Adobe Internship - 2022
CJM project
Technical Document

ScoPortal

Document Control

Version Control

Version	Date	Author	Description
1	14/07/2019	Hariharan B	First draft for internal circulation

Table of Contents

1 Document Overview

1.1 Purpose and Scope

ScoPortal is a Scoping Requirement Automation web-based tool, implemented as part of Adobe Internship-2022 CJM project.

This document will provide technical details of different modules developed in ScoPortal

1.2 Assumptions

This document will be uploaded to the GDC Sharepoint site and then maintained by the GDC team.

1.3 Codebase Details

The codebase is hosted on https://git.corp.adobe.com/gigeorge/Scoping_Automation

2 Design Elements

2.1 High Level Design Areas

The following areas are expected to deliver design elements as part of the solution:

Design Area	Description
Database Connection	Connection of Portal to back-end Database
Signup	One-time registration of User for the tool access
Login	Entry page for the access of the tool, by authenticating registered Users
Navigation Bar	Responsive Navigation Header, common to all pages
Home	Dashboard of the portal, illustrates Ticket data in visual format
All Tickets	Displays all the Ticket data together
Current Ticket	Display all details regarding a specific Ticket
Alert Mechanism	Provision for sending out Email alerts to Users regarding change in Ticket Status

3 Database Connection

3.1 Overview

- The database is hosted on a PostgreSQL server

3.2 Details

- The connection to the database is made from ***create_app()*** in ***__init__.py***.

```
def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = ''+str(config['database']['secret_key'])+''
    app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:'+str(config['database']['URI_pwd'])
    +'@localhost/'+str(config['database']['URI_DB_name'])
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
```

- The database URI and password can be defined from ***config.ini***.

```
[database]
URI_pwd = <database-password>
URI_DB_name = <database-name>
secret_key = ScoPortal
```

- The database Schema is defined in ***models.py.***

```
class User(db.Model, UserMixin):
    __tablename__ = 'User'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(200),unique=True)
    username = db.Column(db.String(150),unique=True)
    password = db.Column(db.String(150))
    usertype = db.Column(db.String(20))
    date_created = db.Column(db.DateTime, default = time.strftime("%d/%B/%Y %H:%M:%S") )
    comments = db.relationship('Comment',backref='User',passive_deletes=True)
    status = db.Column(db.String(20))
```

4 Signup

- This functionality is implemented using the ***flask_login*** module. The function ***signup()*** is defined within ***auth.py***.

```
@auth.route("/signup", methods=['GET','POST'])
def signup():
    if request.method == 'POST':
        email = request.form.get("email")
        username = request.form.get("username")
        password1 = request.form.get("password1")
        password2 = request.form.get("password2")
        usertype = request.form.get("usertype")
        domain = email.split("@")[1]
        email_exists = User.query.filter_by(email=email).first()
        username_exists = User.query.filter_by(username=username).first()
        if email_exists:
            flash('Email already in use',category='error')
        elif (domain != "adobe.com"):
            flash('Please use your Adobe Email',category='error')
        elif username_exists:
            flash('Username is taken',category='error')
        elif password1 != password2:
            flash('Passwords do not match',category='error')
```

- 3 types of users can be created, with varying permissions, namely *Admin, Reporter* and *Assignee*.
- The Front-End html file (***signup.html***) is found under the ***templates*** folder.

4.1 Profile Details

- This functionality is accessible to all users, allowing them to view their Profile details and change their Password if needed.
- This takes place in the function ***view_profile()*** in ***views.py.***

```
@views.route("/viewprofile",methods=['GET','POST'])
@login_required
def view_profile():
    if request.method == 'POST':
        password1 = request.form.get("password1")
        password2 = request.form.get("password2")
        if password1 != password2:
            flash('Passwords do not match',category='error')
        elif len(password1) < 6:
            flash('Password is too short',category='error')
        else:
            user=current_user
            password = generate_password_hash(password1,method="sha256")
            user.password=password
            db.session.commit()
            login_user(user,remember=True)
            flash('Password changed successfully')
            return redirect(url_for('views.home'))
    return render_template('viewprofile.html',logopath=logopath,user=current_user)
```

- The Front-End html file (***viewprofile.html***) is found under the ***templates*** folder.

