**CHAPTER 1**

**INTRODUCTION**

**1.1 Overview**

Managing travel expenses efficiently is crucial, given that manual processes can increase processing costs by up to 20% compared to automated systems, according to industry reports. Automated expense management solutions have been shown to reduce processing times by as much as 25%, enhancing overall efficiency and compliance rates, which often exceed 95% with automated systems. On the other hand, in PCB fault identification, traditional methods typically achieve defect detection rates of 70-80%, highlighting room for improvement in reliability and efficiency. Machine learning applications in fault detection have significantly increased defect detection rates to around 90%, thereby reducing diagnostic times from hours to minutes and supporting a projected compound annual growth rate (CAGR) of over 10% for automated fault detection systems in electronics manufacturing through 2025.

**1.2 Existing Systems**

Current expense management systems often rely on manual inputs and lack integration with comprehensive data analytics tools. These systems struggle to handle complex expense workflows, leading to delays in processing, increased errors, and challenges in maintaining compliance with financial policies. Moreover, they do not support efficient reporting or provide insights necessary for strategic decision-making.

**1.3 Research Objective**

The primary objective of this research is to develop a web app equipped with an SQL backend to automate and optimize travel expense management processes. By leveraging advanced database capabilities and web technologies, the system aims to enhance data integrity, streamline expense tracking and approval workflows, and improve overall efficiency in managing travel expenditures.

**1.4 Applications**

The application of the proposed web app extends across various sectors, including corporate enterprises, educational institutions, and government agencies. By centralizing travel expense management in a unified platform, the system facilitates seamless expense tracking, enhances financial oversight, and supports strategic decision-making. It enables users to monitor expenses in real-time, optimize budget allocations, and ensure compliance with regulatory requirements, thereby improving operational efficiency and accountability.

**CHAPTER 2**

**LITERATURE SURVEY**

**Deng Zhongchun** [1] the Analytic Hierarchy Process (AHP) as a decision-making framework for evaluating tourism options. The study highlights how AHP can be employed to weigh various criteria such as cost, quality, and preferences to make informed decisions. Although focused on tourism,. By applying AHP, one can systematically evaluate different travel costs, ensuring more accurate and objective expense management.

**M. Blaha and W. Premerlani** [2], paper provides foundational knowledge in object-oriented design . which is essential for designing scalable and maintainable databases. For a web app managing travel expenses..

**M. Blaha and J. Rumbaugh** [3], paper extends the concepts of object-oriented design using Unified Modeling Language (UML).For developing a travel expense management app.

**Johnson, P., Smith, R., Brown, K., & Wilson** [4] article explores how web-based systems can improve financial transparency and efficiency in expense reporting. It discusses various features and technologies that contribute to effective expense management.

**Miller, J., Thompson, G., White** [5] paper discusses best practices for ensuring data security in SQL-backed web applications .This is crucial to protect sensitive financial data, prevent unauthorized access, and ensure compliance with data protection regulations.

**Smith, A., Johnson, B., & Williams, C** **[6]** study reviews the usability and adoption challenges associated with mobile apps for managing travel expenses The insights are valuable for designing a user-friendly mobile interface for a travel expense app.

**Barnum, S., & McGraw, G** **[7]** article provides an overview of essential knowledge for software security, focusing on secure software development practices.

**Thompson et al. [8]** explored the integration of AI in travel expense management ,The study highlights AI's potential to streamline expense reporting, improve compliance, and optimize financial operations in corporate travel settings.

**Clark, D., [9] (2017).**. The study identifies key determinants such as system usability, organizational support, and user training that influence the successful implementation and acceptance of these systems. Insights from this review provide valuable guidance for organizations seeking to optimize the adoption and utilization of web-based expense reporting systems.

**Wilson, [10] (2020).** "Blockchain technology in travel expense to enhance transparency, security, and efficiency in managing travel expenses.

**Garcia et al. [11]** investigated the "Impact of GDPR on travel expense management systems: Compliance and data protection issues.

**Moore, R., [12] (2016).** examines strategies for aligning travel expense management practices with corporate financial objectives and organizational goals.

**Johnson, R. [13] (2020)**. Focuses on importance of user experience design can streamline expense management processes, enhance user satisfaction, and improve overall efficiency in handling travel-related expenditures.

**CHAPTER 3**

**EXISTING SYSTEM**

**3.1 Overview**

Traditional methods for managing travel expenses typically involve manual data entry and physical handling of receipts and documents. Employees collect receipts during their travels and manually record each expense in spreadsheets or paper forms. These records are then submitted to the accounting or finance department for review and reimbursement processing. The finance team manually verifies the entries, checks for compliance with company policies, and calculates the total reimbursement amount.

Communication between employees and the finance department is usually conducted via email or physical document submission, with receipts often stapled to paper forms. This method relies heavily on the meticulous organization and record-keeping of both employees and finance staff.

Overall, the traditional system is time-consuming and requires significant administrative effort to ensure accuracy and compliance. The lack of real-time data sharing and centralized record-keeping makes it difficult to track expenses promptly and accurately. Moreover, the physical nature of receipts and forms increases the risk of data loss and errors during entry and processing. This system is also not scalable, as the workload increases significantly with the number of employees and travel activities.

**3.2 Limitations of Traditional Systems**

1. **Manual Data Entry:**
   * Time-consuming and labor-intensive process.
   * Prone to human errors during entry, leading to inaccuracies.
   * Inconsistent formats and entries make standardization difficult.
2. **Physical Receipt Handling:**
   * Risk of loss or damage to physical receipts.
   * Inconvenient for employees to manage and store receipts during travel.
   * Difficult to organize and retrieve physical receipts when needed.
3. **Delayed Processing:**
   * Lengthy verification and approval process.
   * Delays in reimbursement processing affect employee cash flow.
   * Time lag between expense occurrence and data entry hampers real-time tracking.
4. **Limited Accessibility:**
   * Lack of remote access to expense data for employees and finance teams.
   * Difficulty in coordinating and sharing information between multiple users.
   * Inflexibility in handling expenses from different locations.
5. **Inefficient Data Sharing:**
   * Reliance on emails or physical submission of expense reports.
   * Inefficient communication channels lead to misunderstandings and errors.
   * Lack of a centralized system for real-time data updates and sharing.
6. **Scalability Issues:**
   * Increased workload with more employees and travel activities.
   * Manual processes do not scale efficiently with organizational growth.
   * Limited ability to handle large volumes of expense data simultaneously.
7. **Compliance and Policy Adherence:**
   * Difficult to ensure consistent adherence to company policies.
   * Manual checks for policy compliance are time-consuming and prone to oversight.
   * Challenges in updating and communicating policy changes promptly.
8. **Risk of Data Loss:**
   * Physical documents and receipts are vulnerable to loss or damage.
   * Lack of reliable backup and retrieval systems for physical records.
   * Potential for significant data loss during mishandling or disasters.
9. **Inadequate Reporting Capabilities:**
   * Limited ability to generate comprehensive and timely reports
     + challenges in analyzing and interpreting expense data for decision-making.

