the dashboard needs to receive data of all employee claims through a SQL query.

Next, on the admin page, it also checks if the session email is set, to ensure no one can access the admin dashboard without going through the proper login flow. The command

```
if (!isset($_SESSION['email'])) {
    header('location: login.php'); // Redirect to login page if not logged in
    exit();
}
```

Checks if an email session is set, and if it is not set (which means that the value of $_SESSION['email'] is null, then the user is redirected back to the login page.

The <head> tag encloses important links required for the website design and functionality. It establishes the title through <title>LightXpense</title> tag, and sets up a favicon through <link rel="icon" type="image/png" href="images/logo2.png"/> tag, which makes the website look professional in the web browser tab bar as shown below.

Lastly, within the <head> tag, a custom CSS file is also linked which handles design of the admin page. This is done through <link rel="stylesheet" href="css/dashboard.css">, which links dashboard.css stored inside the css folder in the project directory, which applies specific styles tailored to the admin dashboard.

```
    <style>
      .bd-placeholder-img {
        font-size: 1.125rem;
        text-anchor: middle;
        -webkit-user-select: none;
        -moz-user-select: none;
        user-select: none;
      }

      @media (min-width: 768px) {
        .bd-placeholder-img-lg {
          font-size: 3.5rem;
        }
      }
    </style>


    <!-- Custom styles for this template -->
    <link href="dashboard.css" rel="stylesheet">
  </head>
  <body>

  <header class="navbar navbar-dark sticky-top bg-dark flex-md-nowrap p-0 shadow">
  <a class="navbar-brand col-md-3 col-lg-2 me-0 px-3" href="#">Welcome Admin!</a>
  <button class="navbar-toggler position-absolute d-md-none collapsed" type="button" data-bs-toggle="collapse" data-bs-target="#sidebarMenu"
  aria-controls="sidebarMenu" aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <input class="form-control form-control-dark w-100" type="text" placeholder="Search" aria-label="Search">
  <div class="navbar-nav">
    <div class="nav-item text-nowrap">
      <a class="nav-link px-3" href="signout.php">Sign out</a>
    </div>
  </div>
</header>
```

Within the <body> of the core admin page, the navigation bar on the top of the website is first established. This is done through the <header> element that is assigned the Bootstrap classes navbar, navbar-dark, sticky-top, bg-dark, flex-md-nowrap, p-0, and shadow, collectively defining a dark-themed, fixed-top navigation bar with a flexible layout. The navigation bar includes a brand link represented by the <a> element with the class navbar-brand, displaying "Welcome Admin!" and spanning varying column widths for different screen sizes (col-md-3

and col-lg-2). Additionally, a collapsible toggle button is implemented using the <button> element with the classes navbar-toggler, position-absolute, and d-md-none, ensuring its visibility is limited to smaller screens. This button utilizes Bootstrap's built-in collapse functionality (data-bs-toggle and data-bs-target) to control the visibility of the sidebar menu. An input field with the classes form-control, form-control-dark, and w-100 is integrated for search functionality. The with the class navbar-nav encapsulates a navigation link for signing out, structured as a with the classes nav-item and text-nowrap, containing an <a> element styled as a navigation link (nav-link) with additional padding (px-3). In summary, the code employs Bootstrap classes and elements to craft a dynamic and visually cohesive navigation bar, catering to both desktop and mobile interfaces, while also incorporating essential features such as branding, search functionality, and a sign-out link.

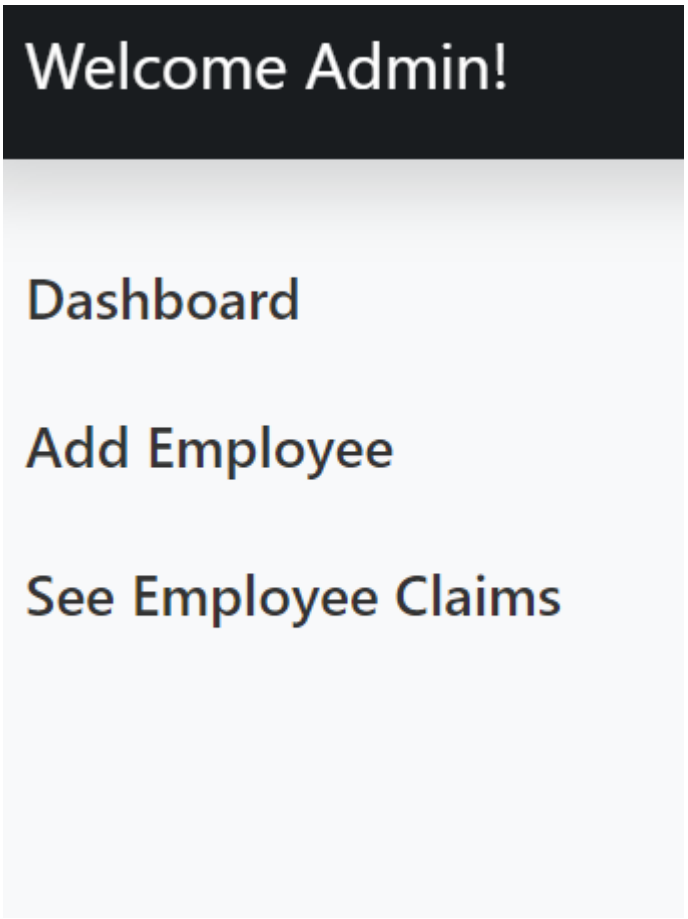
On the front end, through the code above, the output can be seen as such:


```
<div class="container-fluid">
  <div class="row">
    <nav id="sidebarMenu" class="col-md-3 col-lg-2 d-md-block bg-light sidebar collapse">
      <div class="position-sticky pt-3">
        <ul class="nav flex-column">
        <li class="nav-item">
            <a class="nav-link" aria-current="page" href="admin.php?module=dashboard&page=admin">
              <span data-feather="home"></span>
              Dashboard
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="admin.php?module=Add%20Employee&page=add_employee">
              <span data-feather="file"></span>
              Add Employee
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="admin.php?module=See%20All%20Claims&page=claims">
              <span data-feather="file"></span>
              See Employee Claims
            </a>
          </li>
        </ul>


      </div>
    </nav>
```

Next, the <nav> tag created a vertical navigation bar on the website that exists on the left side of the screen. This <nav> tag constructs a sidebar navigation menu within the HTML document. Identified by the ID "sidebarMenu," encompasses the Bootstrap classes col-md-3, col-lg-2, d-md-block, bg-light, sidebar, and collapse." This structure ensures that the sidebar

is displayed as a narrow column on medium to large screens but is initially collapsed on smaller screens. The inner element is designated asposition-sticky with top padding (pt-3), facilitating a fixed positioning for the contained content. Within this div, an unordered list (<ul>) with the classes navandflex-column is defined, configuring a vertical arrangement of navigation items. Each list item (<li>) is marked as a nav-item and contains an anchor (<a>) element styled as a navigation link (nav-link). These links direct to different modules and pages within the admin section, utilizing PHP parameters for dynamic content loading. Each link includes an icon represented by the element with thedata-feather` attribute, enhancing visual clarity. In summary, the code establishes a responsive sidebar navigation menu, incorporating Bootstrap classes to adapt to various screen sizes and providing structured links for seamless navigation throughout the admin interface.

The code establishes the side-bar in the front-end as shown in the image below. Each of the buttons such as ‘Dashboard’, ‘Add Employee’ and ‘See Employee Claims’ are clickable and bring the admin to different aspects of the admin screen for different purposes.



```
    <main class="col-md-9 ms-sm-auto col-lg-10 px-md-4">
      <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-items-center pt-3 pb-2 mb-3 border-bottom">
      <h1 class="h2"><?php echo ucfirst ($_GET['module']);?></h1>
      </div>

    <?php
        $module = $_GET['module'];
        $page = $_GET['page'];
        if(file_exists("module/".$module."/".$page.".php")){
            include("module/$module/$page.php");
        } else {
            echo "Error 404";
        }

    ?>
    </main>
  </div>
</div>



    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP
    +IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

  </body>
</html>
```

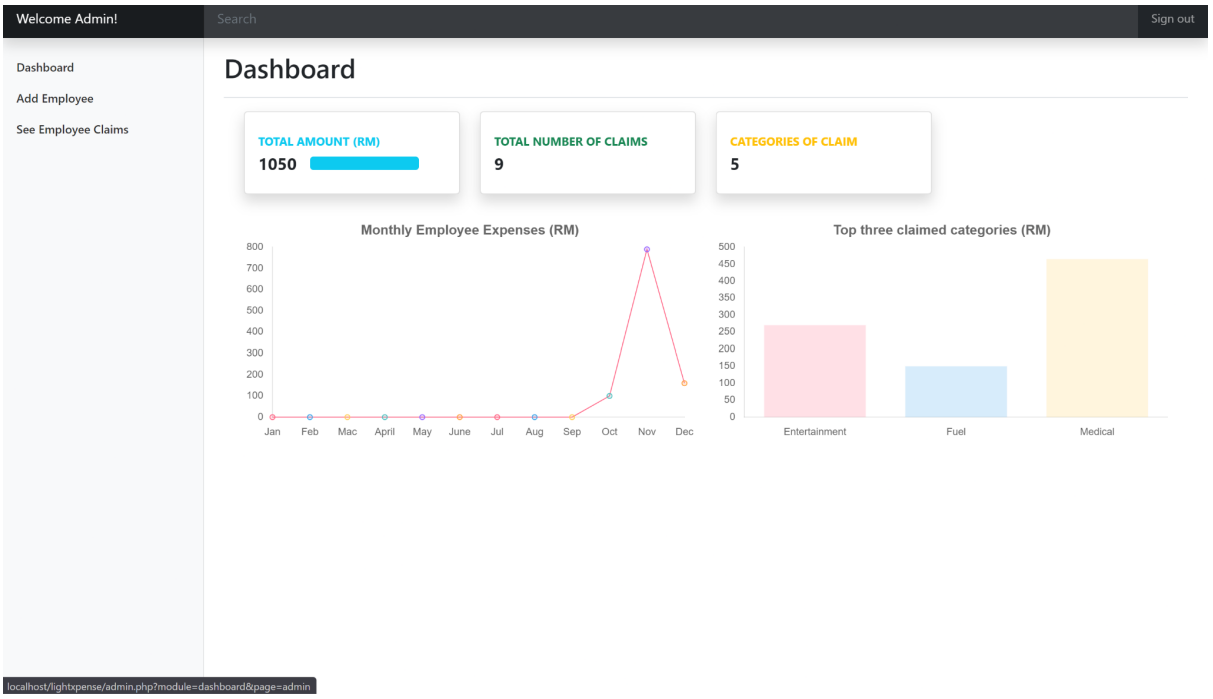
Lastly, this block of code manages the main content area of the web page, offering a dynamic display based on the specified module and page parameters. The <main> element is assigned Bootstrap classes (col-md-9, ms-sm-auto, col-lg-10, px-md-4), defining the layout and padding for the main content. Within this element, a is configured with classes (d-flex, justify-content-between, flex-wrap, flex-md-nowrap, align-items-center, pt-3, pb-2, mb-3, border-bottom), establishing a flexible and responsive container with a border-bottom for visual separation.

The <h1> heading inside this div is styled with the class h2 and dynamically displays the capitalized module name retrieved from the URL parameters using PHP (ucfirst($_GET['module'])). This creates a heading that corresponds to the currently selected module, contributing to the contextual clarity of the page. Moreover, the PHP code block underneath retrieves the module and page parameters from the URL. It then checks whether the corresponding PHP file exists within the "module" directory. If found, the code dynamically includes and renders the content of the specified module and page using include("module/$module/$page.php"). In case the file is not found, an "Error 404" message is echoed, indicating the absence of the requested content. For instance, modules such as ‘dashboard’ and ‘add employee’ have been implemented which can be accessed by clicking on them on the side bar.

In essence, this code segment orchestrates a responsive and contextually relevant main content area by dynamically adjusting the displayed module name in the heading and including content based on the selected module and page parameters. This modular approach

enhances the flexibility and scalability of the web application, allowing for the seamless addition of new modules and pages.

The screenshot below shows how the main content looks. The anchor texts ‘Dashboard’, ‘Add Employees’ and ‘See Employee Claims’ are all clickable which change the main content shown in the screen for the admin. The codes above enable this implementation through the modular concept.



a. Admin Dashboard page

```php
<?php


if(isset($_SESSION['email'])) {
    $email = $_SESSION['email'];
    $db = new myConnection();

    $totalAmount = $db->total_amount_admin($email);
    $numberClaims = $db->number_claims_admin($email);
    $categoryCounts = $db->count_distinct_categories_admin($email);

} else {
    header('location:login.php');
}

?>

<div class="container-fluid">
    <div class="row">
            <div class="col-md-6 col-xl-3 mb-4">
            <div class="card shadow border-start-info py-2">
                <div class="card-body">
                    <div class="row align-items-center no-gutters">
                        <div class="col me-2">
                            <div class="text-uppercase text-info fw-bold text-xs mb-1"><span>Total Amount (RM)</span></div>
                            <div class="row g-0 align-items-center">
                                <div class="col-auto">
                                    <div class="text-dark fw-bold h5 mb-0 me-3"><span><?php echo $totalAmount; ?></span></div>
                                </div>
                                <div class="col">
                                    <div class="progress progress-sm">
                                        <div class="progress-bar bg-info" aria-valuenow="50" aria-valuemin="0" aria-valuemax="100" style="width: 100%;
                                        "><span class="visually-hidden">50%</span></div>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="col-auto"><i class="fas fa-clipboard-list fa-2x text-gray-300"></i></div>
                    </div>
                </div>
            </div>
```

PHP code block is responsible for retrieving and processing information related to the user's session, specifically focusing on data associated with the administrative user. The conditional statement if(isset($_SESSION['email'])) checks whether the 'email' key is set in the session, implying that a user is currently logged in. If this condition holds true, the code proceeds to extract the user's email address from the session, instantiates a new connection to the database using the myConnection class, and subsequently invokes several database methods to gather pertinent information.

Within the conditional block, the variable $email is assigned the value of the user's email address stored in the session. A new database connection ($db) is established through the myConnection class. Subsequently, three distinct pieces of information are retrieved:

1. $totalAmount: The total amount, likely representing a cumulative financial value associated with the administrative user. This value is obtained through the total_amount_admin method of the database connection.
2. $numberClaims: The number of claims initiated by the administrative user, acquired using the number_claims_admin method.

3. $categoryCounts: A count of distinct categories relevant to the administrative user, obtained through the count_distinct_categories_admin method.

Conversely, if the 'email' key is not set in the session, the code redirects the user to the login page (header('location:login.php')). This conditional redirection ensures that the administrative functionalities are only accessible to authenticated users, reinforcing the security and integrity of the web application

After that, there are PHP and HTML codes that encapsulates the construction of a dynamic card visualization within an administrative dashboard, designed to represent the total financial amount for expense claims. The visualization employs Bootstrap grid system classes to ensure responsiveness and consistent layout. Through the strategic use of Bootstrap and Font Awesome classes, the card achieves a visually appealing and informative presentation of real-time data. Some of the major components for it are as follows:

1. Card Structure: The card is enveloped within a Bootstrap grid system, specifically a element, with a subsequent defining its responsiveness and layout characteristics. This structural foundation sets the stage for a visually cohesive presentation.