5 Login

- This functionality is implemented using the ***flask_login*** module. The functions for login and logout are defined within ***auth.py***.

```
@auth.route("/login", methods=['GET','POST'])
def login():
    if request.method == 'POST':
        email = request.form.get("email")
        password = request.form.get("password")
        user = User.query.filter_by(email=email).first()
        if user:
            if check_password_hash(user.password,password):
                login_user(user, remember=True)
                return redirect(url_for('views.home'))
            else:
                flash('Password is incorrect',category='error')
        else:
            flash('Email does not exist',category='error')
    return render_template("login.html",user=current_user,logopath=logopath)
```

- The Front-End html file (***login.html***) is found under the ***templates*** folder.

5.1 Forgot Password

- This functionality is accessible to all users, allowing them to reset their Password if they have forgotten it.
- This takes place in the function ***forgot_password()*** in ***views.py.***

```
@views.route("/forgot-password",methods=['GET','POST'])
def forgot_password():
    if request.method == 'POST':
        username=request.form.get("username")
        password1 = request.form.get("password1")
        password2 = request.form.get("password2")
        if password1 != password2:
            flash('Passwords do not match',category='error')
        elif len(password1) < 6:
            flash('Password is too short',category='error')
        else:
            user=User.query.filter_by(username=username).first()
            password = generate_password_hash(password1,method="sha256")
            user.password=password
            db.session.commit()
            login_user(user,remember=True)
            flash('Password changed successfully',category="error")
            return redirect(url_for('views.home'))
    return render_template('forgotpassword.html',user=current_user,logopath=logopath)
```

- The Front-End html file (***forgotpassword.html***) is found under the ***templates*** folder.

6 Navigation Bar

6.1 Overview

- A responsive navigation header which is present in all pages of the Portal.
- Helpful in navigating the different tabs of the Portal, and varies based on type of User logged in.

6.2 Details

- Present in ***base.html***, which is found under the ***templates*** folder.

```
<nav class="navbar sticky-top navbar-expand-lg py-1 shadow" style="background-color:
    <div class="container-fluid">
        <img src="../static/CompanyLogo.png" width="95px" height="25px" >
        <img src = {{logopath}} width="107px" height="35px" style="padding-left:5px;
        <div class="vl py-0" style="border-left: 2px solid white;height: 25px;"></
        <h5 class="nav-item mb-lg-0" style="color: white; padding-left:5px;font-siz

        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                <li class="nav-item">
                    {% if user.is_authenticated %}
                    <a class="nav-link" href ="/home" id="navhome" style="padding-le
                    {% endif %}
                </li>
                    {% if user.is_authenticated %}
                        <a class="nav-link" href ="/all-tickets" id="navalltickets"s
                        <a class="nav-link" href ="/alert-audit" id="navalertaudit"
                    {% endif %}
                    {% if user.usertype=="admin" %}
                        <a class="nav-link" id="navmaster" href ="/master-ticketcode
                    {% endif %}
                </li>
            </ul>
```

7 Home

7.1 Overview

- This functionality is accessible to all users, showing them a visual overview of the Ticket data in the form of pie, bar and line charts.