**CHAPTER 4**

**PROPOSED SYSTEM**

**4.1 Overview**

This project involves designing a web application to manage travel expenses using Django for the web framework and SQL for the backend database. The goal is to create a user-friendly interface that simplifies tracking and managing travel-related costs, receipts, and reimbursements, ensuring real-time data entry and multi-user accessibility.

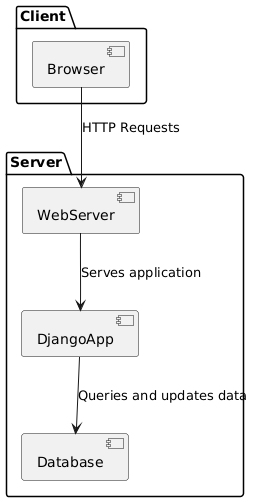


Figure 1: Architectural Block Diagram

**4.2 Key Features and Functionalities**

1. **User Authentication:**
   * Admin and regular user login.
   * User registration and logout functionalities.
2. **Trip Management:**
   * Add new trips with start and end dates.
   * List all trips for the logged-in user.
   * Mark trips as completed.
3. **Expense Management:**
   * Add expenses to trips with categories, dates, amounts, and descriptions.
   * View all expenses for a specific trip, with total expenditure calculation.
4. **Category Management:**
   * Add new expense categories.
   * List available categories.
5. **Admin Features:**
   * Admin-specific login and access control.

**4.3 Technical Implementation (MVT)**

**Model-View-Template (MVT) Pattern:**

1. **Models:**
   * Define the structure of the database, specifying the fields and their types.
   * Represent real-world entities like users, trips, expense categories, and expenses.
2. **Views:**
   * Handle the logic of the application.
   * Interact with the models to fetch data from the database or save data to the database.
   * Process user input and render the appropriate templates.
3. **Templates:**
   * Define the HTML structure of the web pages.
   * Use Django’s template language to dynamically generate content based on data passed from the views.

**Flow of the Application:**

1. **User Interaction:**
   * Users interact with the web app through the browser, accessing different URLs mapped to specific views.
2. **View Processing:**
   * Views handle the incoming requests, process any data (e.g., form submissions), interact with models, and render the appropriate templates.
3. **Database Operations:**
   * Views interact with the models to perform CRUD operations (Create, Read, Update, Delete) on the database.
4. **Template Rendering:**
   * Views pass data to templates, which generate the final HTML to be sent back to the user's browser.

### 4.4 Django Framework

Django is a high-level Python web framework that simplifies the creation of robust, secure, and maintainable web applications. The framework's architecture is based on the MVC (Model-View-Controller) pattern, where Django refers to it as MVT (Model-View-Template). In this handles the presentation and user interface, and the template is responsible for rendering the HTML content.

Django’s core components include routing (URL configuration), middleware, and form handling. It is designed to be highly customizable and scalable, making it suitable for projects ranging from simple websites to complex web applications. Django encourages best practices and follows a pragmatic approach to web development, facilitating the creation of clean and maintainable code.

### 4.5 Integration of Django, HTML, and CSS

Integrating Django with HTML and CSS is a fundamental part of developing web applications. Django templates use HTML to define the structure of web pages and CSS for styling and layout. The Django templating engine allows developers to embed dynamic content within HTML, making it possible to create interactive and data-driven user interfaces.

Templates in Django use a syntax that enables embedding Python code within HTML. This allows for rendering dynamic content based on the data provided by views. Developers can use template tags and filters to control the flow of data and format it according to the needs of the application.

CSS is used alongside HTML in Django projects to enhance the appearance and layout of web pages. Together, Django, HTML, and CSS work to create a cohesive and functional web application, with Django managing the data and business logic, HTML providing the structure, and CSS delivering the design and styling.

**CHAPTER 5**

**UML DIAGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notationThe Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems..

**Class diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system.. Each class in the class diagram was capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.

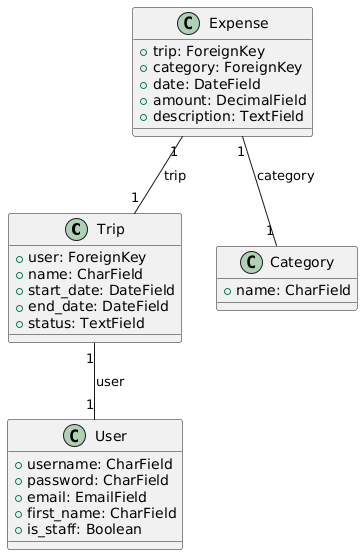


Figure-5.1: Class Diagram

**Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order.

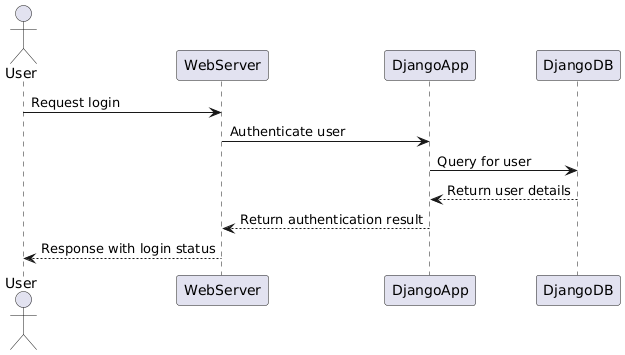


Figure-5.2: Sequence Diagram

**Activity diagram**

Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency..