2. Card Styling: Bootstrap classes such as card, shadow, border-start-info, and py-2 contribute to the visual aesthetics of the card, incorporating shadows, border coloring, and padding. These styling elements collectively enhance the user experience and maintain a harmonious design.

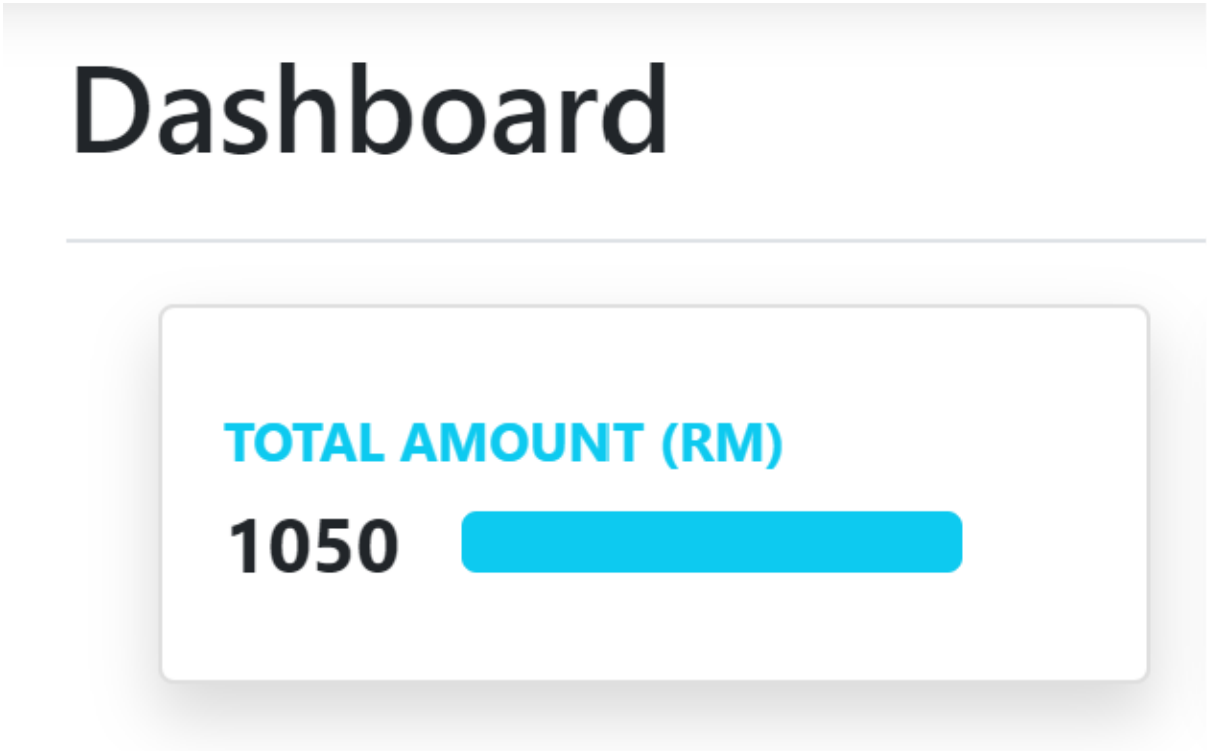
3. Header and Body Content: The card's header intelligently organizes the displayed data metric, "Total Amount (RM)," employing Bootstrap classes like text-uppercase, text-info, fw-bold, and text-xs for clear emphasis. Meanwhile, the body dynamically echoes the total amount ($totalAmount) obtained from the database within a element, ensuring the real-time representation of financial data.

4. Progress Bar: A Bootstrap-styled progress bar, located within the card's body, visually represents the data metric. Its width is dynamically adjusted based on the data value (width:

100%), offering a proportional visualization. Bootstrap classes such as progress, progress-sm, and bg-info enhance the aesthetic appeal and clarity of the progress bar.

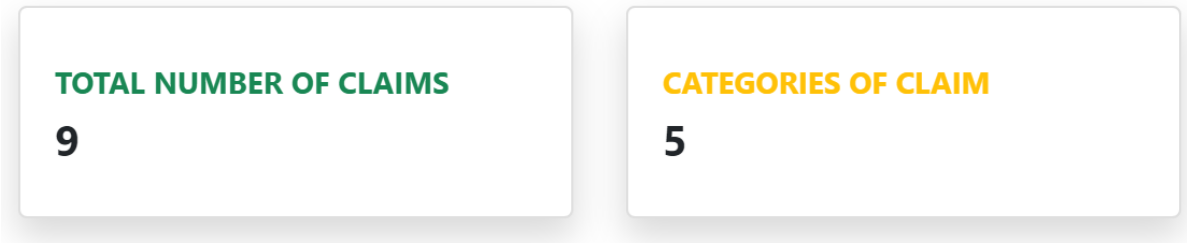
5. Icon Representation: Complementing the card, an icon represented by provides a visual cue related to the nature of the presented financial data.

With these, the card which has been created is:



```
            </div>
        </div>
        <div class="col-md-6 col-xl-3 mb-4">
            <div class="card shadow border-start-success py-2">
                <div class="card-body">
                    <div class="row align-items-center no-gutters">
                        <div class="col me-2">
                            <div class="text-uppercase text-success fw-bold text-xs mb-1"><span>Total Number of Claims</span></div>
                            <div class="text-dark fw-bold h5 mb-0"><span><?php echo $numberClaims; ?></span></div>
                        </div>
                        <div class="col-auto"><i class="fas fa-dollar-sign fa-2x text-gray-300"></i></div>
                    </div>
                </div>
            </div>
        </div>
        <div class="col-md-6 col-xl-3 mb-4">
            <div class="card shadow border-start-warning py-2">
                <div class="card-body">
                    <div class="row align-items-center no-gutters">
                        <div class="col me-2">
                            <div class="text-uppercase text-warning fw-bold text-xs mb-1"><span>Categories of Claim</span></div>
                            <div class="text-dark fw-bold h5 mb-0"><span><?php echo $categoryCounts; ?></span></div>
                        </div>
                        <div class="col-auto"><i class="fas fa-comments fa-2x text-gray-300"></i></div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="row">
        <div class="col" >
            <canvas id="myChart"></canvas>
        </div>
        <div class="col" >
            <canvas id="myChart2"></canvas>
        </div>
    </div>
</div>
<?php
    // the easiest way to do it but this cost a lot of resources. can change to Looping and array.
    $data_jan = 0;
```

These codes also work in a similar way as described above to make two additional cards to showcase important KPIs in the dashboard. Only the progress bar has been removed for these two cards to show an effective visualization without overcrowding.



```php
    $data_jan = 0;
    $data_feb = 1;
    $data_mac = 3;
    $data_apr = 4;
    $data_may = 5;
    $data_jun = 6;
    $data_jul = 7;
    $data_aug = 8;
    $data_sep = 9;
    $data_oct = 10;
    $data_nov = 11;
    $data_dec = 12;

    $data_by_month = $db->data_chart_by_month_admin();
    $data_month = array();
    foreach($data_by_month as $key => $dtbm){
        $data_month[$key] = $dtbm;
    }

    $statuses = $db->category();
    $st = array();
    foreach($statuses as $s => $stt){
        $st[$s] = $stt;
    }

    $status_inv = $db->data_chart_by_category_admin();
    $data_status = array();
    foreach($status_inv as $bil => $dcbs){
        $data_status[$bil] = $dcbs;
    }

?>

<script>
const ctx = document.getElementById('myChart').getContext('2d');
const myChart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: ['Jan','Feb','Mac','April','May','June','Jul','Aug','Sep','Oct','Nov','Dec'],
        datasets: [{
            label: '',
```

This PHP code snippet employs efficient data processing techniques to facilitate the retrieval and organization of dynamic data for chart visualizations within the administrative dashboard. Focused on monthly and category-based metrics, the code transforms individual variables into a more scalable array-based structure.

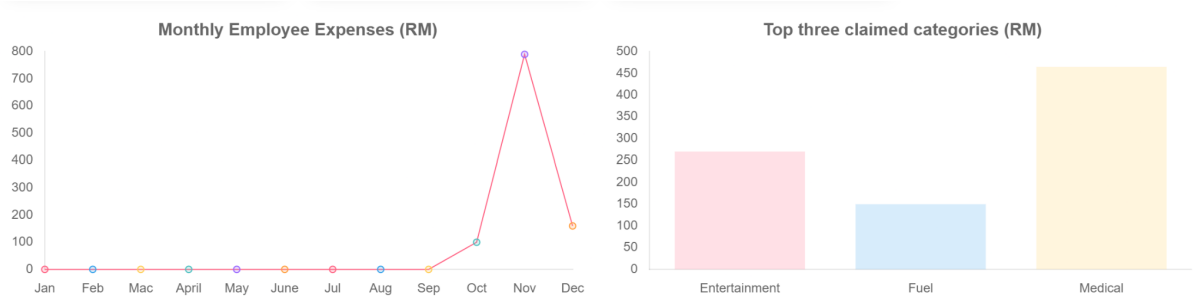
1. Monthly Data Mapping: The initial section of the code assigns individual variables to represent each month, providing a rudimentary but resource-intensive approach. However, recognizing the need for a more scalable and efficient solution, the subsequent code introduces an array-based approach to map data by month. A loop iterates through the retrieved data, creating an associative array ($data_month) that holds monthly data values, dynamically adapting to changes in data structure or volume.

2. Category Data Mapping: Similarly, the code fetches and processes category-related data through a loop, resulting in an associative array ($st) representing various statuses. The status categories are obtained through a call to $db->category().

3. Monthly Status Data Mapping: The code further retrieves monthly data categorized by status through the $db->data_chart_by_category_admin() method. This information is processed in a loop, resulting in an associative array ($data_status). This array efficiently organizes data, paving the way for streamlined chart visualizations.

4. Database Queries: Database interactions are prominent in the code, demonstrated by calls to methods such as $db->data_chart_by_month_admin(), $db->category(), and $db->data_chart_by_category_admin(). These methods fetch essential data from the database, allowing for the real-time representation of metrics in the administrative dashboard.

With the explanation above, we are able to achieve the following graphs on the dashboard



b. Add Employee page

```php
<?php

include 'db_connection.php';

if(isset($_POST['submit'])){
   $firstName = mysqli_real_escape_string($conn, $_POST['firstName']);
   $lastName = mysqli_real_escape_string($conn, $_POST['lastName']);
   $email = mysqli_real_escape_string($conn, $_POST['email']);
   $phone = mysqli_real_escape_string($conn, ($_POST['phone']));
   $password = mysqli_real_escape_string($conn, ($_POST['password']));
   $department = mysqli_real_escape_string($conn, ($_POST['department']));

   $select_users = mysqli_query($conn, "SELECT * FROM `user_details` WHERE email = '$email' AND password = '$password'") or die('query failed');

   if(mysqli_num_rows($select_users) > 0){
        $error_message = 'User already exists!';
   }else{
        mysqli_query($conn, "INSERT INTO `user_details`(first_name, last_name, email, phone_number, password, department) VALUES ('$firstName',
        '$lastName', '$email', '$phone', '$password', '$department')") or die('query failed');
        $success_message = 'Registered successfully!';
   }
}

?>
```

This PHP code snippet orchestrates user registration logic within a web application, interfacing with a MySQL database. Upon the submission of a registration form, the code processes the user-provided information, escapes potential SQL injection vulnerabilities, and validates the uniqueness of the user by querying the database. Here are some fundamental points of the working of this code.

1. Database Connection: The code initiates by including a database connection file (db_connection.php). This file contains essential configurations for establishing a connection to the MySQL database, facilitating subsequent interactions.

2. Form Data Processing: Upon the submission of the registration form (isset($_POST['submit'])), the code captures and sanitizes user inputs such as first name, last name, email, phone number, password, and department. The mysqli_real_escape_string function is utilized to prevent SQL injection attacks, ensuring the integrity of the entered data.

3. Database Query - User Existence Check: A crucial aspect of the code involves querying the database to ascertain the existence of the user. The query (SELECT * FROM user_details WHERE email = '$email' AND password = '$password') checks for any records matching the provided email and password combination. If the query returns rows (mysqli_num_rows($select_users) > 0), it signifies that the user already exists, triggering an error message.

4. Database Query - User Registration: In the absence of existing user records, the code proceeds to execute an INSERT query to add the user's details to the database. This is achieved through the mysqli_query function, which adds a new record to the user_details table, including the user's first name, last name, email, phone number, password, and department.

5. Error and Success Handling: The code includes error handling mechanisms to manage potential failures in the database queries. If a user with the same email and password already exists, an error message ($error_message) is generated. Conversely, if the registration is successful, a success message ($success_message) is set.

For each new employee that an admin aims to register, the above PHP code runs and it is completely handled. After it is complete, the codes below work on displaying the front-end to the users.

```
<!-- Display Error or Success Message -->
<div class="message-container">
    <?php
    if(isset($error_message)){
        echo '
        <div class="message error-message">
            <span>'.$error_message.'</span>
            <i class="fas fa-times close-button" onclick="this.parentElement.remove();"></i>
        </div>
        ';
    }

    if(isset($success_message)){
        echo '
        <div class="message success-message">
            <span>'.$success_message.'</span>
            <i class="fas fa-times close-button" onclick="this.parentElement.remove();"></i>
        </div>
        ';
    }
    ?>
</div>
```

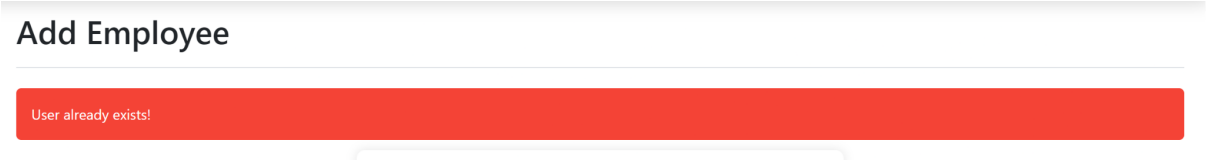
The code above dynamically generates messages based on the outcomes of the registration process, such as success or error.

1. Feedback Container Structure: The code introduces an HTML element with the class message-container. This container serves as the encapsulation for error or success

messages, ensuring a consistent and visually cohesive presentation within the web application interface.

2. PHP Conditionals for Message Display: Within the feedback container, PHP conditionals evaluate the existence of error ($error_message) or success ($success_message) messages. If an error message exists, an HTML structure for an error message is dynamically generated. Similarly, if a success message is present, an HTML structure for a success message is generated. This is coded through the if() block, which checks whether an $error_message or a $success_message variable is set from the admin input before.