7.2 Details

- Implemented using the ***chartjs*** Javascript Library.
- This takes place in the functions ***home(), graph_settings()*** and ***piegraph_settings()*** in ***views.py.***

```
@views.route("/")
@views.route("/home")
@login_required
def home():
    yearago = str(datetime.today() - timedelta(days=365)).split(" ")[0]
    now = str(datetime.today()).split(" ")[0]
    assignees = User.query.filter_by(usertype="assignee").all()
    reporters = User.query.filter_by(usertype="reporter").all()
    statuses = Status.query.all()
    tickets_date_created = []
    tickets_date_closed = []
    if(current_user.usertype=="assignee"):
        tickets_closed = Ticket.query.filter(Ticket.assignee_id==current_user.id,Ticket.date_created>yeara
        for ticket in tickets_closed:
            tickets_date_closed.append(str(ticket.date_closed))
        tickets = Ticket.query.filter(Ticket.assignee_id==current_user.id,Ticket.date_created>yearago).ord
        for ticket in tickets:
            tickets_date_created.append(str(ticket.date_created))
```

```
@views.route("/graph-settings",methods=['GET','POST'])
@login_required
def graph_settings():
    yearago = str(datetime.today() - timedelta(days=365)).split(" ")[0]
    assignees = User.query.filter_by(usertype="assignee").all()
    reporters = User.query.filter_by(usertype="reporter").all()
    statuses = Status.query.all()
    count_vs_status = []
    tickets_closed = Ticket.query.filter(Ticket.date_created>yearago).order_by(Ticket.date_created.asc())
    tickets_date_closed = []
    for ticket in tickets_closed:
        tickets_date_closed.append(str(ticket.date_closed))
    for status in statuses :
        count = Ticket.query.filter_by(status = status.status).count()
        stat_tuple = (count,status.status)
        count_vs_status.append(stat_tuple)
```

```
@views.route("/piegraph-settings",methods=['GET','POST'])
@login_required
def piegraph_settings():
    yearago = str(datetime.today() - timedelta(days=365)).split(" ")[0]
    tickets_created = Ticket.query.filter(Ticket.date_created>yearago).order_by(Ticket.date_created.asc())
    tickets_date_created = []
    for ticket in tickets_created:
        tickets_date_created.append(str(ticket.date_created))
    assignees = User.query.filter_by(usertype="assignee").all()
    reporters = User.query.filter_by(usertype="reporter").all()
    statuses = Status.query.all()
    count_vs_status = []
    for status in statuses :
        count = Ticket.query.filter_by(status = status.status).count()
        stat_tuple = (count,status.status)
        count_vs_status.append(stat_tuple)
```

- The Front-End html file (***dashboard.html***) is found under the ***templates*** folder.

8 All Tickets

- This functionality is accessible to all users, showing them all the Tickets relevant to them, in a tabular format, implemented using the ***DataTables*** JQuery Plugin.
- This takes place in the functions ***all_tickets()*** in ***views.py.***

```
@views.route("/")
@views.route("/all-tickets")
@login_required
def all_tickets():
    if current_user.usertype == 'assignee':
        tickets = Ticket.query.filter_by(assignee_id=current_user.id).all()
    elif current_user.usertype == 'reporter':
        tickets = Ticket.query.filter_by(author_id=current_user.id).all()
    else:
        tickets = Ticket.query.all()
    return render_template("all_tickets.html",user=current_user,tickets=tickets,logopath=logopath)
```

- Also includes the Export functionality to export the Ticket Data in Excel Format, takes place in the function ***export_alltickets()*** in ***views.py.***

```
@views.route("/export-alltickets",methods=['GET','POST'])
@login_required
def export_alltickets():
    tickets = Ticket.query.all()
    questions = Question.query.all()
    efforts = Effort.query.all()
    output = io.BytesIO()
    workbook = xlwt.Workbook()
    sh = workbook.add_sheet('All Tickets')
    sh.write(0, 0, 'Ticket Number')
    sh.write(0, 1, 'Customer Name')
    sh.write(0, 2, 'Reporter Name')
    sh.write(0, 3, 'Current Status')
    sh.write(0, 4, 'Assignee Name')
    sh.write(0, 5, 'Requirement Title')
    sh.write(0, 6, 'Region')
    sh.write(0, 7, 'Approx Start Date')
    sh.write(0, 8, 'Created On')
    sh.write(0, 9, 'Assigned On')
    sh.write(0, 10, 'Closed On')
    sh.write(0, 11, 'Last Modified On')
```

- The Front-End html file (***all_tickets.html***) is found under the ***templates*** folder.

8.1 Ticket Creation

- This functionality is only accessible to *Reporter* users, allowing them to create Tickets based on the requirements they have gathered from the customer.
- This takes place in the function ***create_ticket()*** in ***views.py***.

```
@views.route("/create-ticket",methods=['GET','POST'])
@login_required
def create_ticket():
    questions = Question.query.all()
    masteralertconfig = MasterAlertConfig.query.all()
    if(questions and masteralertconfig):
        if request.method == "POST":
            now = time.strftime("%d/%B/%Y %H:%M:%S")
            status= "Opened"
            ticket = Ticket(author_id = current_user.id,status=stat
            custname = request.form.get("custname")
            title = request.form.get("title")
            region = request.form.get("region")
            startdate = request.form.get("startdate")
            ticket.custname = custname
            ticket.title = title
            ticket.region = region
            ticket.startdate = startdate
            ticket.ticket_code = masterticketcode
            db.session.add(ticket)
            if(custname and title and startdate and region):
                db.session.commit()
```

- The Front-End html file (***create_ticket.html***) is found under the ***templates*** folder.