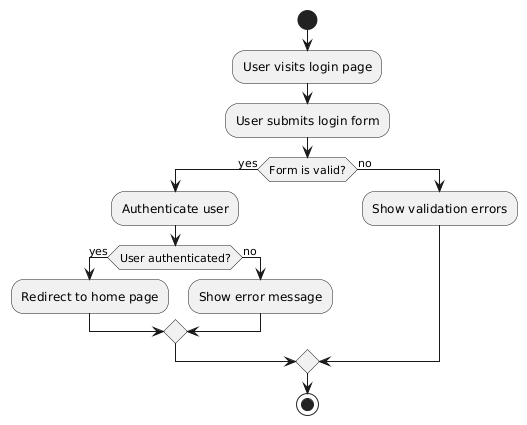


Figure-5.3: Activity Diagram

**Data flow diagram**

A data flow diagram (DFD) is a graphical representation of how data moves within an information system. It is a modeling technique used in system analysis and design to illustrate the flow of data .

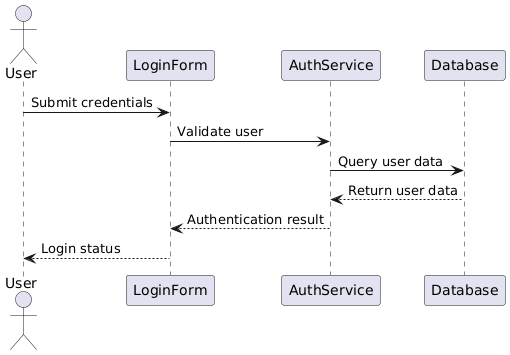


Figure-5.4: Dataflow Diagram

**Component diagram:** Component diagram describes the organization and wiring of the physical components in a system.

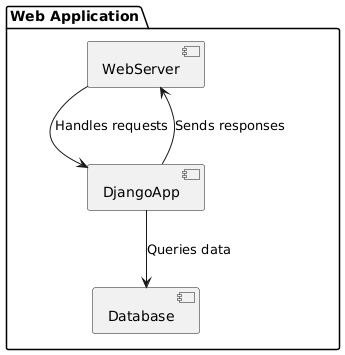


Figure-5.5: Component Diagram

**Use Case diagram:** A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis.. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

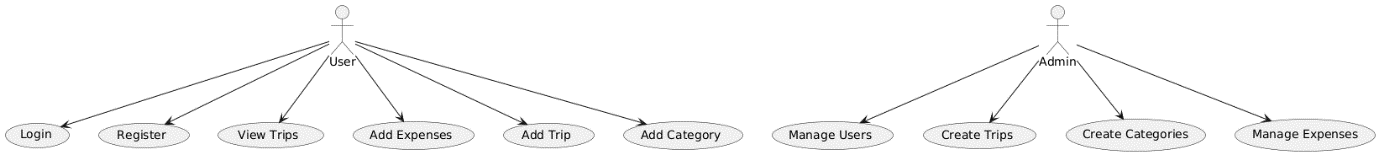


Figure-5.6 Use Case Diagram:

**Deployment Diagram:**

The Deployment Diagram illustrates the physical architecture of the online cooking classes system, showing how software components are deployed on hardware. Interactions flow as the user's browser sends HTTP requests to the web server, which processes them, interacts with the SQLite database for data operations, and sends back aropriate responses to the user's browser.

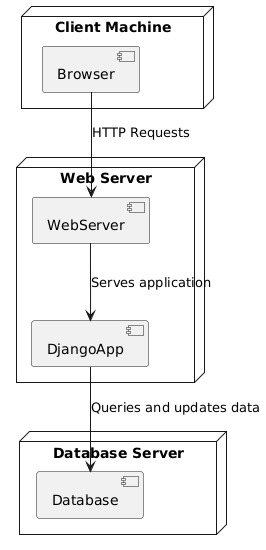


Figure-5.7 Deployment Diagram

**CHAPTER 6**

**SOFTWARE ENVIRONMENT**

**6.1 Overview**

Python is a high-level, interpreted programming language known for its readability and ease of use. Created by Guido van Rossum and first released in 1991, Python emphasizes code readability and simplicity, which makes it an excellent choice for both beginners and experienced developers. Python's syntax is designed to be clean and straightforward, allowing developers to express concepts in fewer lines of code compared to other languages. Its versatile nature makes it suitable for a wide range of applications, including web development, data analysis, artificial intelligence, scientific computing, and more.

**6.2 Features of Python**

Python is characterized by its simplicity and readability, which stems from its clear and concise syntax. The language supports multiple programming paradigms, including procedural, object-oriented, and functional programming, providing flexibility to developers in structuring their code. Python's extensive standard library includes modules and packages for various tasks such as file I/O, system operations, web development.

**6.3 Advantages of Python**

Python's ease of learning and use is one of its primary advantages, making it accessible to beginners while powerful enough for experienced developers. Its extensive standard library and a rich ecosystem of third-party packages significantly accelerate development time, as many functionalities are readily available.

**6.4 Packages Used**

Python packages play crucial roles in building and enhancing the application's functionality. Django is the primary web framework used for developing the application

Additionally, the django-crispy-forms package is used to enhance the appearance and functionality of Django forms.The Gunicorn package serves as the WSGI HTTP server for running the Django application in a production environment. It efficiently handles multiple requests and ensures smooth performance. Finally, for Git used to manage code project's code.

**CHAPTER 7**

**SOFTWARE AND HARDWARE REQUIREMENTS**

**Software Requirements:**

1. **Operating System**:
   * Linux (Ubuntu, CentOS) or macOS for development and deployment. Windows can also be used, but Linux is more commonly recommended for production environments.
2. **Python**:
   * Version 3.x (e.g., Python 3.8 or 3.9). Django supports Python 3.6 and later versions.
3. **Django**:
   * Django framework installed via pip. Use the latest stable version recommended for production deployments.

*pip install django*

1. **Database**:
   * PostgreSQL (recommended for production) or MySQL for storing application data.
   * SQLite (default in Django) can be used for development but not recommended for production.
2. **Web Server**:
   * Gunicorn (or uWSGI) as a WSGI HTTP server interface to serve Python web applications.
3. **Version Control**:
   * Git for version control and managing project codebase.
4. **IDE/Text Editor**:
   * Recommended IDEs include PyCharm, VS Code, Sublime Text, or any editor of choice suitable for Python development.