3. Error Message Structure: In the event of an error message, the PHP code dynamically echoes HTML code for an error message. The error message is visually styled with the class message error-message. It includes a element to display the error message text and an <i> element with the class fas fa-times close-button. This close button is an interactive feature, allowing users to dismiss the error message by clicking on it. The error message displayed would look like this:



4. Success Message Structure: If a success message exists, the PHP code dynamically echoes HTML code for a success message. The success message is styled with the class message success-message and includes a element for displaying the success message text. Similar to the error message, a close button (<i> element with the class fas fa-times close-button) is included for user interaction. The success message displayed would look like this:

5. Interactive Message Removal: Both error and success messages incorporate a close button with an onclick attribute, calling a JavaScript function (onclick="this.parentElement.remove();") to remove the respective message container when

clicked. This provides users with a seamless and interactive way to dismiss feedback messages.

```
<div class="container">
<form method="post">
        <label for="firstName">First Name:</label>
        <input type="text" id="firstName" name="firstName" required>

        <label for="lastName">Last Name:</label>
        <input type="text" id="lastName" name="lastName" required>

        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>

        <label for="phone">Phone Number:</label>
        <input type="tel" id="phone" name="phone" required>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>

        <label for="department">Department:</label>
        <input type="text" id="department" name="department" required>

        <button type="submit" name="submit">Register</button>
</form>
</div>

```

This HTML code snippet defines the structure of a user registration form within a web application interface. Enclosed within a element with the class container, the form incorporates various input fields for capturing user details, adhering to a clean and user-friendly design. This academic analysis dissects the HTML structure, the purpose of each form element, and the integration of attributes to enhance user experience and facilitate secure data submission.

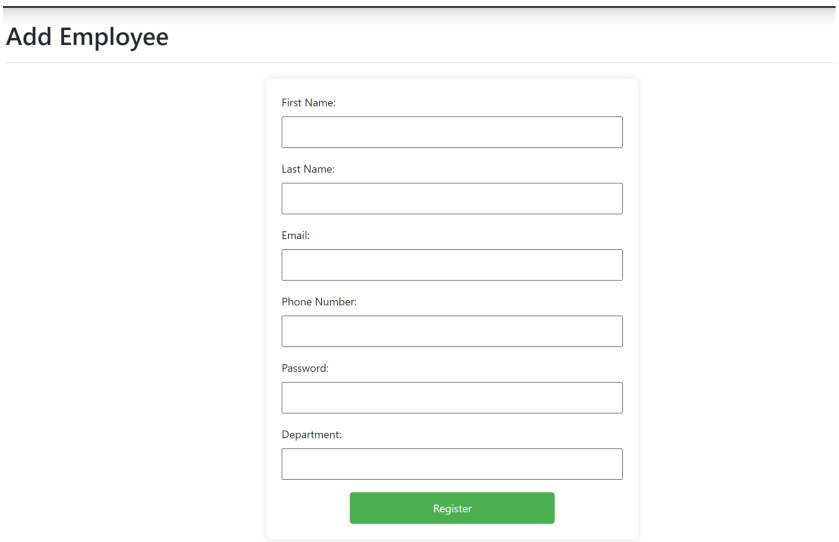
1. Form Container Structure: The form is enveloped within a element with the class container, emphasizing a structured layout for the user registration section within the web application. This design choice contributes to visual clarity and consistency.

2. Form Element: The HTML <form> element encompasses the entire user registration form, indicating the initiation of a data submission process. The method="post" attribute specifies that the form data will be sent to the server using the HTTP POST method.

3. Input Fields: The form incorporates several <label> and <input> pairs, each designed to capture specific user details. These input fields include:

First Name: An input field of type text (<input type="text">) with the id firstName and a required attribute.
Last Name: An input field of type text with the id lastName and a required attribute.
Email: An input field of type email (<input type="email">) with the id email and a required attribute.
Phone Number: An input field of type tel (<input type="tel">) with the id phone and a required attribute.
Password: An input field of type password (<input type="password">) with the id password and a required attribute.
Department: An input field of type text with the id department and a required attribute.
4. Submit Button: The form includes a <button> element with the attribute type="submit" and name="submit". This button triggers the submission of the form data to the server when clicked, initiating the user registration process.

Everything above comes together to build this well crafted and complete employee addition page, which brings immense value to companies who scale and need to add new employee accounts securely, while keeping their own records tightly under control with strict protocols. The presented solution here features procedures that top SaaS companies provide their customers with the feasibility and flexibility of adding or removing employees.



c. See all employee claims & edit page

```php
<?php

if(isset($_SESSION['email'])) {
    $email = $_SESSION['email'];
    $db = new myConnection();

    // Fetch data from the MySQL table
    $sql = "SELECT id, user_details.first_name as firstName, date, month, category, amount FROM chart_data LEFT JOIN user_details on user_details.email =
    chart_data.email";
    $result = $db->query($sql);
}

if ($result->num_rows > 0) {
    // Display table header
    echo "<table><tr><th>Employee</th><th>Date</th><th>Category</th><th>Amount</th><th class='edit-column'>Edit</th></tr>";

    // Output data of each row
    while ($row = $result->fetch_assoc()) {
        echo "<tr><td>" . $row["firstName"] . "</td><td>" . $row["date"] . "</td><td>" . $row["category"] . "</td><td> RM " . $row["amount"] . "</td>";

        echo "<td class='edit-column'><button class='edit-button' onclick=\"editRow(" . $row["id"] . ")\">Edit</button></td>
        </tr>";
    }

    // Close table
    echo "</table>";
} else {
    echo "0 results";
}

?>

<!-- Add JavaScript function for editing -->
<script>
    function editRow(id) {
        // Redirect to the edit page with the specific row ID
        window.location.href = "user.php?module=See%20All%20Claims&page=edit_page&id=" + id;
    }
</script>
```

This PHP code segment handles the retrieval and presentation of employee expense data within LightXpense web application. Utilizing the MySQL database, the code fetches relevant information and dynamically generates an HTML table to display employee-specific expense details.

1. Session Check and Database Connection:

The code initiates by checking the existence of an active user session (isset($_SESSION['email'])). If a session is present, the user's email is extracted, and a new database connection ($db = new myConnection()) is established using the custom myConnection class.

2. MySQL Query and Data Retrieval:

A SQL query is formulated to fetch relevant data from the MySQL table (chart_data). The query involves a LEFT JOIN operation with the user_details table to incorporate employee details. The result is stored in the $result variable through the $db->query($sql) method. The back end operations to enable this will be described in detail on the next section.

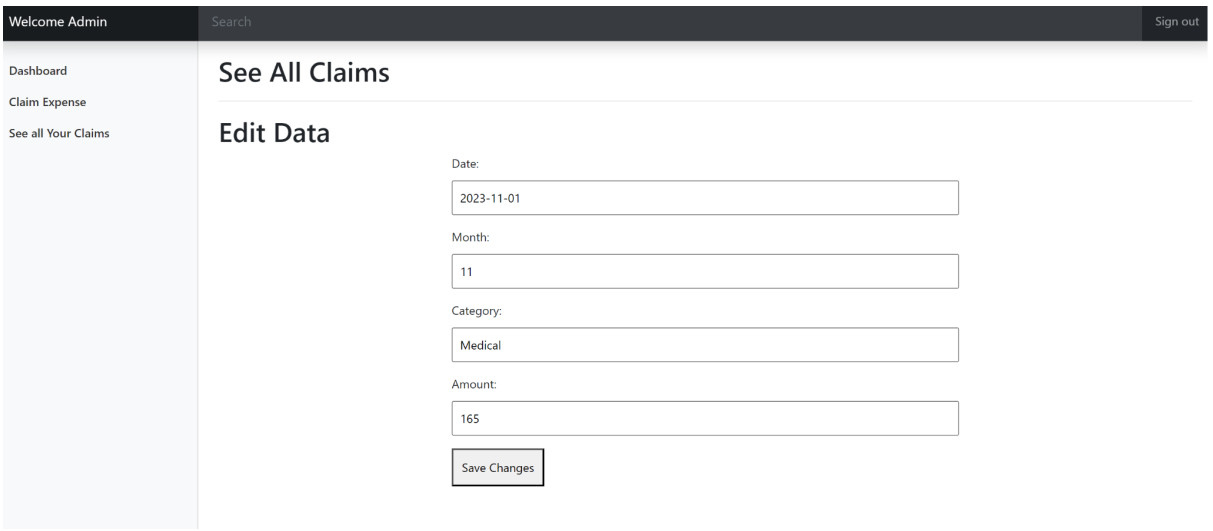
3. Table Display Logic:

If the query yields results ($result->num_rows > 0), the code proceeds to dynamically construct an HTML table to display the retrieved data. The table header includes columns such as "Employee," "Date," "Category," "Amount," and an "Edit" column. The latter serves as a placeholder for potential interactive editing functionality. The table is created using echo "<table><tr><th>Employee</th><th>Date</th><th>Category</th><th>Amount</th><th class='edit-column'>Edit</th></tr>" which sends out HTML code through a PHP echo command. Each table header is enclosed in <th> tag and the whole headers are enclosed in <tr>.

For each row in the query result, the code outputs corresponding data within table rows. Employee-specific details such as first name ($row["firstName"]), date ($row["date"]), category ($row["category"]), and amount ($row["amount"]) are presented. An "Edit" button (<button class='edit-button' onclick="editRow(" . $row["id"] . ")">Edit</button>) is also included in the last column for potential data editing. All these are done to ensure data input to the table is done dynamically based on the entries on the mySQL server.

4. Interactive Editing Feature:

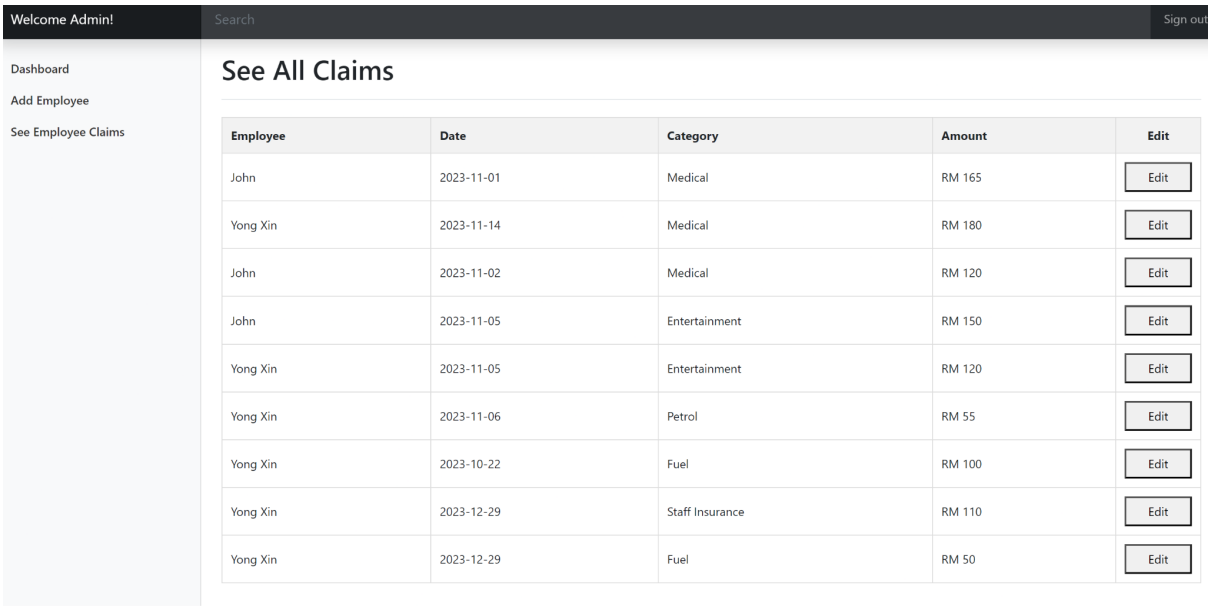
The "Edit" button is designed with an onclick attribute, calling a JavaScript function (editRow()) when clicked. This interactive feature hints at a potential mechanism for users to edit specific data entries within the displayed table. Upon clicking this, users are re-directed to an edit modal, as shown below.



5. No Results Message:

In the scenario where the query yields no results, the code echoes "0 results," providing a clear indication when no employee expense data is available for display.

From the codes above, the see all claims page has the following look:



3. Employee Account

The Employee Section of our website represents a sophisticated and tailored interface designed to optimize the management and classification of employee reimbursement

processes. This section encompasses three integral components: the Dashboard, the Claim Expense page, and the View All Claims page. The Dashboard is meticulously crafted to provide employees with a visually insightful presentation of key performance indicators (KPIs) directly relevant to their individual claims. This feature empowers users to comprehensively monitor and analyze their financial activities, fostering a data-driven approach to reimbursement management. The Claim Expense page is strategically designed to facilitate a seamless submission process, enabling employees to effortlessly upload images and details to initiate a claim expense. Furthermore, the View All Claims page serves as a centralized repository, offering a comprehensive overview of all submitted claims

```php
<?php

include("include/mysql.php");

session_start(); // Resume the session

// Check if the user is logged in
if (!isset($_SESSION['email'])) {
    header('location: login.php'); // Redirect to login page if not logged in
    exit();
} else {
  $f_name = $_SESSION['name'];
}


// $db = new myConnection;

?>

```

This PHP code functions to manage sessions and authentication employees on the LightXpense webpage. It begins by including an external file, 'mysql.php,' containing configurations for MySQL database interaction and various functions necessary for the employee section. The session_start(); command initializes or resumes a session, enabling the use of session variables to persist data across pages. The subsequent code checks whether the user is logged in by verifying the existence of the 'email' session variable. If the 'email' session variable is not set, indicating an unauthenticated user, the code redirects them to the login page using header('location: login.php'); and terminates script execution with exit();. Conversely, if the 'email' session variable is set, implying the user is authenticated, the code retrieves the user's first name from the 'name' session variable. This extracted information can be used to personalize the user experience, such as displaying the user's name on the page.

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>LightXpense</title>
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
    <script
        src="https://code.jquery.com/jquery-3.6.0.js"
        integrity="sha256-H+K7U5CnXl1h5ywQfKtSj8PCmoN9aaq30gDh27Xc0jk="
        crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://cdn.datatables.net/1.12.1/js/jquery.dataTables.min.js"></script>
    <script src="https://cdn.datatables.net/1.12.1/js/dataTables.bootstrap5.min.js"></script>
    <link rel="stylesheet" href="css/dashboard.css">
    <link rel="icon" type="image/png" href="images/logo2.png"/>


    <style>
      .bd-placeholder-img {
        font-size: 1.125rem;
        text-anchor: middle;
        -webkit-user-select: none;
        -moz-user-select: none;
        user-select: none;
      }

      @media (min-width: 768px) {
        .bd-placeholder-img-lg {
          font-size: 3.5rem;
        }
      }
    </style>


    <!-- Custom styles for this template -->
    <link href="dashboard.css" rel="stylesheet">
  </head>
```

The code snippet above initializes requirements necessary for the employee view of the webpage. It starts with the <head> tag, which has components as follows:

1. Meta, title and favicon: These are some of the base components that make the website user friendly through design on the tab bar of the browser, and some of the metadata associated with the webpage. For instance, <title> LightXpense </title> and <link rel="icon" type="image/png" href="images/logo2.png"/> together define how the webpage is shown in the user's end on the tab bar, as demonstrated below.

2. Links to external styling files and scripting files: The code also contains links to external styling sheets such as bootstrap and jquery, which facilitates the design of the webpage. It also includes self-coded files such as dashboard.css which designs components specific to the employee dashboard page.