9 Current Ticket

- This functionality is accessible to all users, displaying all the details regarding a specific Ticket on one page, in the function ***tickets()*** in ***views.py***.
- The Front-End html file (***current_ticket.html***) is found under the ***templates*** folder.
- There are multiple functionalities, which are accessible from the Front-End via this page. Their corresponding functions from ***views.py*** are given below:
    1. Assign Assignee (Only for *Admin* users): ***assign_assignee()***
    2. Effort Estimation (Only for *Assignee* users): ***estimated_details()***
    3. Update Status (Only for *Admin* and *Assignee* users): ***update_status()***
    4. Edit Ticket (For all users): ***edit_ticket()***
    5. Reopen Ticket (Only for *Admin* and *Assignee* users): ***reopen_ticket()***

```python
@views.route("/reopen-ticket/<ticket_id>",methods=['GET'])
@login_required
def reopen_ticket(ticket_id):
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    ticket.status = "Opened"
    ticket.date_closed = None
    db.session.commit()
    return redirect('/tickets/'+str(ticket_id))
```

6. Delete Ticket (Only for *Reporter* users): ***delete_ticket()***

```python
@views.route("/delete-ticket/<id>")
@login_required
def delete_ticket(id):
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    ticket = Ticket.query.filter_by(id=id).first()

    if not ticket:
        flash("Ticket does not exist", category='error')
    elif current_user.id != ticket.author_id:
        flash("You do not have permission to delete this ticket",category='error')
    else:
        ticket.assignee.status = "Available"
        ticket.last_modified = now
        db.session.delete(ticket)
        db.session.commit()
        flash('Ticket Deleted',category='success')
    return redirect(url_for('views.home'))
```

7. Comment Section:
    - Add Comment: ***create_comment()***

```python
@views.route("create-comment/<ticket_id>",methods=['POST'])
@login_required
def create_comment(ticket_id):
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    text = request.form.get('text')

    if not text:
        flash("Comment cannot be empty",category='error')
    else:
        if ticket:
            comment = Comment(text=text,author=current_user.id,ticket_id=ticket_id,date_created=now)
            db.session.add(comment)
            ticket.last_modified = now
            db.session.commit()
        else:
            flash("Post does not exist",category='error')

    return redirect('/tickets/'+str(ticket_id))
```

- Edit Comment : ***edit_comment()***

```
@views.route("/edit-comment/<comment_id>",methods=['POST'])
@login_required
def edit_comment(comment_id):
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    comment = Comment.query.filter_by(id=comment_id).first()
    ticket_id = comment.ticket_id
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    if not comment:
        flash("Comment does not exist", category='error')
    elif current_user.id != comment.author:
        flash("You do not have permission to update status",category='error')
    else:
        if request.method == "POST":
            output = request.get_json()
            result = json.loads(output)
            newtext = str(result["newtext"])
            comment.text = newtext
            comment.status = "Edited"
            ticket.last_modified = now
            db.session.commit()
    return redirect('/tickets/'+str(ticket_id))
```

- Delete Comment : ***delete_comment()***

```
@views.route("/delete-comment/<comment_id>")
@login_required
def delete_comment(comment_id):
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    comment = Comment.query.filter_by(id=comment_id).first()
    ticket_id = comment.ticket_id
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    if not comment:
       flash("Comment does not exist", category='error')
    elif current_user.id != comment.author:
        flash("You are not authorized to delete this comment", category='error')
    else:
        comment.status = "Deleted"
        ticket.last_modified = now
        db.session.commit()

    return redirect('/tickets/'+str(ticket_id))
```

8. Attachments Section:
   - Add Attachment: ***attach_file()***

```
@views.route("/attach-file/<ticket_id>",methods=['POST'])
@login_required
def attach_file(ticket_id):
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    upload = request.files['upload']
    if not upload:
        return 'No pic uploaded!', 400

    filename = secure_filename(upload.filename)
    mimetype =upload.mimetype
    if not filename or not mimetype:
        return 'Bad upload!', 400

    file = File(data=upload.read(), name=filename, mimetype=mimetype,ticket_id=ticket_id)
    db.session.add(file)
    ticket.last_modified = now
    db.session.commit()
    return redirect('/tickets/'+str(ticket_id))
```