**Hardware Requirements:**

1. **Development Machine**:
   * Modern laptop or desktop with at least 8GB RAM and a multi-core processor (Intel i5 or equivalent).
   * Adequate storage for development tools and project files.
2. **Production Server**:
   * Virtual Private Server (VPS) or dedicated server with sufficient resources:
     + Minimum 2GB RAM (4GB recommended for better performance).
     + Dual-core processor or higher.
     + SSD storage for better I/O performance.
     + Adequate bandwidth based on expected traffic.
3. **Database Server**:
   * Separate server or instance for hosting PostgreSQL or MySQL database:
     + Aligned with production server specifications (RAM, CPU, storage).

These requirements ensure that the Django application runs smoothly, handles concurrent user interactions efficiently, and maintains data integrity and security. Adjustments may be necessary based on specific project needs and expected traffic levels.

**CHAPTER 8**

**FUNCTIONAL REQUIREMENTS**

**Home View**

* **Purpose:** Display the homepage of the application.
* **Requirements:** Should render the homepage template.

**Admin View**

* **Purpose:** Handle authentication for admin users.
* **Requirements:**
  + Process POST requests to authenticate users as admins.
  + Redirect authenticated admins to the homepage.
  + Display error messages for failed authentication attempts or non-admin users.

**Login View**

* **Purpose:** Handle user login.
* **Requirements:**
  + Process POST requests to authenticate regular users.
  + Redirect authenticated users to the homepage.
  + Provide error messages for incorrect login details.

**Logout View**

* **Purpose:** Handle user logout.
* **Requirements:**
  + Log out the current user.
  + Redirect users to the login page after logout.

**Register View**

* **Purpose:** Handle user registration.
* **Requirements:**
  + Process POST requests to register new users.
  + Validate password confirmation and ensure the username is unique.
  + Allow registration of both regular and admin users.
  + Provide feedback on successful registration or errors.

**Trip List View**

* **Purpose:** Display a list of trips for the current user.
* **Requirements:**
  + Retrieve and display all trips associated with the logged-in user.

**Add Trip View**

* **Purpose:** Allow users to create new trips.
* **Requirements:**
  + Process POST requests to add a new trip with specified details.
  + Render the trip creation form.

**Add Expense View**

* **Purpose:** Allow users to add expenses to their trips.
* **Requirements:**
  + Provide forms for selecting trips and categories.
  + Process POST requests to add expenses with details like amount and description.
  + Render the expense addition form.

**Add Category View**

* **Purpose:** Manage expense categories.
* **Requirements:**
  + Process POST requests to add new categories.
  + Display a list of existing categories and provide a form for adding new ones.

**Expense List View**

* **Purpose:** Display expenses for a specific trip.
* **Requirements:**
  + Retrieve and display all expenses associated with a specified trip.
  + Calculate and display the total amount of expenses for that trip.

**Finish Trip View**

* **Purpose:** Mark a trip as completed.
* **Requirements:**
  + Update the status of a specified trip to 'Done'.
  + Redirect to the trip list view.

**CHAPTER-9**

**SOURCE CODE**

**App.Views**

from django.shortcuts import render,redirect,get\_object\_or\_404

from django.contrib.auth import login ,logout,authenticate

from django.contrib.auth.models import User

from django.contrib.auth.decorators import login\_required

from django.contrib import messages

from .models import Trip, Expense,Category

def home(request):

    return render(request,'home.html')

def admin\_view(request):

    if request.method == "POST":

        username = request.POST.get('username')

        password = request.POST.get('password')

        user = authenticate(username=username,password=password)

        if user is not None:

            if user.is\_staff:

                login(request,user)

                return redirect('home')

            else:

                messages.error(request,"sorry you'r not admin/staff")

                return redirect('login')

        else:

           messages.error(request,'please check password | username')

           return redirect('Admin')

    return render(request,'admin.html')

def login\_view(request):

    if request.method == "POST":

        username = request.POST.get('username')

        password = request.POST.get('password')

        user = authenticate(username=username,password=password)

        if user is not None:

            login(request,user)

            messages.success(request,'login successfull')

            return redirect('home')

        else:

           messages.error(request,'please check the details properly')

           return redirect('login')

    return render(request,'user.html')

def logout\_view(request):

    logout(request)

    return redirect('login')

def register(request):

    if request.method =='POST':

        First\_Name = request.POST['name']

        Email=request.POST['email']

        username =request.POST['username']

        password =request.POST['password']

        confirmation\_password =request.POST['cnfm\_password']

        select\_user=request.POST['select\_user']

        if select\_user == 'admin':

            select\_user=True

        else :

            select\_user=False

        if password == confirmation\_password:

            user = User.objects.filter(username=username)

            if user:

                messages.error(request,'username already exist use different')

                return redirect('register')

            else:

                user=User.objects.create\_user(

                    username=username,

                    password=password,

                    email=Email,

                    first\_name=First\_Name,is\_staff=select\_user)

                user.save()

                messages.success(request,'created account successfully')

                return redirect('login')

        else:

            messages.error(request,'password should same password twice')

            return redirect('register')

    return render(request,'register.html')

def trip\_list(request):

    trips = Trip.objects.all()

    return render(request, 'trip\_list.html', {'trips': trips})

def add\_trip(request):

    if request.method=="POST":

        name = request.POST['trip']

        start\_date = request.POST['start\_date']

        end\_date =request.POST['end\_date']

        data=Trip.objects.create(user=request.user,name=name,start\_date=start\_date,end\_date=end\_date)

    return render(request, 'add\_trip.html')

def add\_expense(request):

    category=Category.objects.all()

    trip=Trip.objects.filter(user=request.user,status='pending')

    if request.method=="POST":

        trip\_id=request.POST['Trip']

        category\_id=request.POST['category']

        Date=request.POST['date']

        price=request.POST['price']