```
60    <body>
61
62    <header class="navbar navbar-dark sticky-top bg-dark flex-md-nowrap p-0 shadow">
63    <a class="navbar-brand col-md-3 col-lg-2 me-0 px-3" href="#">Welcome <?php echo $f_name; ?></a>
64    <button class="navbar-toggler position-absolute d-md-none collapsed" type="button" data-bs-toggle="collapse"
      data-bs-target="#sidebarMenu" aria-controls="sidebarMenu" aria-expanded="false" aria-label="Toggle navigation">
65      <span class="navbar-toggler-icon"></span>
66    </button>
67    <input class="form-control form-control-dark w-100" type="text" placeholder="Search" aria-label="Search">
68    <div class="navbar-nav">
69      <div class="nav-item text-nowrap">
70        <a class="nav-link px-3" href="signout.php">Sign out</a>
71      </div>
72    </div>
73  </header>
74
```

This block of HTML code defines the structure of the LightXpense employee view header, providing a navigation bar with various elements. The header element has the classes "navbar," "navbar-dark," "sticky-top," "bg-dark," "flex-md-nowrap," "p-0," and "shadow," which collectively contribute to its appearance and behavior. Within the header, there is a hyperlink (<a>) with the class "navbar-brand" representing the website's brand or logo. It occupies a specific grid column width for medium (col-md-3) and large (col-lg-2) screens. The dynamic element <?php echo $f_name; ?> is included in the hyperlink text, personalizing the greeting by displaying the user's first name. The value is received through a SQL query so each employee will see their own personal name in their specific dashboard.

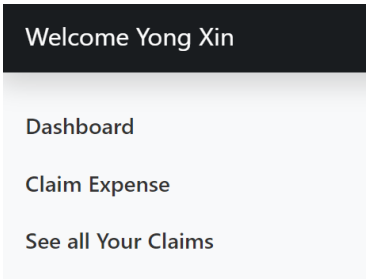
Additionally, there is a button (<button>) with the class "navbar-toggler" that, on smaller screens (d-md-none), triggers the collapse and expansion of a navigation sidebar. This functionality is achieved through Bootstrap's data attributes (data-bs-toggle and data-bs-target). The button features an icon represented by the element.

Next, an input field (<input>) with the classes "form-control" and "form-control-dark" is present for search functionality. It spans the entire width of its container (w-100). Users can enter search queries in this field, enhancing the website's usability. The header also includes a navigation section ( ) containing a link (<a>) with the class "nav-link" for signing out. This link directs users to the "signout.php" page when clicked. The entire navigation section is encapsulated within a div with the class "nav-item text-nowrap.". The navigation bar developed through this is shown below:

```
<div class="container-fluid">
  <div class="row">
    <nav id="sidebarMenu" class="col-md-3 col-lg-2 d-md-block bg-light sidebar collapse">
      <div class="position-sticky pt-3">
        <ul class="nav flex-column">
        <li class="nav-item">
            <a class="nav-link" aria-current="page" href="user.php?module=dashboard&page=home">
              <span data-feather="home"></span>
              Dashboard
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="user.php?module=Claim%20Expense&page=expense">
              <span data-feather="file"></span>
              Claim Expense
            </a>
          </li>
          <li class="nav-item">
            <a class="nav-link" href="user.php?module=See%20Your%20Claims&page=claims">
              <span data-feather="file"></span>
              See all Your Claims
            </a>
          </li>
        </ul>


      </div>
    </nav>
```

This HTML code defines a navigation sidebar for a web page, utilizing Bootstrap classes for styling and responsiveness. The nav element has the ID "sidebarMenu" and the classes "col-md-3," "col-lg-2," "d-md-block," "bg-light," "sidebar," and "collapse." These classes contribute to the layout, background color, and collapsibility of the sidebar. Within the sidebar, a div with the class "position-sticky" and a top padding (pt-3) is present. This ensures that the content within the sidebar remains sticky as the user scrolls. Inside this div, an unordered list (<ul>) with the class "nav flex-column" is used to structure the navigation links.

The list items (<li>) within the unordered list represent individual navigation items. Each item includes an anchor (<a>) element with the class "nav-link" and an href attribute pointing to different pages based on the user's interaction. The first item directs users to the "Dashboard" page, the second to the "Claim Expense" page, and the third to the "See all Your Claims" page.

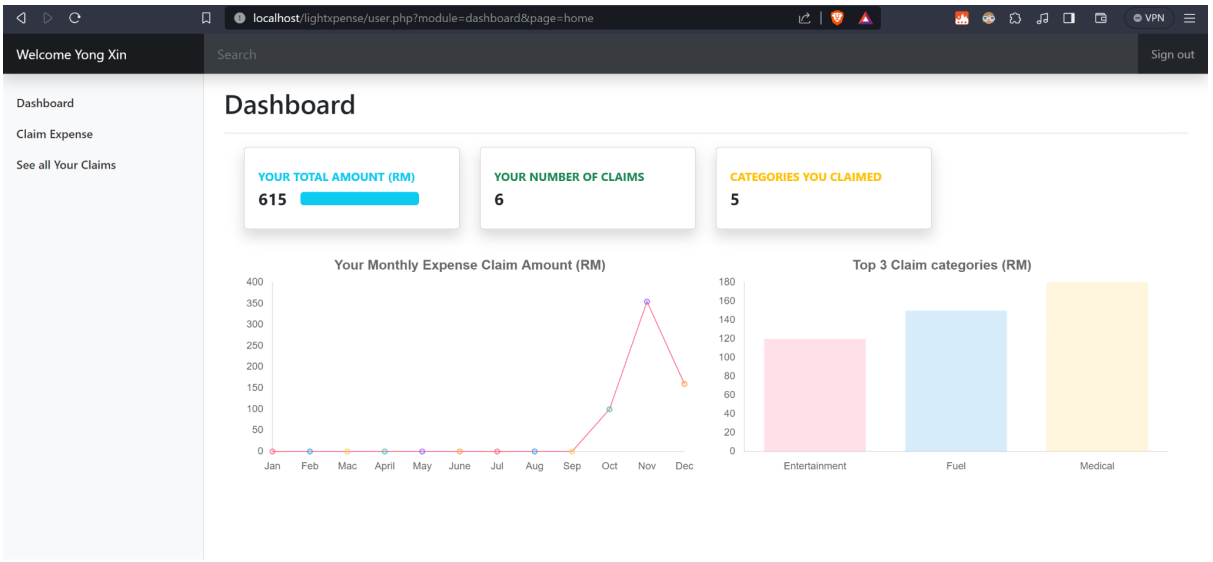


```
    <main class="col-md-9 ms-sm-auto col-lg-10 px-md-4">
      <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-items-center pt-3 pb-2 mb-3 border-bottom">
      <h1 class="h2"><?php echo ucfirst ($_GET['module']);?></h1>
      </div>

    <?php
        $module = $_GET['module'];
        $page = $_GET['page'];
        if(file_exists("module/".$module."/".$page.".php")){
            include("module/$module/$page.php");
        } else {
            echo "Error 404";
        }

    ?>


    </main>
  </div>
</div>

    <!-- Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p" crossorigin="anonymous"></script>

  </body>
</html>
```

The main element has the classes "col-md-9," "ms-sm-auto," "col-lg-10," and "px-md-4," which are Bootstrap classes determining the width and padding of the main content area. This section is expected to occupy a significant portion of the page's width. Within the main element, there is a div with the classes "d-flex," "justify-content-between," "flex-wrap," "flex-md-nowrap," "align-items-center," "pt-3," "pb-2," "mb-3," and "border-bottom." This div represents a header section with flexible and responsive layout properties. Inside this div, there is an h1 heading element with the class "h2." The text content of this heading is dynamically generated using PHP (<?php echo ucfirst ($_GET['module']);?>). It takes the 'module' parameter from the URL, capitalizes the first letter, and displays it as the heading.

This dynamic approach allows for modular content generation. This means that the main page of the employee screen depends upon the page that the employee selects from the side bar defined above. By default, this opens the dashboard, however, when an employee clicks on ‘Claim Expense’, the content enclosed in the <main> </main> tags are dynamically changed to reflect the employee’s request. The PHP code is used to include content based on the 'module' and 'page' parameters obtained from the URL. The script checks if a corresponding PHP file exists in the "module" directory. If the file exists, it includes the file; otherwise, it displays an "Error 404" message. This approach supports modular development, allowing different modules and pages to be dynamically loaded based on user interactions.

The code concludes with the inclusion of the Bootstrap Bundle with Popper script, facilitating Bootstrap functionalities such as popovers and tooltips. The integrity and crossorigin attributes ensure the secure loading of the script. With the code above, the following screen greets an employee when they first login. The main content displayed is dynamic and changes when the user clicks on any navigational element on the side bar.



a. Employee Dashboard page

```php
<?php


if(isset($_SESSION['email'])) {
    $email = $_SESSION['email'];
    $db = new myConnection();

    $totalAmount = $db->total_amount($email);
    $numberClaims = $db->number_claims($email);
    $categoryCounts = $db->count_distinct_categories($email);

} else {
    header('location:login.php');
}

?>

```

The dashboard begins with a PHP code segment designed for session-based user authentication and data retrieval from a database. Here's a breakdown of its functionality:

The code initiates by checking if the 'email' session variable is set using isset($_SESSION['email']). This serves as a security measure to ensure that the user is authenticated and an active session is in progress. If the 'email' session variable is set, indicating that the user is logged in, the code proceeds to retrieve the user's email address from the session and initiates a connection to the database using a custom class, named myConnection. This class encapsulates various database operations. Subsequently, three distinct functions are called on the database object ($db):

1. $totalAmount = $db->total_amount($email);: This function, named total_amount, retrieves the total amount of expenses or claims associated with the user's email.
2. $numberClaims = $db->number_claims($email);: This function, named number_claims, fetches the total number of claims submitted by the user.
3. $categoryCounts = $db->count_distinct_categories($email);: This function, named count_distinct_categories, retrieves counts of distinct categories associated with the user's claims.

These variables ($totalAmount, $numberClaims, and $categoryCounts) hold valuable data related to the user's expense claims, providing insights into their financial activities. These are parsed and displayed in an intuitive and user-friendly way in the dashboard, which will be explained further from codes below.

In the event that the 'email' session variable is not set, meaning the user is not authenticated, the code redirects the user to the login page using header('location:login.php'). This prevents unauthorized access to the data retrieval functionalities and ensures that only authenticated users can access and interact with their claim-related information.

```
<div class="container-fluid">
    <div class="row">
            <div class="col-md-6 col-xl-3 mb-4">
            <div class="card shadow border-start-info py-2">
                <div class="card-body">
                    <div class="row align-items-center no-gutters">
                        <div class="col me-2">
                            <div class="text-uppercase text-info fw-bold text-xs mb-1"><span>Your Total Amount (RM)</span></div>
                            <div class="row g-0 align-items-center">
                                <div class="col-auto">
                                    <div class="text-dark fw-bold h5 mb-0 me-3"><span><?php echo $totalAmount; ?></span></div>
                                </div>
                                <div class="col">
                                    <div class="progress progress-sm">
                                        <div class="progress-bar bg-info" aria-valuenow="50" aria-valuemin="0" aria-valuemax="100" style="width: 100%;
                                        "><span class="visually-hidden">50%</span></div>
                                    </div>
                                </div>
                            </div>
                        </div>
                        <div class="col-auto"><i class="fas fa-clipboard-list fa-2x text-gray-300"></i></div>
                    </div>
                </div>
            </div>
        </div>
```

After that, there are PHP and HTML codes that encapsulates the construction of a dynamic card visualization within an employee dashboard, designed to represent the total financial amount for expense claims. The visualization employs Bootstrap grid system classes to ensure responsiveness and consistent layout. Through the strategic use of Bootstrap and Font Awesome classes, the card achieves a visually appealing and informative presentation of real-time data. Some of the major components for it are as follows:

1. Card Structure: The card is enveloped within a Bootstrap grid system, specifically a element, with a subsequent defining its responsiveness and layout characteristics. This structural foundation sets the stage for a visually cohesive presentation.

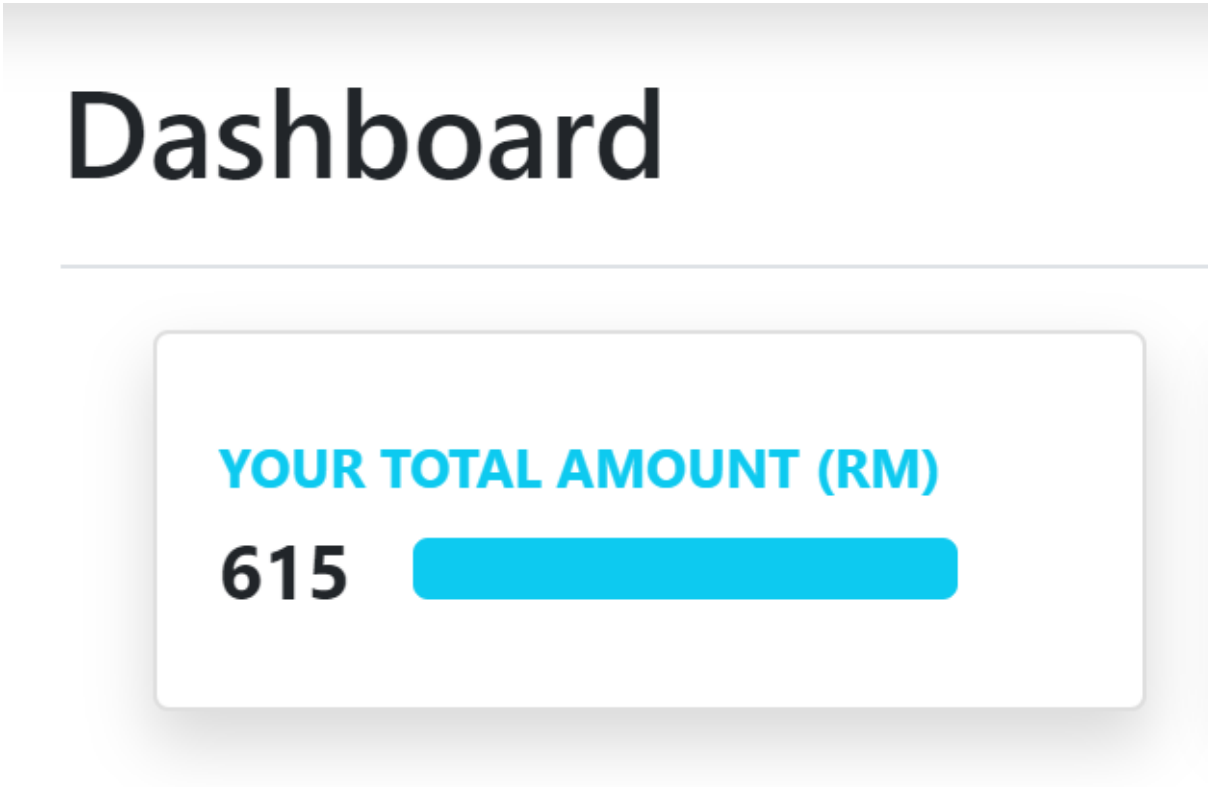
2. Card Styling: Bootstrap classes such as card, shadow, border-start-info, and py-2 contribute to the visual aesthetics of the card, incorporating shadows, border coloring, and padding. These styling elements collectively enhance the user experience and maintain a harmonious design.

3. Header and Body Content: The card's header intelligently organizes the displayed data metric, "Total Amount (RM)," employing Bootstrap classes like text-uppercase, text-info, fw-bold, and text-xs for clear emphasis. Meanwhile, the body dynamically echoes the total amount ($totalAmount) obtained from the database within a element, ensuring the real-time representation of financial data.

4. Progress Bar: A Bootstrap-styled progress bar, located within the card's body, visually represents the data metric. Its width is dynamically adjusted based on the data value (width: 100%), offering a proportional visualization. Bootstrap classes such as progress, progress-sm, and bg-info enhance the aesthetic appeal and clarity of the progress bar.

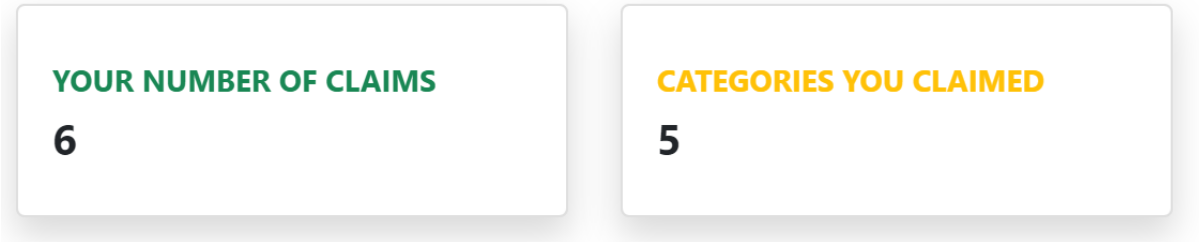
5. Icon Representation: Complementing the card, an icon represented by provides a visual cue related to the nature of the presented financial data.

The card development is done in a similar manner to the admin dashboard. However, the key difference is that the value of total amount is queried specifically to show only those that are from the employee that is logged in. With this, the card looks as follows:



```
        <div class="col-md-6 col-xl-3 mb-4">
            <div class="card shadow border-start-success py-2">
                <div class="card-body">
                    <div class="row align-items-center no-gutters">
                        <div class="col me-2">
                            <div class="text-uppercase text-success fw-bold text-xs mb-1"><span>Your Number of Claims</span></div>
                            <div class="text-dark fw-bold h5 mb-0"><span><?php echo $numberClaims; ?></span></div>
                        </div>
                        <div class="col-auto"><i class="fas fa-dollar-sign fa-2x text-gray-300"></i></div>
                    </div>
                </div>
            </div>
        </div>
        <div class="col-md-6 col-xl-3 mb-4">
            <div class="card shadow border-start-warning py-2">
                <div class="card-body">
                    <div class="row align-items-center no-gutters">
                        <div class="col me-2">
                            <div class="text-uppercase text-warning fw-bold text-xs mb-1"><span>Categories you Claimed</span></div>
                            <div class="text-dark fw-bold h5 mb-0"><span><?php echo $categoryCounts; ?></span></div>
                        </div>
                        <div class="col-auto"><i class="fas fa-comments fa-2x text-gray-300"></i></div>
                    </div>
                </div>
            </div>
        </div>
    </div>
```

These codes also work in a similar way as described above to make two additional cards to showcase important KPIs in the dashboard. Only the progress bar has been removed for these two cards to show an effective visualization without overcrowding.




```
<?php
    // the easiest way to do it but this cost a lot of resources. can change to looping and array.
    $data_jan = 0;
    $data_feb = 1;
    $data_mac = 3;
    $data_apr = 4;
    $data_may = 5;
    $data_jun = 6;
    $data_jul = 7;
    $data_aug = 8;
    $data_sep = 9;
    $data_oct = 10;
    $data_nov = 11;
    $data_dec = 12;

    $data_by_month = $db->data_chart_by_month($email);
    $data_month = array();
    foreach($data_by_month as $key => $dtbm){
        $data_month[$key] = $dtbm;
    }

    $statuses = $db->category();
    $st = array();
    foreach($statuses as $s => $stt){
        $st[$s] = $stt;
    }

```

```php
    $status_inv = $db->data_chart_by_category($email);
    $data_status = array();
    foreach($status_inv as $bil => $dcbs){
        $data_status[$bil] = $dcbs;
    }

?>
```

This PHP code snippet employs efficient data processing techniques to facilitate the retrieval and organization of dynamic data for chart visualizations within the employee dashboard. Focused on monthly and category-based metrics, the code transforms individual variables into a more scalable array-based structure.

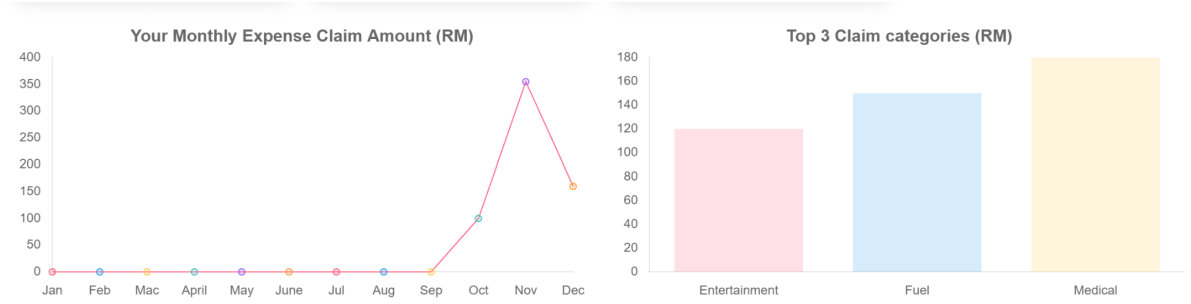
1. Monthly Data Mapping: The initial section of the code assigns individual variables to represent each month, providing a rudimentary but resource-intensive approach. However, recognizing the need for a more scalable and efficient solution, the subsequent code introduces an array-based approach to map data by month. A loop iterates through the retrieved data, creating an associative array ($data_month) that holds monthly data values, dynamically adapting to changes in data structure or volume.

2. Category Data Mapping: Similarly, the code fetches and processes category-related data through a loop, resulting in an associative array ($st) representing various statuses. The status categories are obtained through a call to $db->category().

3. Monthly Status Data Mapping: The code further retrieves monthly data categorized by status through the $db->data_chart_by_category() method. This information is processed in a loop, resulting in an associative array ($data_status). This array efficiently organizes data, paving the way for streamlined chart visualizations.

4. Database Queries: Database interactions are prominent in the code, demonstrated by calls to methods such as $db->data_chart_by_month(), $db->category(), and $db->data_chart_by_category(). These methods fetch essential data from the database, allowing for the real-time representation of metrics in the administrative dashboard.

With the explanation above, we are able to achieve the following graphs on the dashboard



b. Claim Expense page

Claim module

```php
<?php

if (isset($_POST['submit'])) {
  $uploadDir = 'uploads/'; // Directory to store uploaded images
  $uploadName = basename($_FILES['image']['name']);
  $uploadFile = $uploadDir . basename($_FILES['image']['name']);
  $file_path = urlencode($uploadFile);
  // Check if the file is an image
  $imageFileType = strtolower(pathinfo($uploadFile, PATHINFO_EXTENSION));
  $allowedExtensions = ['jpg', 'jpeg', 'png', 'gif'];

```

This PHP code segment handles image file uploads that affect the front-end, upon the submission of a form. It begins by checking if the form has been submitted (isset($_POST['submit'])). If the condition is met, the code proceeds with processing the file upload. Key operations include defining the upload directory ($uploadDir) and constructing the complete file path ($uploadFile). The file path is then URL-encoded ($file_path) for potential use in URLs. The code checks the file type by extracting the file extension ($imageFileType) and converts it to lowercase for consistent comparison. An array of allowed image extensions ($allowedExtensions) is defined. These operations lay the groundwork for subsequent steps, such as verifying if the uploaded file is of an acceptable image type.

Further back-end working to extract text from the image and classify it will be explained in the back-end section below.

```
<!--Start Status Bar-->
<div class="status-bar">
        <div class="dot-green">
          <div class="dot-label">Claim</div>
        </div>
        <div class="line-grey"></div>
        <div class="dot-grey">
        <div class="dot-label">Confirmation</div>
        </div>
        <div class="line-grey"></div>
        <div class="dot-grey">
          <div class="dot-label">Submit</div>
        </div>
      </div>
      <!--End Status Bar-->
```

This HTML code defines a visual status bar intended to depict stages in a process. Enclosed within HTML comments, the "Status Bar" section begins with a div classed as "status-bar," serving as the container for the entire status bar. Within this container, there are distinct elements for each stage:

Firstly, a green dot labeled "Claim" is represented by a div with the class "dot-green." A nested div with the class "dot-label" provides the text label for this initial stage. A grey line follows, contributing to the visual separation of stages within the status bar. Subsequently, a grey dot labeled "Confirmation" is represented by a div with the class "dot-grey." Similar to the green dot, it includes a nested div with the class "dot-label" containing the text "Confirmation" as the label for this stage. Another grey line follows, emphasizing the distinction between stages. Lastly, another grey dot labeled "Submit" is represented by a div with the class "dot-grey." A nested div with the class "dot-label" provides the text label "Submit" for this final stage. The color and highlighting of the stage in the status bar is done dynamically based on which stage the user is in, and changes color to show the correct stage an employee is in.

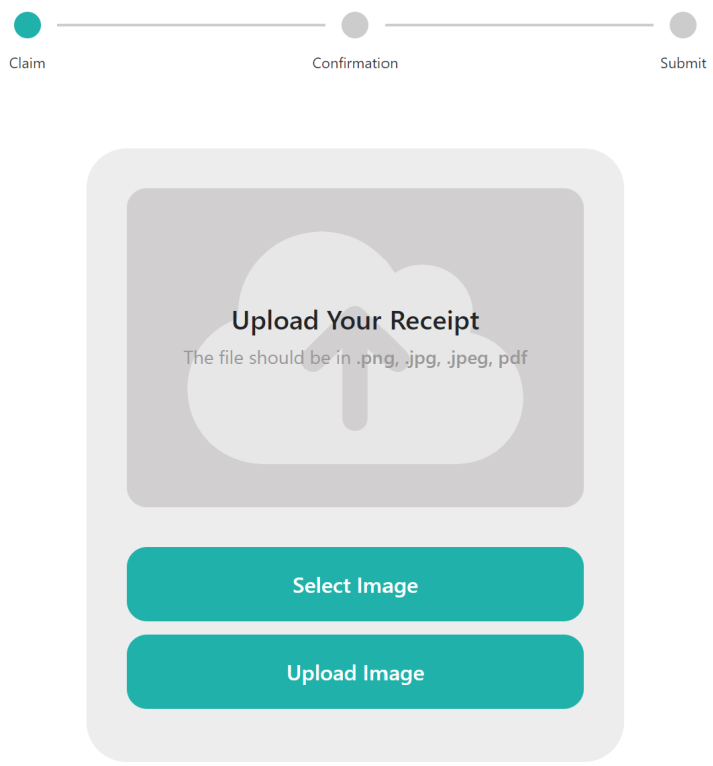
```
<!-- Start Claim Expenses -->
<form method="post" enctype="multipart/form-data">
  <div class="claim">
    <div class="container">
      <input type="file" name="image" id="file" accept="image/*" hidden>
      <div class="img-area">
        <h3>Upload Your Receipt</h3>
        <p>The file should be in <span>.png, .jpg, .jpeg, pdf</span></p>
        <i class="fas fa-solid fa-cloud-arrow-up"></i>
      </div>
      <div class="claim-buttons">
        <label for="file" class="select-image">Select Image</label>
        <input type="submit" value="Upload Image" name="submit" class="select-image">
      </div>
    </div>
  </div>
</form>
<!-- End Claim Expenses -->
```

This HTML code delineates a form section for claiming expenses, providing users with the ability to upload receipts. Enclosed within HTML comments, the "Claim Expenses" section begins with a form element set to use the POST method and handle file uploads (enctype="multipart/form-data"). Within the form, a div with the class "claim" serves as the container for the entire claim expenses section. Nested within this container div is another div with the class "container," encapsulating the elements for the file upload interface.

An input element of type "file" with the name "image" is included. It is designed to accept image files (accept="image/*"). This input field is later triggered by a label and serves as the mechanism for users to select and upload their receipt images. A div with the class "img-area" encompasses a heading, paragraph, and an icon. The heading prompts users to upload their receipt, while the paragraph provides guidance on acceptable file formats (".png, .jpg, .jpeg, pdf"). An icon (solid cloud with an upward arrow) enhances the visual representation of the upload process.

Finally, a div with the class "claim-buttons" contains two elements: a label for the file input (<label for="file" class="select-image">Select Image</label>) and a submit button (<input type="submit" value="Upload Image" name="submit" class="select-image">). These elements allow users to interact with the file upload functionality.

These codes together build the image input for employees to submit an image of their receipt to submit an expense claim as shown below.



Confirm module

```php
<?php

include 'db_connection.php';