- Download Attachment: ***download_file()***

```
@views.route("/download-file/<file_id>",methods=['GET'])
@login_required
def download_file(file_id):
    file = File.query.filter_by(id=file_id).first()
    ticket_id = file.ticket_id
    if(file):
        return send_file(io.BytesIO(file.data),attachment_filename=file.name,as_attachment=True)
    else:
        flash("File not found",category="error")
        return redirect('/tickets/'+str(ticket_id))
```

- Delete Attachment: ***delete_file()***

```
@views.route("/delete-file/<file_id>",methods=['GET'])
@login_required
def delete_file(file_id):
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    file = File.query.filter_by(id=file_id).first()
    ticket_id = file.ticket_id
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    db.session.delete(file)
    ticket.last_modified = now
    db.session.commit()
    return redirect('/tickets/'+str(ticket_id))
```

- View Attachment (Only for images): Processed in ***proc_files()*** and displayed using ***display_image()*** on ***display_image.html*** (under ***templates*** folder)

```
def proc_files(files_raw,ticket_id):
    files_list=[]
    for file_raw in files_raw:
        if(str(file_raw.ticket_id) == str(ticket_id) and (file_raw.mimetype
            npimg = np.fromstring(file_raw.data, np.uint8)
            img = cv2.imdecode(npimg,cv2.IMREAD_COLOR)
            img = Image.fromarray(img.astype("uint8"))
            rawBytes = io.BytesIO()
            img.save(rawBytes, "JPEG")
            rawBytes.seek(0)
            img_base64 = base64.b64encode(rawBytes.read())
            img_base64 = str(img_base64).lstrip('b')
            img_base64 = str(img_base64).strip("'")
            imgtuple = (img_base64,file_raw.id,file_raw.mimetype)
            files_list.append(imgtuple)
        else:
            filetuple = (file_raw.name,file_raw.id,file_raw.mimetype)
            files_list.append(filetuple)
    return files_list
```

```
@views.route("/display-image/<file_id>",methods=['GET'])
@login_required
def display_image(file_id):
    file = File.query.filter_by(id=file_id).first()
    npimg = np.fromstring(file.data, np.uint8)
    img = cv2.imdecode(npimg,cv2.IMREAD_COLOR)
    img = Image.fromarray(img.astype("uint8"))
    rawBytes = io.BytesIO()
    img.save(rawBytes, "JPEG")
    rawBytes.seek(0)
    img_base64 = base64.b64encode(rawBytes.read())
    img_base64 = str(img_base64).lstrip('b')
    img_base64 = str(img_base64).strip("'")
    return render_template("display_image.html",user=current_user,logopath=logopath,imgsrc=img_base64)
```

9.1 Assign Assignee

- This functionality is only accessible to *Admin* users, allowing them to assign a Ticket to a specific *Assignee*/Consultant based on their availability.
- This takes place in the function ***assign_assignee()*** in ***views.py.***

```
@views.route("/assign-assignee/<ticket_id>",methods=['GET','POST'])
@login_required
def assign_assignee(ticket_id):
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    assignees = User.query.filter_by(usertype='assignee').all()
    if not ticket:
        flash("Ticket does not exist", category='error')
    elif current_user.usertype != 'admin':
        flash("You do not have permission to assign assignee",category='error')
    else:
        if request.method == "POST":
            if(ticket.assignee_id):
                curr_assignee_id = ticket.assignee_id
                curr_assignee = User.query.filter_by(id=curr_assignee_id).first()
                curr_assignee.status="Available"
            output = request.get_json()
            result = json.loads(output)
            assignee_name = str(result)
            assignee = User.query.filter_by(username=assignee_name).first()
```

- On the Front-End, this is a part of the Current Ticket page.