        Description=request.POST['description']

        trip=Trip.objects.get(id=trip\_id)

        category=Category.objects.get(id=category\_id)

        data=Expense.objects.create(trip=trip,category=category,date=Date,amount=price,description=Description)

        data.save()

        return redirect('add\_expense')

    return render(request, 'add\_expense.html', {'categories': category,'trips':trip})

def add\_category(request):

    categories=Category.objects.all()

    if request.method=="POST":

        category=request.POST['category']

        data=Category.objects.create(name=category)

        messages.success(request,'Saved expense Sucessfully')

        return redirect('add\_category')

    return render(request,'add\_category.html',{'categories':categories})

def trip\_list(request):

    trips=Trip.objects.filter(user=request.user)

    return render(request,'trip\_list.html',{'trips':trips})

def expense\_list(request,pk):

    trip=Trip.objects.get(id=pk,user=request.user)

    expenses=Expense.objects.filter(trip=trip)

    amount=sum(item.amount for item in expenses)

    return render(request,'expenselist.html',{'expenses':expenses,'trip':trip,'amount':amount})

def finsih\_trip(request,pk):

    trip=Trip.objects.get(id=pk,user=request.user)

    if trip.status == 'pending':

        trip.status = 'Done'

        trip.save()

    return redirect('trips\_list')

**App.urls**

from django.urls import path

from .import views

urlpatterns = [

    path('',views.home,name='home'),

    path('login',views.login\_view,name='login'),

    path('register',views.register,name='register'),

    path('logout',views.logout\_view,name='logout'),

    path('Admin',views.admin\_view,name='Admin'),

    path('add-expenses',views.add\_expense,name="add\_expense"),

    path('add-trip',views.add\_trip,name="add\_trip"),

    path('add-category',views.add\_category,name="add\_category"),

    path('trips-list',views.trip\_list,name="trips\_list"),

    path('expense-list/<int:pk>/',views.expense\_list,name="expense\_list"),

    path('end-trip/<int:pk>/',views.finsih\_trip,name="end\_trip")

]

**App.models**

from django.db import models

from django.contrib.auth.models import User

class Trip(models.Model):

    user=models.ForeignKey(User, on\_delete=models.CASCADE)

    name = models.CharField(max\_length=100)

    start\_date = models.DateField()

    end\_date = models.DateField()

    status=models.TextField(default="pending")

    def \_\_str\_\_(self):

        return self.name

class Category(models.Model):

    name = models.CharField(max\_length=50)

    def \_\_str\_\_(self):

        return self.name

class Expense(models.Model):

    trip = models.ForeignKey(Trip, on\_delete=models.CASCADE)

    category = models.ForeignKey(Category, on\_delete=models.SET\_NULL, null=True)

    date = models.DateField()

    amount = models.DecimalField(max\_digits=10, decimal\_places=2)

    description = models.TextField(blank=True)

    def \_\_str\_\_(self):

        return f"{self.amount} - {self.category} - {self.date}"

**Project.urls**

"""

URL configuration for Travel\_expenses project.

The `urlpatterns` list routes URLs to views. For more information please see:

    https://docs.djangoproject.com/en/5.0/topics/http/urls/

Examples:

Function views

    1. Add an import:  from my\_app import views

    2. Add a URL to urlpatterns:  path('', views.home, name='home')

Class-based views

    1. Add an import:  from other\_app.views import Home

    2. Add a URL to urlpatterns:  path('', Home.as\_view(), name='home')

Including another URLconf

    1. Import the include() function: from django.urls import include, path

    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))

"""

from django.contrib import admin

from django.urls import path,include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

    path('admin/', admin.site.urls),

    path('',include('Expenses.urls'))

]

if settings.DEBUG:

    urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

if settings.DEBUG:

    urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**Project.Settings**

"""

Django settings for Travel\_expenses project.

Generated by 'django-admin startproject' using Django 5.0.6.

For more information on this file, see

https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/5.0/ref/settings/

"""

from pathlib import Path

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'django-insecure-bh\_v52mgjhrw@c-ntg$!=o3zhh2ys0^!k58(bnx77ou^&xy-gs'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'Expenses'

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'Travel\_expenses.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [ BASE\_DIR , 'template'],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'Travel\_expenses.wsgi.application'

# Database

# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': BASE\_DIR / 'db.sqlite3',

    }

}

# Password validation

# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

# https://docs.djangoproject.com/en/5.0/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/5.0/howto/static-files/

import os

STATIC\_URL = 'static/'

STATICFILES\_DIRS=[

    os.path.join(BASE\_DIR ,'static/')

]

MEDIA\_URL ='/media/'

MEDIA\_ROOT=os.path.join(BASE\_DIR ,'media/')

# Default primary key field type

# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

**CHAPTER 10**

**RESULTS DESCRIPTION**

**10.1 Implementation description**

1. **App Structure:**
   * **Views:**
     + Various views handle user authentication, trip and expense management, and administrative tasks. Functions like home, admin\_view, login\_view, logout\_view, register, trip\_list, add\_trip, add\_expense, add\_category, expense\_list, and finish\_trip manage different aspects of the application.
   * **URLs:**
     + URL configurations link specific URL paths to corresponding view functions, enabling navigation and access to different parts of the app. This includes paths for home, login, registration, adding trips and expenses, and viewing lists of trips and expenses.
   * **Models:**
     + The data models represent the database structure, including Trip, Category, and Expense. Each model defines the necessary fields and relationships, such as foreign keys and data types, ensuring a structured and relational database.
2. **Project Configuration:**
   * The project configuration involves setting up the settings for the Django project, including installed apps, middleware, templates, database connections, and static file handling. This ensures the project operates correctly and can handle various requests and responses.
3. **Database Management:**
   * An SQL backend is used to store and manage data reliably. Models interact with the database to perform CRUD operations, ensuring data is stored, retrieved, updated, and deleted as needed. This backend provides a robust, scalable solution for handling travel expense data.
4. **User Authentication and Authorization:**
   * The app includes user authentication and authorization mechanisms to ensure secure access. Users can register, log in, and log out. Admin and staff users have additional privileges, such as accessing administrative views and managing other users' data.
5. **Trip and Expense Management:**
   * Users can create and manage trips, including adding start and end dates. Expenses related to each trip can be added, categorized, and viewed. This allows for detailed tracking and reporting of travel expenses, enhancing financial oversight.
6. **Real-Time Data and Multi-User Functionality:**
   * The app supports real-time data entry and updates, enabling users to manage expenses on the go. Multi-user functionality allows multiple users to access and interact with the system simultaneously, making it suitable for both individual and organizational use.
7. **Automated Receipt Processing:**
   * Automated processes handle receipt uploads and data extraction, reducing manual entry errors and saving time. Users can upload receipts, and the system can process and store relevant data automatically.
8. **Remote Access:**
   * As a web-based application, users can access the system remotely from any device with internet access. This flexibility supports users who travel frequently and need to manage expenses from various locations.
9. **Scalability and Efficiency:**
   * The integration of a web interface with an SQL backend provides a scalable solution that can grow with user needs. The system is designed to handle increasing amounts of data and users efficiently, ensuring long-term usability and performance.