    if(isset($_SESSION['email'])){
      $email = $_SESSION['email'];
    }

    if (isset($_GET['file_name'])) {
        // Decode variables from URL
        $image_path = urldecode($_GET['file_name']);
        $category = urldecode($_GET['category']);
        $amount = urldecode($_GET['totalamount']);
        $claim_date = date('Y-m-d');
    }

```

This PHP code includes a database connection file and performs operations based on session and URL parameters. The code does pre-processing needed to get necessary data to be displayed in the front-end.

It begins with an inclusion of the 'db_connection.php' file, establishing a connection to the database. Subsequently, the code checks if the 'email' session variable is set (isset($_SESSION['email'])). If set, it retrieves the user's email address from the session and assigns it to the variable $email. Another conditional statement checks if specific parameters are set in the URL (isset($_GET['file_name'])). If true, it proceeds to decode URL parameters related to a file, including 'file_name', 'category', 'totalamount', and sets the variable $claim_date to the current date.

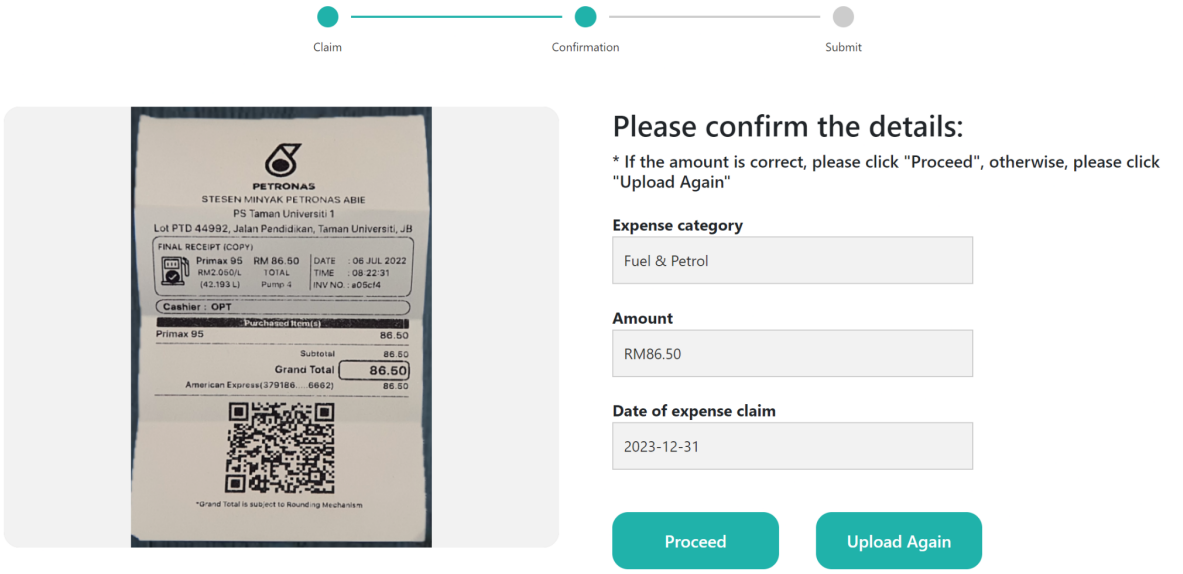
```
<!-- Start Confirmation-->
  <div class="confirmation">
    <div class="image-container">
          <div class="img-area">
          <?php
              // Check if the file exists before attempting to display it
              if (file_exists($image_path)) {
                  echo "<img class='image' src='$image_path' alt='Image'>";
              } else {
                  echo "Image not found.";
              }
          ?>
          </div>
    </div>
    <form class="details-container" method = "POST">
      <div class="details">
        <h3>Please confirm the details:</h3>
        <h6>* If the amount is correct, please click "Proceed", otherwise, please click "Upload Again"</h6>
        <div class="details-row">
          <label class="details-label">Expense category</label>
          <input class="details-input" type="text" readonly value="<?php echo htmlspecialchars($category); ?>">
        </div>
        <div class="details-row">
          <label class="details-label">Amount</label>
          <input class="details-input" type="text" readonly value="RM<?php echo htmlspecialchars($amount); ?>">
        </div>
        <div class="details-row">
          <label class="details-label">Date of expense claim</label>
          <input class="details-input" type="text" readonly value="<?php echo htmlspecialchars($claim_date); ?>">
        </div>
      </div>
      <div class="buttons">
        <button class="button-upload" type="submit" name="submit">Proceed</button>
        <a href="user.php?module=Claim%20Expense&page=expense" class="button-upload">Upload Again</a>
      </div>
      </div>
    </form>
  </div>
</div>
```

With the dynamic data retrieved and set from the PHP code above, this code snippet continues with designing the main content of the confirmation page. Within the HTML code, the "Confirmation" section begins with a div having the class "confirmation."

Inside this div, there is a div with the class "image-container" housing another div with the class "img-area." The embedded PHP code checks if the specified file path ($image_path) exists and, if true, displays an image using an <img> tag. If the file doesn't exist, it outputs an "Image not found" message.

Following the image section, there is a form with the class "details-container" set to use the POST method. This form includes a div with the class "details" containing various details for user confirmation. Within the "details" div, there are three detail rows, each with a label and an input field. The label and input fields display information related to the expense category, amount (formatted as RM), and the date of the expense claim. These fields are set to readonly to prevent user input. Below the details, there is a div with the class "buttons" containing two buttons. The first button, with the class "button-upload," serves as a form submission button labeled "Proceed." The second button, styled as a link (<a>), redirects the user to a specified URL ("user.php?module=Claim%20Expense&page=expense") labeled "Upload Again."

With these codes, the confirmation screen is developed as shown below. The status bar also changes dynamically to show to the employee that they are in the confirmation stage.



Submit module

```php
<?php

include 'db_connection.php';

if(isset($_SESSION['email'])){
    $email = $_SESSION['email'];
  }

if (isset($_GET['status'])) {
    // Decode variables from URL
    $message = urldecode($_GET['status']);
    $category = urldecode($_GET['label']);
    $amount = urldecode($_GET['total']);
    $claimed_date = urldecode($_GET['date']);
}

?>

<style>
```

This PHP code, similar to the code for confirmation page, includes a database connection file and performs operations based on session and URL parameters. The code does pre-processing needed to get necessary data to be displayed in the front-end.

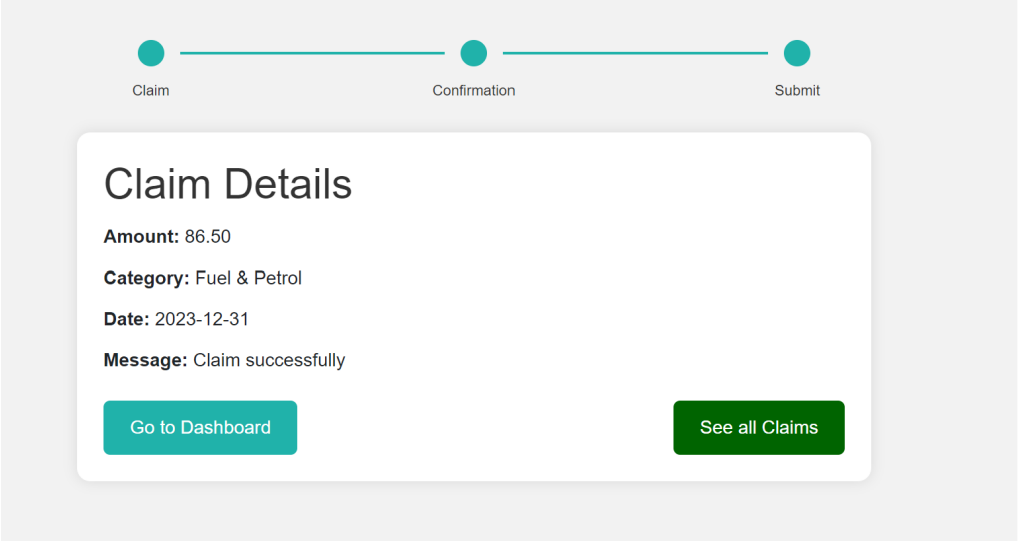
```
<div class="claim-details-container">
    <div class="claim-details">
      <h2>Claim Details</h2>
      <p><strong>Amount:</strong> <?php echo $amount; ?></p>
      <p><strong>Category:</strong> <?php echo $category; ?></p>
      <p><strong>Date:</strong> <?php echo $claimed_date; ?></p>
      <p><strong>Message:</strong> <?php echo $message; ?></p>
    </div>

    <div class="buttons-container">
      <a href="user.php?module=dashboard&page=home" class="dashboard-button">Go to Dashboard</a>
      <a href="user.php?module=See%20Your%20Claims&page=claims" class="all-claims-button">See all Claims</a>
    </div>
  </div>
```

The HTML code for the submit screen represents a section for displaying claim details in a web application, for the employee to view details for a final time and confirm them.

The structure includes two main div containers: "claim-details-container" and "claim-details." Within the "claim-details-container," there is a "claim-details" div that encompasses information such as the claimed amount, category, date, and a message. The displayed values are dynamically populated using PHP echo statements, retrieving information from the corresponding PHP variables.

Following the claim details, there is another div labeled "buttons-container" containing two buttons styled as links. The first button, with the class "dashboard-button," redirects the user to the dashboard page when clicked. The second button, with the class "all-claims-button," leads the user to a page displaying all their claims.

With these codes, the submission confirmation screen is displayed as shown below.



c. See all personal claims

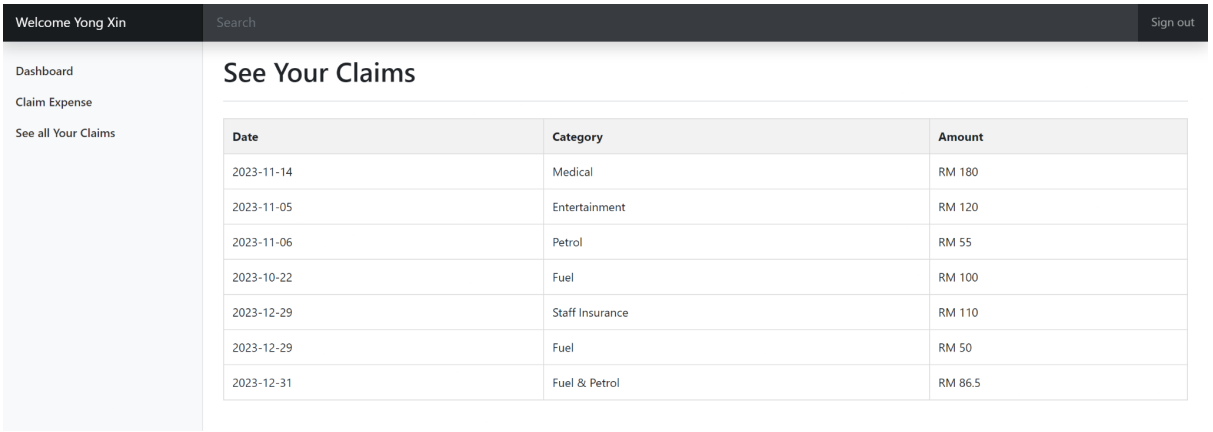
```php
<?php

if (isset($_SESSION['email'])) {
    $email = $_SESSION['email'];
    $db = new myConnection();

    // Fetch data from the MySQL table
    $sql = "SELECT id, date, month, email, category, amount FROM chart_data where email = '$email'";
    $result = $db->query($sql);
}

if ($result->num_rows > 0) {
    // Display table header
    echo "<table><tr><th>Date</th><th>Category</th><th>Amount</th></tr>";

    // Output data of each row
    while ($row = $result->fetch_assoc()) {
        echo "<tr><td>" . $row["date"] . "</td><td>" . $row["category"] . "</td><td> RM " . $row["amount"] . "</td></tr>";
    }

    // Close table
    echo "</table>";
} else {
    echo "0 results";
}

?>
```

This PHP and HTML code segment interacts with a MySQL database to retrieve and display data in a tabular format. The break down of its functionality is as follows:

The code begins by checking if the 'email' session variable is set (isset($_SESSION['email'])). If true, it retrieves the user's email from the session and initializes a new database connection using the 'myConnection' class. Within the conditional block, a SQL query is constructed to select specific columns from the 'chart_data' table where the email matches the user's email. The query is executed, and the result is stored in the variable $result.

The code then checks if there are rows in the result set ($result->num_rows > 0). If true, it enters a block to display the data in a table format. Inside the block, an HTML table is echoed, starting with a table header row containing column names: Date, Category, and Amount. The code then iterates through each row of the result set using a while loop and outputs the data within table rows. The amount is formatted with the currency symbol "RM."

If there are no results in the result set, it shows "0 results" to the employee.



Back-end codes

The backend of LightXpense web application has been pivotal in facilitating personalization, ensuring that users can interact with the system in a manner tailored to their unique needs and preferences. In the context of LightXpense, the backend is instrumental in establishing seamless communication between the web application and the MySQL database, enabling the presentation of individualized details to users. This integration is accomplished through the utilization of PHP in conjunction with SQL, and further augmented by the incorporation of Python through a shell_exec() command.

1. PHP with SQL Integration:

The synergy between PHP and SQL forms the backbone of backend functionality, allowing for the seamless exchange of data between the LightXpense web application and the MySQL database. This integration is evident across key components of the system:

Login Page

```php
<?php

include 'db_connection.php';

session_start();

if(isset($_POST['submit'])){

    $email = mysqli_real_escape_string($conn, $_POST['email']);
    $pass = mysqli_real_escape_string($conn, ($_POST['pass']));

    $select_users = mysqli_query($conn, "SELECT * FROM `user_details` WHERE email = '$email' AND password = '$pass'") or die
    ('query failed');

    if(mysqli_num_rows($select_users) > 0){

        $row = mysqli_fetch_assoc($select_users);

        $_SESSION['name'] = $row['first_name'];
        $_SESSION['email'] = $row['email'];
        $_SESSION['department'] = $row['department'];

        if($row['department'] == 'ADMIN'){

            header('location:admin.php?module=dashboard&page=admin');

        }else{

            header('location:user.php?module=dashboard&page=home');

        }

    } else{
        $errors[] = 'Incorrect email or password';
    }

}

?>
```

PHP integrated with SQL empowers the login page, enabling users to authenticate their identity by submitting their email and password. The backend communication sends the email and hashed password to the database, retrieving user details if the credentials match. Below are some of the key features of this back-end integration.

1. Database Connection:

   The code includes an external file named 'db_connection.php' using the include statement. This file likely contains code to establish a connection to the database. The mysqli_real_escape_string function is used to escape special characters in the user inputs ($_POST['email'] and $_POST['pass']). This is a security measure to prevent SQL injection.

2. Session Initialization:

   The session_start() function initializes a session or resumes the current session. Sessions are used to store and retrieve values across multiple pages for a specific user.

3. Form Submission Check:

The code checks if the form has been submitted by checking the existence of the 'submit' key in the $_POST superglobal using isset($_POST['submit']).

4. User Authentication:

   If the form has been submitted, the user-entered email and password are retrieved using $_POST['email'] and $_POST['pass']. A SQL query is executed to select all columns from the 'user_details' table where the email and password match the provided credentials. The mysqli_query function is used for this purpose. The mysqli_num_rows function checks if there is at least one row in the result set, indicating a successful match of email and password.

5. Session Variables:

   If a match is found, the code fetches the details of the user using mysqli_fetch_assoc and assigns them to session variables. The session variables include 'name,' 'email,' and 'department' retrieved from the database.

6. Redirection Based on User Type:

   The code then checks the user's department. If the department is 'ADMIN,' the user is redirected to the admin dashboard using header('location:admin.php?module=dashboard&page=admin'). If the department is not 'ADMIN,' which indicates that the user is a normal employee, the user is redirected to the user dashboard using header('location:user.php?module=dashboard&page=home').

7. Error Handling:

   If no matching user is found in the database, an error message is added to an array ($errors[]) stating "Incorrect email or password." This array can be used to display error messages to the user.

<u>i. Admin Dashboard page</u>

For admin accounts, PHP communicates with the database, fetching all expense claims associated with all of the employees. The returned data is parsed and dynamically showcased in intuitive charts within the dashboard, as showcased in the front-end codes above.

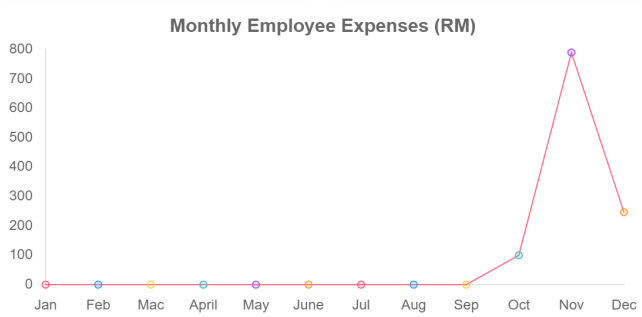
```php
function data_chart_by_month_admin(){
    $datamon = [];
    $month = [1,2,3,4,5,6,7,8,9,10,11,12];

    foreach($month as $mon){
        $data = 0;  // Reset $data for each month
        $sqldata = "SELECT date, month, category, amount FROM chart_data WHERE month='$mon'";
        $querydata = $this->query($sqldata);

        while($getdata = $querydata->fetch_array()){
            $data += $getdata['amount'];
        }