9.2 Effort Estimation

- This functionality is only accessible to *Assignee* users, allowing them to update the results of their Effort Estimation on the portal.
- This takes place in the function ***estimated_details()*** in ***views.py.***

```
@views.route("/estimated-details/<ticket_id>",methods=['GET','POST'])
@login_required
def estimated_details(ticket_id):
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    efforts = Effort.query.order_by(Effort.id.asc()).all()
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    if(TicketEffortMap.query.filter_by(ticket_id=ticket_id).first()):
        reason = "So that duplicate mapping not created"
    else:
        for effort in efforts:
            ticketeffortmap = TicketEffortMap(ticket_id=ticket_id,effort_id=effort.id)
            db.session.add(ticketeffortmap)
        db.session.commit()
    ticketeffortmaps = TicketEffortMap.query.filter_by(ticket_id=ticket_id).all()
    if request.method == "POST":
        if(efforts):
            for effort in efforts:
                answer = request.form.get("e"+str(effort.id))
                map = TicketEffortMap.query.filter_by(ticket_id=ticket.id,effort_id=effort.id).first()
                map.value = str(answer)
            comment = Comment(text="Effort Estimation Details Updated",author=current_user.id,ticket_i
            db.session.add(comment)
            ticket.last_modified = now
            db.session.commit()
```

- The Front-End html file (***estimate_details.html***) is found under the ***templates*** folder.

9.3 Update Status

- This functionality is accessible to *Admin* and *Assignee* users, allowing them to update the current status of a Ticket.
- This takes place in the function ***update_status()*** in ***views.py.***

```
@views.route("/update-status/<ticket_id>",methods=['GET','POST'])
@login_required
def update_status(ticket_id):
    ticketeffortmap = TicketEffortMap.query.filter_by(ticket_id=ticket_id).all()
    statuses = Status.query.all()
    if(statuses):
        pass
    else:
        flash("Please complete Master Setup first",category="error")
        return render_template("/tickets/"+ticket.id)
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    oldstatus = ticket.status
    if not ticket:
        flash("Ticket does not exist", category='error')
    elif current_user.usertype == 'reporter':
        flash("You do not have permission to update status",category='error')
    elif current_user.id != ticket.assignee_id and current_user.usertype != 'admin':
```

- On the Front-End, this is a part of the Current Ticket page

9.4 Edit Ticket

- This functionality accessible to all users, allowing them to edit different fields of a Ticket, depending on their permissions.
- This takes place in the function ***edit_ticket()*** in ***views.py.***

```
@views.route("/edit-ticket/<ticket_id>",methods=['GET','POST'])
@login_required
def edit_ticket(ticket_id):
    questions = Question.query.order_by(Question.id.asc()).all()
    statuses = Status.query.order_by(Status.id.asc()).all()
    efforts = Effort.query.order_by(Effort.id.asc()).all()
    ticketquestionmaps = TicketQuestionMap.query.filter_by(ticket_id=ticket_id).all()
    ticketeffortmaps = TicketEffortMap.query.filter_by(ticket_id=ticket_id).all()
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    ticket = Ticket.query.filter_by(id=ticket_id).first()
    assignees = User.query.filter_by(usertype='assignee').all()
    if not ticket:
        flash("Ticket does not exist", category='error')
    else:
        if request.method == "POST":
            if(current_user.usertype=='reporter'):
                custname = request.form.get('custname')
                title = request.form.get('title')
                region = request.form.get('region')
                startdate = request.form.get('startdate')
                ticket.custname=custname
```

- The Front-End html file (***edit_ticket.html***) is found under the ***templates*** folder.

10 Alert Mechanism

- This functionality enables the portal to send out automated alerts to users via Email, implemented using SMTP.
- This takes place in the functions ***alertmechanism() and emailbysmtp()*** in ***views.py.***

```
def alertmechanism(ticket_status, ticket_id):
    current = MasterAlertConfig.query.filter_by(ticket_status=ticket_status).first()
    if(current):
        ticket = Ticket.query.filter_by(id = ticket_id).first()
        ticket_code = ticket.ticket_code
        admins = User.query.filter_by(usertype = "admin").all()
        reporter_id = ticket.author_id
        assignee_id = ticket.assignee_id
        reporter = User.query.filter_by(id = reporter_id).first()
        reporter_email = reporter.email
        assignee = User.query.filter_by(id = assignee_id).first()
        if(assignee):
            assignee_email = assignee.email
        else:
            assignee_email = ""
        alertsub =  current.alert_subject
        alertbody = current.alert_body
        alertbody = alertbody.replace("<ticket_id>",str(ticket_code)+str(ticket_id))
        alertbody = alertbody.replace("<ticket_status>",str(ticket_status))
        body_type = current.body_type
```

```
def emailbysmtp(recipients_email , alertsub , alertbody , body_type):
    arr = recipients_email.split(";")
    sender_email = ''+str(config['email']['sendermail'])+''
    password = ''+str(config['email']['senderpwd'])+''
    for i in arr :
        receiver_email = str(i)
        domain = receiver_email.split("@")[1]
        if(domain == "adobe.com"):
            message = MIMEMultipart("alternative")
            message["Subject"] = alertsub
            message["From"] = sender_email
            message["To"] = receiver_email
            if(body_type == 'HTML'):
                html = alertbody
                part1 = MIMEText(html, "html")
                message.attach(part1)
```

- The history of alerts sent is stored in the database and displayed in the Alert Audit Table.