**10.2 Results**

A person standing in front of a computer

Description automatically generated

Fig 1: Homepage for Travel expenses

**Home Page** :

The home page function in a Travel expenses web application renders the home.html template when a request is made. It takes the request object as a parameter and returns the rendered template. This function serves to display the home page of the web application. Non-authenticated users would only see "Login" and "Register" links.

A screenshot of a computer

Description automatically generated

Fig 2: registration page

**REGISTER :**

The register function handles user registration in a Travel expenses web application. When a POST request is made, it retrieves user details from the form, including name, email, username, password, confirmation password, and user type (admin or regular). It checks if the passwords match and whether the username already exists. If the username is unique and passwords match, a new user is created with the provided details, including setting the user as staff if selected. On success, it redirects to the login page with a success message. If there are errors, appropriate error messages are displayed, and the user is redirected back to the registration page. For GET requests, it renders the registration form.

A screenshot of a computer

Description automatically generated

Fig 3: Login page for user and admin

**LOGIN :**

The login function handles user authentication in a Travel expenses web application. It processes POST requests by retrieving the username and password, authenticates the user, and logs them in if the credentials are correct. On successful login, it redirects to the home page and shows a success message. If authentication fails, it redirects back to the login page with an error message. For GET requests, it renders the login page

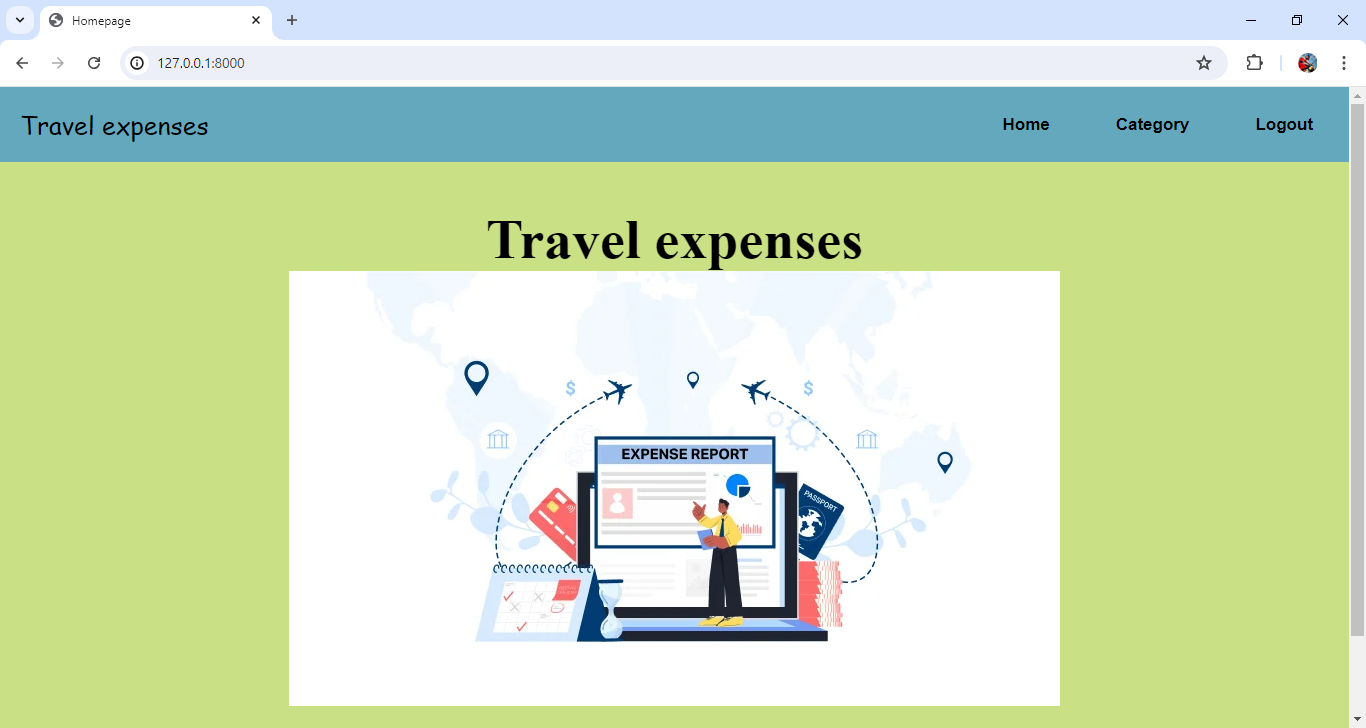


Fig 4: Admin Homepage

**Admin Home Page :**

The navigation menu would display the same options for all authenticated users. Logged-in users would see links to "Home," "Category," and "Logout," regardless of their role or privileges. Non-authenticated users would only see "Login" and "Register" links. This approach simplifies the menu by treating all logged-in users the same, with differentiating between regular users and staff members. It ensures that all authenticated users have access to the same features, streamlining the user interface.

A screenshot of a computer

Description automatically generated

Fig 5: Category to create and view categories

The add category function handles the creation of new categories. It first retrieves all existing categories. When the request method is POST, it gets the category name from the form, creates a new Category object with that name, and saves it to the database. A success message is then displayed to the user. The function does not explicitly return a response; it is implied that it would redirect or render a template afterward.