        $datamon[$mon] = $data;
    }

    return $datamon;
}
```

This PHP function, named data_chart_by_month_admin, is designed to retrieve and process data for a chart, specifically targeting an administrative context. The function initializes an empty array $datamon to store data for each month. Additionally, an array named $month is defined, containing numerical representations of the months from 1 to 12.

The function then iterates through each month using a foreach loop. For each iteration, it resets the variable $data to 0. This variable will accumulate the total amount for the specific month. Inside the loop, a SQL query is constructed to select data from the 'chart_data' table where the 'month' column matches the current month in the loop ($mon). The query is executed using the query method, belonging to a class that encapsulates database operations.

A while loop is used to iterate through the result set obtained from the query. For each row, the 'amount' column value is added to the $data variable. The total amount for the current month is then stored in the $datamon array with the month as the key. This process repeats for all months in the loop. Finally, the function returns the populated $datamon array containing the total amounts for each month.

This is used to send data to the admin dashboard for this chart:

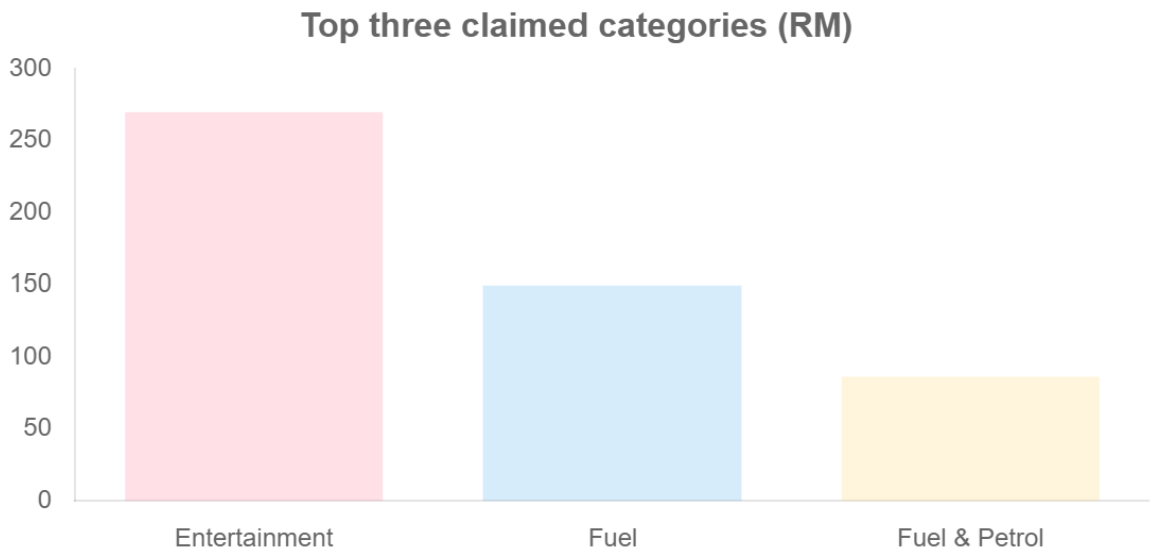


```php
function data_chart_by_category_admin(){
    $datamon = [];
    $category = $this->category();

    foreach ($category as $sta) {
        $data = 0;  // Reset $data for each category
        $sqldata = "SELECT amount FROM chart_data WHERE category='$sta'";
        $querydata = $this->query($sqldata);

        while ($getdata = $querydata->fetch_array()) {
            $data += $getdata['amount'];
        }

        $datamon[$sta] = $data;
    }

    return $datamon;
}
```

This PHP function, named data_chart_by_category_admin, is designed to retrieve and process data for a chart based on different expense categories across all users in an administrative context. The function initializes an empty array $datamon to store data for each category. It also retrieves an array of expense categories using the category method, which is assumed to be part of the same class.

The function then iterates through each category using a foreach loop. For each iteration, it resets the variable $data to 0. This variable will accumulate the total amount for the specific category. Inside the loop, a SQL query is constructed to select data from the 'chart_data' table where the 'category' column matches the current category in the loop ($sta). The query is executed using the query method. A while loop is used to iterate through the result set obtained from the query. For each row, the 'amount' column value is added to the $data variable. The total amount for the current category is then stored in the $datamon array with the category as the key. This process repeats for all categories in the loop.

Finally, the function returns the populated $datamon array containing the total amounts for each expense category across all users. The return of this function in back-end, is integrated to show two charts in the admin dashboard as follows:

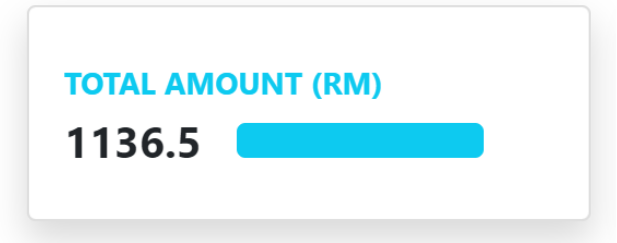



```php
function total_amount_admin(){
    $total = 0;
    $sqldata = "SELECT amount FROM chart_data";
    $querydata = $this->query($sqldata);

    while($getdata = $querydata->fetch_array()){
        $total += $getdata['amount'];
    }

    return $total;
}
```

This PHP function, named total_amount_admin, is designed to calculate and retrieve the total amount of expenses across all users in an administrative context. The function initializes a variable $total to 0. This variable will accumulate the total amount of expenses.

Inside the function, a SQL query is constructed to select the 'amount' column from the 'chart_data' table. The query is executed using the query method. A while loop is used to iterate through the result set obtained from the query. For each row, the 'amount' column value is added to the $total variable.

After processing all rows, the function returns the final total amount, which is shown in the front-end in the admin dashboard:

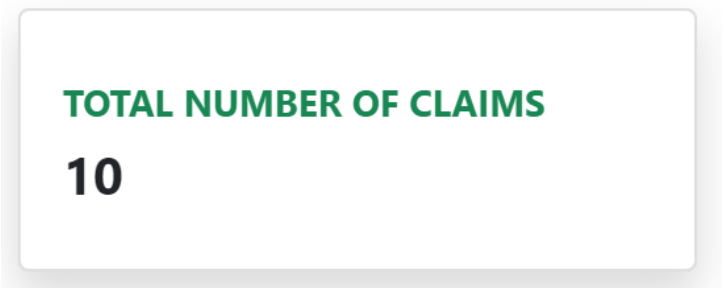



```php
function number_claims_admin(){
    $totalClaims = 0;
    $sqldata = "SELECT COUNT(id) AS claim_count FROM chart_data";
    $querydata = $this->query($sqldata);

    if ($querydata) {
        $result = $querydata->fetch_assoc();
        $totalClaims = $result['claim_count'];
    }

```

This PHP function, named number_claims_admin, is designed to calculate and retrieve the total number of expense claims across all users in an administrative context. The function initializes a variable $totalClaims to 0. This variable will store the total number of expense claims.

Inside the function, a SQL query is constructed to count the number of rows (claims) in the 'chart_data' table using the COUNT aggregate function. The query is executed using the query method. The function then checks if the query was successful (if ($querydata)). If true, it fetches the result as an associative array, extracting the count of claims and assigning it to the $totalClaims variable.

Finally, the function returns the calculated total number of expense claims. This is also displayed in the front-end of the admin dashboard as shown below:

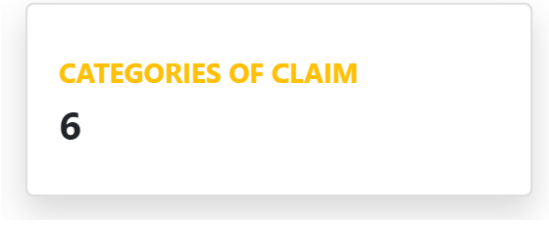


```php
    function count_distinct_categories_admin(){
        $sqldata = "SELECT COUNT(DISTINCT category) AS category_count FROM chart_data";
        $querydata = $this->query($sqldata);

        $categoryCount = 0;

        if ($querydata) {
            $result = $querydata->fetch_assoc();
            $categoryCount = $result['category_count'];
        }

        return $categoryCount;
    }
}
```

This PHP function, named count_distinct_categories_admin, is designed to calculate and retrieve the total number of distinct expense categories across all users in an administrative context. The function initializes a variable $categoryCount to 0. This variable will store the total number of distinct expense categories.

Inside the function, a SQL query is constructed to count the number of distinct categories in the 'chart_data' table using the COUNT aggregate function along with the DISTINCT keyword. The query is executed using the query method. The function then checks if the query was successful (if ($querydata)). If true, it fetches the result as an associative array, extracting the count of distinct categories and assigning it to the $categoryCount variable.

Finally, the function returns the calculated total number of distinct expense categories. This value is then integrated in the front-end as shown below:



ii. Add Employee page: Register Form

Admins leverage backend integration to add or remove employee accounts through SQL queries based on email addresses. The inserted data instantly becomes accessible for login purposes, highlighting the efficiency of the backend integration.

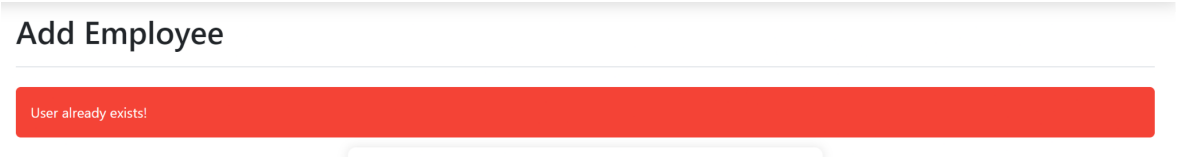
```php
if(isset($_POST['submit'])){
    $firstName = mysqli_real_escape_string($conn, $_POST['firstName']);
    $lastName = mysqli_real_escape_string($conn, $_POST['lastName']);
    $email = mysqli_real_escape_string($conn, $_POST['email']);
    $phone = mysqli_real_escape_string($conn, ($_POST['phone']));
    $password = mysqli_real_escape_string($conn, ($_POST['password']));
    $department = mysqli_real_escape_string($conn, ($_POST['department']));

    $select_users = mysqli_query($conn, "SELECT * FROM `user_details` WHERE email = '$email' AND password = '$password'") or die('query failed');

    if(mysqli_num_rows($select_users) > 0){
        $error_message = 'User already exists!';
    }else{
        mysqli_query($conn, "INSERT INTO `user_details`(first_name, last_name, email, phone_number, password, department) VALUES ('$firstName',
        '$lastName', '$email', '$phone', '$password', '$department')") or die('query failed');
        $success_message = 'Registered successfully!';
    }
}

?>
```

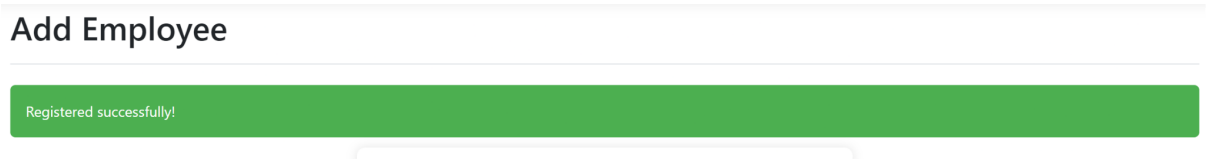
This block of PHP code is typically used for handling form submissions related to user registration. The code checks if the form has been submitted by checking the existence of the 'submit' key in the $_POST array (if(isset($_POST['submit']))).

If the form is submitted, the script proceeds to retrieve user input from the submitted form using the $_POST array. The values are assigned to corresponding variables after escaping them to prevent SQL injection using mysqli_real_escape_string.

A SQL query is then executed to check if a user with the provided email and password already exists in the 'user_details' table. The query result is stored in the variable $select_users. The code checks if there are any rows returned by the query (if(mysqli_num_rows($select_users) > 0)). If true, it means that a user with the provided email and password already exists, and an error message is assigned to the variable $error_message. If no user with the provided credentials is found, a new user is inserted into the 'user_details' table using another SQL query. If the insertion is successful, a success message is assigned to the variable $success_message.

This is then displayed to the admin as shown below or as shown in the front-end section of this guide.



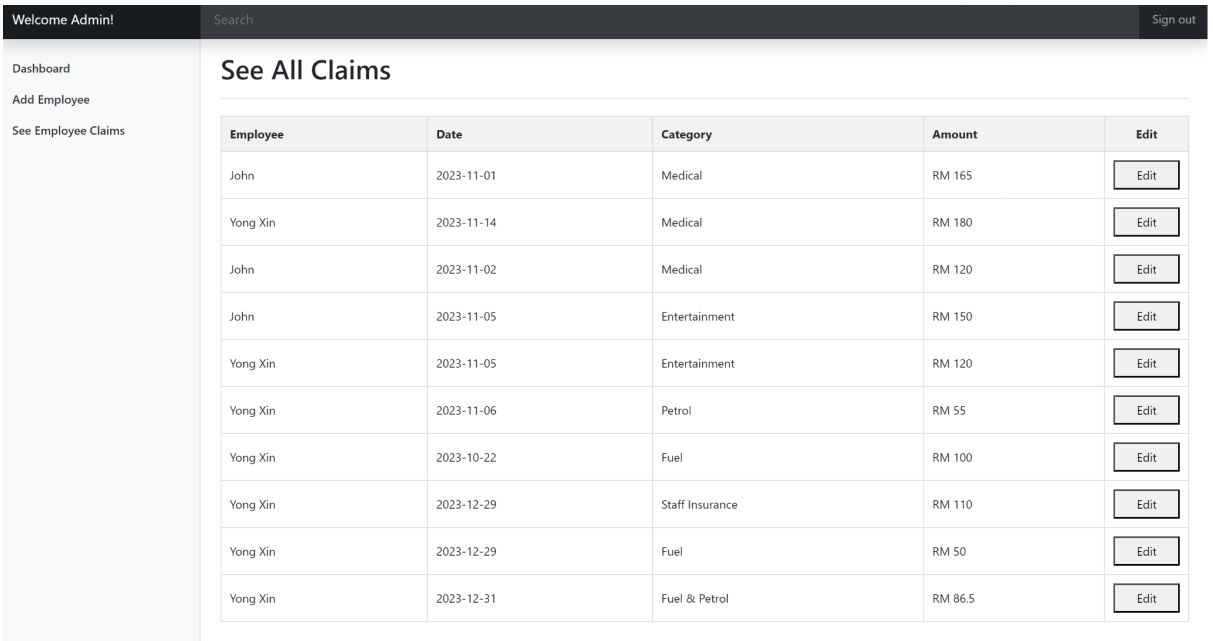


iii. See all employee claims & edit page

```php
<?php

if(isset($_SESSION['email'])) {
    $email = $_SESSION['email'];
    $db = new myConnection();

    // Fetch data from the MySQL table
    $sql = "SELECT id, user_details.first_name as firstName, date, month, category, amount FROM chart_data LEFT JOIN
    user_details on user_details.email = chart_data.email";
    $result = $db->query($sql);
}

if ($result->num_rows > 0) {
    // Display table header
```

This PHP code defines the necessary information needed to establish the see all claims page. It starts with a check to see if a user is logged in by verifying the existence of the 'email' key in the $_SESSION array.

If a user is logged in, it proceeds to fetch data from the MySQL database. If a user is logged in, the script retrieves the user's email from the session ($email = $_SESSION['email']) and instantiates a new instance of the myConnection class, which initiates connection with the mySQL database. A SQL query is then constructed to fetch data from the 'chart_data' table with additional information from the 'user_details' table using a LEFT JOIN. The query selects specific columns from both tables and links them based on the 'email' column. The result set is stored in the variable $result by executing the query using the query method of the database connection class. This result is used to display the all expense claim page on the admin section of the website as shown below.



```php
<?php
// edit_page.php

// Check if the ID parameter is set in the URL
if (isset($_GET['id'])) {
    $id = $_GET['id'];

    // Perform database connection and retrieve data based on ID
    $db = new myConnection();
    $sql = "SELECT * FROM chart_data WHERE id = '$id'";
    $result = $db->query($sql);

    if ($result->num_rows == 1) {
        $row = $result->fetch_assoc();

        // Check if the form is submitted for updates
        if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['save'])) {
            // Retrieve updated values from the form
            $updatedDate = $_POST['updated_date'];
            $updatedMonth = $_POST['updated_month'];
            $updatedCategory = $_POST['updated_category'];
            $updatedAmount = $_POST['updated_amount'];

            // Update the data in the database
            $updateSql = "UPDATE chart_data SET date = '$updatedDate', month = '$updatedMonth' ,category = '$updatedCategory', amount = '$updatedAmount'
            WHERE id = '$id'";
            $db->query($updateSql);

            // Redirect back to the main page after saving changes
            header("Location: user.php?module=See%20All%20Claims&page=claims");
            exit();
        }
        ?>
```

The page also has an edit button next to each of the entries on the database, to do manual editing by the admin or accountant if needed. The PHP code above handles the edit functionality by first checking if the 'id' parameter is set in the URL. If true, it retrieves the ID from the URL and establishes a connection to the database using the myConnection class.

A SQL query is then constructed to select all columns from the 'chart_data' table where the 'id' matches the provided ID from the URL. The result set is stored in the variable $result. The code ensures that there is exactly one row in the result set, fetching the row as an associative array. Subsequently, the script checks if the form has been submitted for updates by examining the request method and the presence of the 'save' button in the form data. If the form is submitted, the script retrieves the updated values from the form ($_POST) and constructs an SQL query to update the corresponding row in the 'chart_data' table with the new values. The update query is executed using the query method of the database connection class.

After the update is performed successfully, the script redirects the user back to the main page (user.php?module=See%20All%20Claims&page=claims) to view the modified data. This PHP code segment thus facilitates a user-friendly interface for editing and updating records of expenses from all employees, if needed.

Employee account:

i. Employee Dashboard page

The backend PHP functions add dynamic data display for employee accounts. The functions that have been used are described in this section.

```php
// where month is set by its relevant label
function data_chart_by_month($email){
    $datamon = [];
    $month = [1,2,3,4,5,6,7,8,9,10,11,12];

    foreach($month as $mon){
        $data = 0;  // Reset $data for each month
        $sqldata = "SELECT date, month, category, amount FROM chart_data WHERE month='$mon' AND email='$email'";
        $querydata = $this->query($sqldata);

        while($getdata = $querydata->fetch_array()){
            $data += $getdata['amount'];
        }

        $datamon[$mon] = $data;
    }

    return $datamon;
}
```

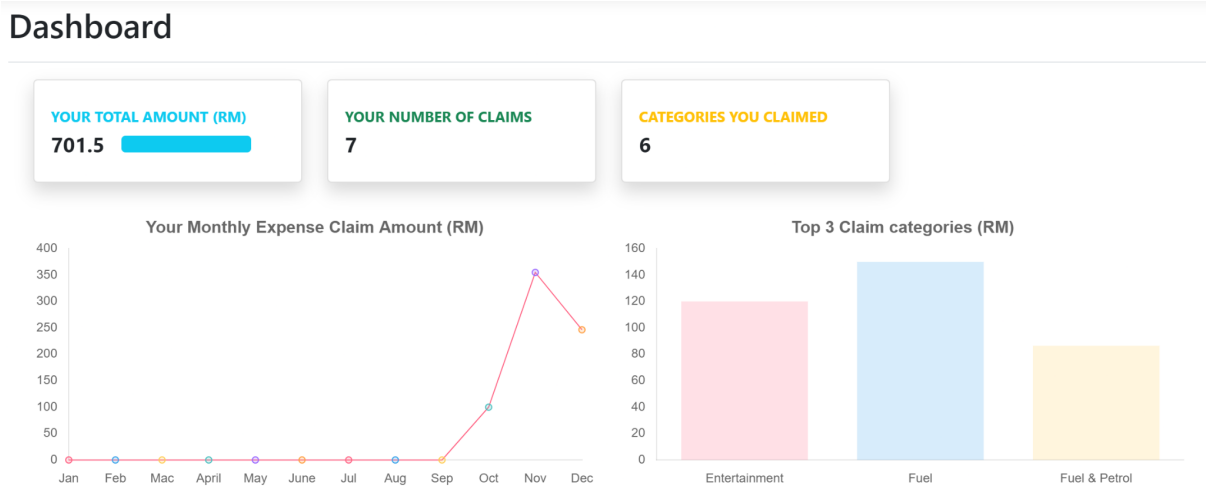
The provided PHP function, named data_chart_by_month, is designed to retrieve and organize data related to expense amounts for a specified user and each month of the year. This and codes below are similar to the admin integration, but take an additional $email as input based on the user that is logged and limit output that are related to that particular employee only.

```php
function data_chart_by_category($email){
    $datamon = [];
    $category = $this->category();

    foreach ($category as $sta) {
        $data = 0;  // Reset $data for each category
        $sqldata = "SELECT amount FROM chart_data WHERE category='$sta' AND email = '$email'";
        $querydata = $this->query($sqldata);

        while ($getdata = $querydata->fetch_array()) {
            $data += $getdata['amount'];
        }

        $datamon[$sta] = $data;
    }

    return $datamon;
}
```

This PHP function, named data_chart_by_category, is designed to retrieve and process data for a chart based on different expense categories across entire expense claims. The function

takes the user's email as a parameter and initializes an empty array $datamon to store data for each category. This is similar to the admin function as detailed above, but has an additional argument as the employee's email and outputs results based on the ones related to the employee email only.

With these two and the other SQL integration as defined in the admin section, the dashboard data is populated in the employee dashboard as shown below.



ii. Claim Expense

```php
if (in_array($imageFileType, $allowedExtensions)) {
    if (move_uploaded_file($_FILES['image']['tmp_name'], $uploadFile)) {

        // Call Python script with the uploaded file name
        $pythonScript = 'uploads/main.py'; // Replace with the actual path to your Python script
        // Execute the command
        $output = shell_exec("python $pythonScript $uploadName 2>&1");

        // Decode the JSON-like string into a PHP array
        $data = json_decode($output, true);

        if ($data === null || json_last_error() !== JSON_ERROR_NONE) {
          // Display an error div with a close button
          $error_message = "Failed to decode JSON string. Error: " . json_last_error_msg();
          include('error_message.php'); // Include a separate file for error message display
        } else {
          $status = $data['status'];
          $label = $data['label'];
          $category = urlencode($label);
          $amount = $data['amount'];
          $totalamount = urldecode($amount);

          if ($status === 400) {
            // Display an error div with a close button
            $error_message = "The receipt you uploaded isn't in MYR";
            include('error_message.php'); // Include a separate file for error message display
          } else {
            header("Location: user.php?module=Claim%20Expense&page=confirm&file_name=$file_path&category=$category&totalamount=$totalamount");
          }
        }
    } else {
        // Display an error div with a close button
        $error_message = 'Error uploading image.';
        include('error_message.php'); // Include a separate file for error message display
    }
} else {
    // Display an error div with a close button
    $error_message = 'Invalid file format. Please upload a valid image (jpg, jpeg, png, gif).';
    include('error_message.php'); // Include a separate file for error message display
}
}
```

This PHP code is part of the image upload and processing script, to integrate with the employee claims process. It enables employees to submit an expense claim by uploading a picture of the receipt. The code begins by checking if the uploaded file type (retrieved from $_FILES['image']['type']) is in the list of allowed extensions specified in the $allowedExtensions array. This is done to limit uploads to image file formats only. If the file type is valid, the script proceeds to move the uploaded file from its temporary location ($_FILES['image']['tmp_name']) to the specified upload directory ($uploadFile).

Upon successful file upload, the script calls a Python script (main.py) using the shell_exec function. The uploaded file name ($uploadName) is passed as an argument to the Python script. The output of the Python script, which is a JSON-like string, is captured and decoded into a PHP array using json_decode. It returns the category of the receipt as outputted by the machine learning model, the amount as outputted by the OCR mechanism and the status code. The code checks whether the decoding was successful and whether the decoded data contains valid JSON. If there is an issue with decoding, an error message is displayed, and an error div is included from a separate file (error_message.php).

If the JSON decoding is successful, the script extracts relevant information from the decoded array, such as the status, label, and amount. If the status indicates an error (status code 400), an error message specific to the situation is displayed. If there are no errors and the decoding is successful, the script constructs a URL with parameters for the next step (user.php?module=Claim%20Expense&page=confirm&file_name=$file_path&category=$category&totalamount=$totalamount). This URL is used for redirection to a confirmation page, passing along essential details such as the file path, category, and total amount.

In case of any errors during the process, such as issues with file upload, invalid file formats, or errors in Python script execution, appropriate error messages are displayed.

```php
<?php

include 'db_connection.php';

    if(isset($_SESSION['email'])){
      $email = $_SESSION['email'];
    }

    if (isset($_GET['file_name'])) {
        // Decode variables from URL
        $image_path = urldecode($_GET['file_name']);
        $category = urldecode($_GET['category']);
        $amount = urldecode($_GET['totalamount']);
        $claim_date = date('Y-m-d');
    }

    if (isset($_POST['submit'])) {

      $month = date('m', strtotime($claim_date));

      $insert_query = "INSERT INTO chart_data (category, amount, date, month, email) VALUES ('$category', '$amount',
      '$claim_date', '$month', '$email')";

      if (mysqli_query($conn, $insert_query)) {
          $message = urlencode("Claim successfully");
      } else {
          $message = urlencode("Error submitting data, please try again");
      }

      $label = urlencode($category);
      $total = urlencode($amount);
      $date = urlencode($claim_date);
      header("Location: user.php?module=Claim%20Expense&page=submitted&status=$message&label=$label&total=$total&date=$date");
  }


?>
```

Following that, the PHP code continues in the confirmation screen to display the output and wait for the employee to confirm submission. The initial section of the code focuses on user authentication. It checks if the user is authenticated by verifying the existence of the 'email' key in the session. If the user is authenticated, their email is retrieved and stored in the $email variable.

Following that, the script processes URL parameters. It checks for the presence of certain parameters (file_name, category, and totalamount). If these parameters exist in the URL, they are decoded and assigned to variables ($image_path, $category, $amount). Additionally, the $claim_date variable is set to the current date using the date function.

The script then handles the submission of the claim form. It checks if the 'submit' key is present in the $_POST array, indicating that the claim submission form has been submitted. If this condition is true, it extracts the month from the $claim_date variable and constructs an SQL query to insert the claim data (category, amount, date, month, email) into the 'chart_data' table.

The SQL query is executed using mysqli_query. If the insertion is successful, a success message (Claim successfully) is set in the $message variable; otherwise, an error message (Error submitting data, please try again) is set. These messages, along with other claim details, are URL-encoded and sent as parameters in the URL.

Finally, the script redirects the user to a confirmation page (user.php?module=Claim%20Expense&page=submitted) with the relevant status, label, total, and date parameters.

2. PHP with Python Integration through shell_exec():

The incorporation of Python into the backend workflow enhances the LightXpense web application's capabilities, particularly in the processing of receipt images submitted by users. This integration is witnessed in the following scenario: .

```
15          // Call Python script with the uploaded file name
16          $pythonScript = 'uploads/main.py'; // Replace with the actual path to your Python script
17          // Execute the command
18          $output = shell_exec("python $pythonScript $uploadName 2>&1");
19
20          // Decode the JSON-like string into a PHP array
21          $data = json_decode($output, true);
22
```

The shell_exec function is used to call the python script, which is stored in uploads/main.py. The functionality of Python script is essentially to perform OCR on the upload image, which extracts description and the total amount. Then, with the description, it also performs machine learning classification based on the accounting categories. This is then returned as a JSON object. The code below shows how the Python script handles input and returns output.

```
# Get input numbers from command line arguments
file_name = str(sys.argv[1])
path = 'uploads\\'
image_name = path + file_name

output = process(image_name)
output = " ".join(line.strip() for line in output.splitlines())

# Call function:
predicted_label = process_text(output)

# Call function:
amount = extract_text_files(output)

# Return as JSON-encoded string
if predicted_label:
    returnDict = {
        'status': 200,
        'label': predicted_label,
        'amount': amount
    }
else:
    returnDict = {
        'status': 400,
        'label': 'None',
        'amount': 'None'
    }

returnDict = json.dumps(returnDict)

print(returnDict)
```

This sophisticated backend integration, blending PHP with SQL and Python, underscores LightXpense's commitment to delivering a personalized and efficient expense management system. The seamless communication between the web application and the database, coupled with the dynamic capabilities of Python, ensures a robust foundation for user-specific interactions and data processing.

Responsiveness

```
/* ------------------------------------------------------------------
 * responsive:
 * home section
 * ------------------------------------------------------------------ */

@media only screen and (max-width: 1300px) {
  .home-image-right {
    right: 5rem;
  }
}

@media only screen and (max-width: 1200px) {
  #home {
    overflow: hidden;
  }
  .home-content {
    vertical-align: middle;
  }
  .home-content-left {
    padding: 18rem 0 12rem 40px;
  }
  .home-content-left h3 {
    font-size: 1.5rem;
  }
  .home-content-left h1 {
    font-size: 4.8rem;
  }
  .home-image-right {
    padding-top: 18rem;
  }
  .home-image-right img {
    vertical-align: top;
    width: 70%;
  }
}

@media only screen and (max-width: 1100px) {
  .home-content-left h1 {
    font-size: 4.6rem;
  }
```

In the CSS code for LightXpense, @media queries are utilized to make the webpage responsive for different screen sizes.

The first @media query, with a maximum width of 768 pixels, targets screens that are smaller, adjusting styles for improved responsiveness. The #home element's minimum height is set to 'auto,' allowing it to adjust based on its content. The font sizes of h3 and h1 elements

within the .home-content-left class are scaled down to 1.4rem and 3.6rem, respectively, ensuring better readability on smaller screens.

The second @media query, specifically designed for screens with a maximum width of 600 pixels, further refines the layout for even smaller devices. The padding of .home-content-left is adjusted to 12rem at the top, 6rem on the sides, and 366px at the bottom, optimizing the spacing for a smaller viewport. Font sizes for h1 and .home-social-list are modified to 3.3rem and 2rem, respectively, enhancing the visual appeal and user experience on devices with limited screen space. Additionally, the .button.stroke width is set to 100%, ensuring full-width display, and a maximum width of 280px is imposed. The scroll-icon position is shifted to 21px from the top, aligning it appropriately for a more cohesive design.

These @media queries play a crucial role in adapting the webpage's styles based on the screen width, enhancing the overall responsiveness and user experience across various devices. This has been used extensively throughout LightXpense website to ensure responsive experience.