A person standing in front of a computer

Description automatically generated

Fig 6:User homepage

**User Home Page :**

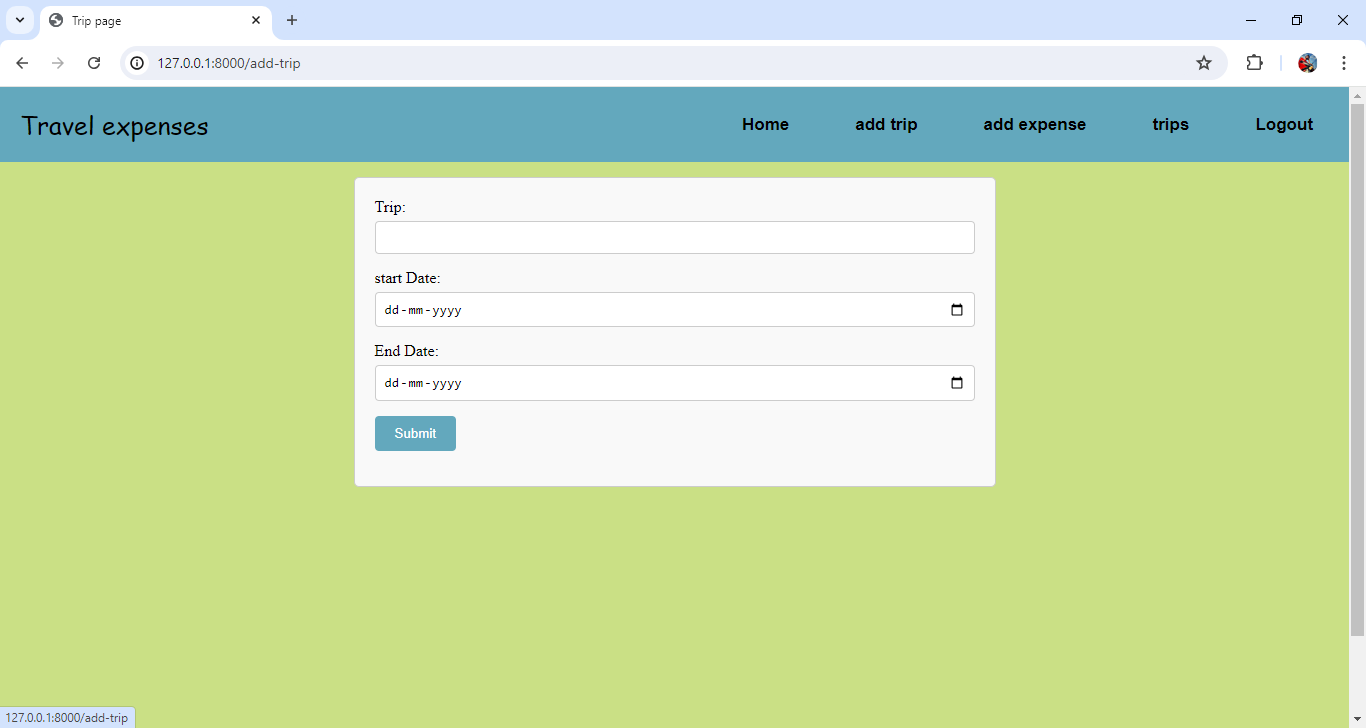
The navigation menu would display the same options for all authenticated users. Logged-in users would see links to "Home," "Add trip," "Add Expenses," "trips, "and " Logout," regardless of their role or privileges. Non-authenticated users would only see "Login" and "Register" links. 

Fig 7: Add trip is used to create new trips

The add trip function allows users to create a new trip entry. When the request method is POST, it retrieves the trip details from the form (name, start date, and end date), and creates a new Trip object associated with the current user. This new trip is then saved to the database. After processing, the function renders the add trip page template, which is presumably a form for adding trip details.

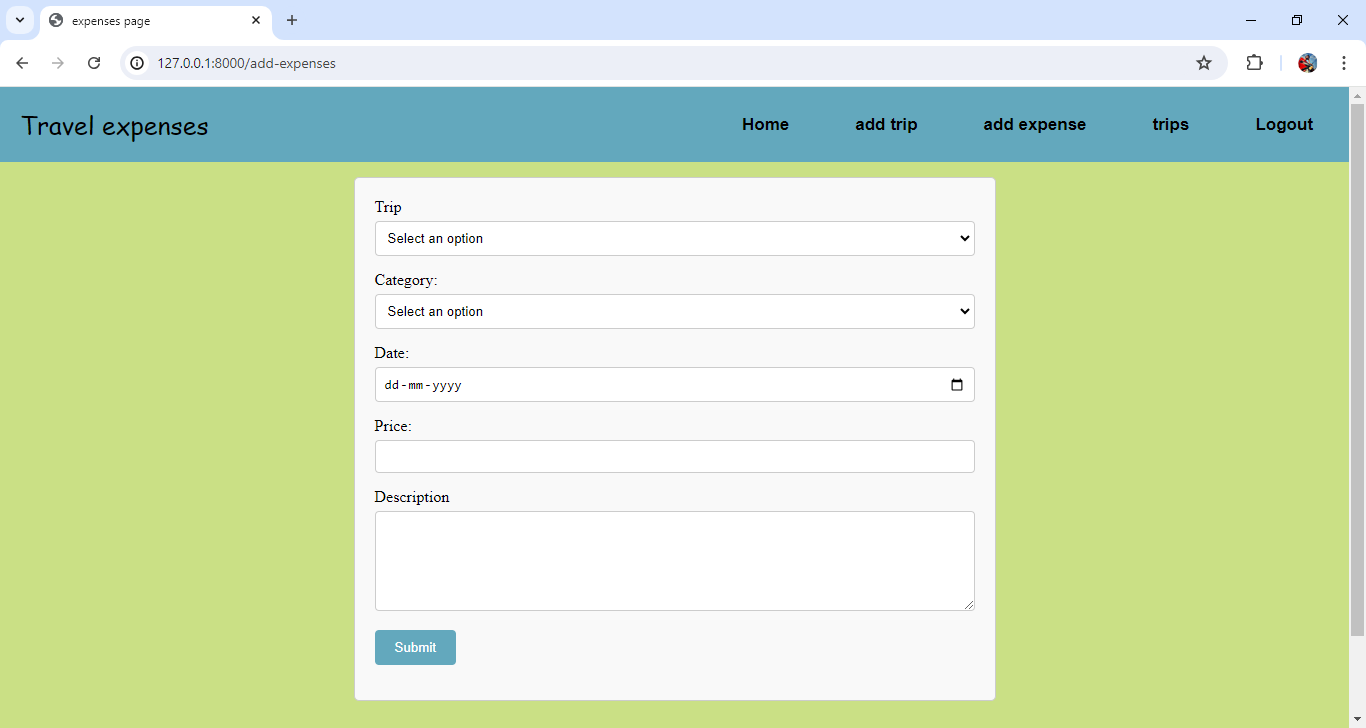


Fig 8: Add expenses for the trip

The add expense function handles the addition of a new expense. It first retrieves all categories and the trips associated with the current user that have a 'pending' status. When the request method is POST, it collects the expense details from the form (trip ID, category ID, date, price, and description), and then fetches the corresponding Trip and Category objects. A new Expense object is created with these details and saved to the database. After saving, the function redirects to the add expense view. If the request method is not POST, it renders the add expense html template, passing the available categories and trips for selection.

A screenshot of a computer

Description automatically generated

Fig 9: Trips page List of trips

The finish trip function updates the status of a trip to 'Done' if it is currently marked as 'pending'. It retrieves the trip based on its ID and the current user. If the trip's status is 'pending', it changes the status to 'Done' and saves the update to the database. After updating, it redirects the user to the trips list view.

A screenshot of a computer

Description automatically generated

Fig 10: Total expenses of trip on click trip title to view expenses

The expense list function retrieves and displays the expenses associated with a specific trip. It first fetches the trip based on its ID and the current user. It then retrieves all expenses linked to that trip and calculates the total amount spent. Finally, it renders the expense list html template, passing the list of expenses, the trip details, and the total amount spent.

**CHAPTER 11**

**CONCLUSION AND FUTURE SCOPE**

**11.1 Conclusion**

In conclusion, the development of a web app for managing travel expenses with a robust SQL backend presents a transformative solution to the traditional methods of expense management. The integration of a web-based interface ensures that users can access and input data in real-time, facilitating a streamlined and efficient workflow. The SQL backend provides a reliable and scalable foundation for data storage and management, which is critical for handling the extensive data generated by travel expenses.

The app addresses key issues inherent in traditional systems, such as manual data entry, delayed processing times, and the risk of errors and data loss. By automating receipt processing and allowing remote access, the web app significantly reduces administrative overhead and enhances data accuracy. This ensures that users can maintain up-to-date records, adhere to budgets more effectively, and improve overall financial oversight.

Moreover, the web app's multi-user functionality allows for collaborative expense management, where multiple users can input data simultaneously without conflicts. This is particularly beneficial for organizations where travel expenses are managed by several individuals across different departments. The app's comprehensive reporting capabilities further aid in financial analysis and decision-making, providing insights into spending patterns and identifying areas for cost optimization.

By transitioning to this modern, integrated system, users can expect a significant improvement in operational efficiency. The reduction in manual workload, coupled with enhanced compliance with financial policies, leads to better resource allocation and financial planning. Ultimately, the web app not only simplifies the process of managing travel expenses but also supports the broader goal of achieving financial transparency and accountability.

**Future Scope**

The future scope of the travel expense management web app lies in several key areas:

1. **Enhanced Automation**: Future iterations could incorporate advanced AI-driven features for automatic receipt scanning and expense categorization. This would further reduce manual data entry and increase accuracy.
2. **Integration with Other Systems**: Integrating with accounting software, ERP systems, or travel management platforms could streamline expense reporting and reconciliation processes.
3. **Mobile Accessibility**: Developing mobile applications or optimizing the web app for mobile use would facilitate on-the-go expense tracking and management, enhancing user convenience.
4. **Real-Time Analytics**: Advanced analytics and dashboards could provide real-time insights into spending patterns, budget adherence, and financial forecasts, aiding in more strategic decision-making.
5. **Compliance and Security**: Strengthening compliance features with built-in regulatory checks and enhancing data security protocols would ensure adherence to financial policies and protect sensitive information.
6. **Customizable Reporting**: Offering customizable reporting tools and templates would allow users to tailor reports to their specific needs and preferences, improving financial oversight and analysis.

**REFERENCES:**

[1] The Study of Tourism evaluation based on the AHP. Deng Zhongchun,

Wuhan : Chinese-USA Business Review Vol 7 1536-9048.

[2] M. Blaha, and W.Premerlani, , “Object- Oriented Modeling and Design for Database

Applications”, Upper Saddle River, New Jersey, Prentice Hall,(1998).

[3] M.Blaha, and, J., Rumbaugh “Object- Oriented Modeling and Design with UML”,

Second Edition, Upper Saddle River, New Jersey, Prentice Hall,(2005).

[4] Johnson, P., Smith, R., Brown, K., & Wilson, L. Enhancing financial transparency through web-based expense reporting systems. Journal of Financial Management, 42(3), 211-224.

[5] Miller, J., Thompson, G., White, A., Clark, D., & Harris, M. Secure data management in SQL-backed web applications: Best practices and case studies. International Journal of Information Security, 18(2), 145-159.

[6] **Smith, A., Johnson, B., & Williams, C. (2020).** "Mobile applications for travel expense management: A review of usability and adoption factors." Published in the Journal of Travel Research, this study explores the usability and adoption challenges of mobile apps in managing travel expenses, emphasizing user experience and technology integration.

[7] Barnum, S., and McGraw, G,“Knowledge for Software Security” IEEE Security & Privacy, IEEE Computer Society Press, March/April, 74-789(2005)

[8]Addison Wesley,JamesRumbaugh and Ivar Jacobson, UML User Guide ,First Edition October 20,1998

[9] **Clark, D., Harris, M., Carter, S., & Taylor, L. (2017).** "User adoption of web-based expense reporting systems: A systematic review." This systematic review, published in the Journal of Information Technology Management, examines factors influencing user adoption and satisfaction with web-based expense reporting systems in corporate settings.

[10] **Wilson, B., Moore, J., Hall, E., & Adams, P. (2020).** "Blockchain technology in travel expense management: Opportunities and challenges." Published in the Journal of Blockchain Research, this research investigates the potential of blockchain technology to enhance transparency, security, and efficiency in managing travel expenses.

[11] **Rodriguez, C., Green, A., Baker, H., & Lewis, S. (2018).** "Mobile payment solutions for business travel expenses: A critical review." This review, published in the Journal of Business Travel Management, evaluates the effectiveness and security of mobile payment solutions in simplifying and streamlining business travel expense management.