10.1 Alert Audit

- This functionality is accessible to all users, showing them the Alert Audit, in a tabular format, implemented using the ***DataTables*** JQuery Plugin.
- This takes place in the functions ***all_tickets()*** in ***views.py.***

```
@views.route("/")
@views.route("/all-tickets")
@login_required
def all_tickets():
    if current_user.usertype == 'assignee':
        tickets = Ticket.query.filter_by(assignee_id=current_user.id).all()
    elif current_user.usertype == 'reporter':
        tickets = Ticket.query.filter_by(author_id=current_user.id).all()
    else:
        tickets = Ticket.query.all()
    return render_template("all_tickets.html",user=current_user,tickets=tickets,logopath=logopath)
```

- Also includes the Export functionality to export the Audit data in Excel Format, takes place in the function ***export_audit()*** in ***views.py.***

```
@views.route("/export-audit",methods=['GET','POST'])
@login_required
def export_audit():
    audits = MasterAlertAudit.query.all()
    output = io.BytesIO()
    workbook = xlwt.Workbook()
    sh = workbook.add_sheet('Alert Audit')
    sh.write(0, 0, 'Id')
    sh.write(0, 1, 'Ticket Status')
    sh.write(0, 2, 'Alert Subject')
    sh.write(0, 3, 'Alert Body')
    sh.write(0, 4, 'Recipients')
    sh.write(0, 5, 'Sent On')

    idx = 0
    for audit in audits:
        sh.write(idx+1, 0, str(audit.id))
        sh.write(idx+1, 1, audit.ticket_status)
        sh.write(idx+1, 2, audit.alert_subject)
```

- The Front-End html file (***alert_audit.html***) is found under the ***templates*** folder.

11 Master Setup

- This functionality is only accessible to *Admin* users, allowing them to setup the Master Data for their specific team and use-case. The types of Master Data and their corresponding functions in ***views.py***:
  1. Ticket Code – ***master_ticketcode_home()***

```
@views.route("/master-ticketcode-home",methods=['GET','POST'])
@login_required
def master_ticketcode_home():
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    if(Status.query.all()):
        pass
    else:
        status1 = Status(status="Opened",date_created=now)
        status2 = Status(status="Assigned",date_created=now)
        status3 = Status(status="In-Review",date_created=now)
        status4 = Status(status="Closed",date_created=now)
        status5 = Status(status="Cancelled",date_created=now)
        db.session.add(status1)
        db.session.add(status2)
        db.session.add(status3)
        db.session.add(status4)
        db.session.add(status5)
```

  2. Ticket Status – ***master_status_home()***

```
@views.route("/master-status",methods=['POST'])
@login_required
def master_status():
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    if request.method == "POST":
        output = request.get_json()
        for ele in output:
            if(str(ele["value"])!=""):
                status = Status(status=str(ele["value"]),date_created=now,author_id=current_user.id)
                db.session.add(status)
                db.session.commit()
            else:
                flash("Please fill status field",category="error")
                return redirect('/master-status-home')
    return redirect('/master-status-home')
```

  3. Questionnaire – ***master_question_home()***

```
@views.route("/master-question",methods=['POST'])
@login_required
def master_question():
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    if request.method == "POST":
        output = request.get_json()
        for ele in output:
            if(str(ele["value"])!=""):
                question = Question(question=str(ele["value"]),date_created=now,author_id=current_user.id)
                db.session.add(question)
            else:
                flash("Please fill question field",category="error")
            db.session.commit()
    return redirect('/master')
```

4. Effort Estimation Fields – ***master_effort_home()***

```
@views.route("/master-effort",methods=['POST'])
@login_required
def master_effort():
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    if request.method == "POST":
        output = request.get_json()
        for ele in output:
            if(str(ele["value"])!=""):
                effort = Effort(effort=str(ele["value"]),date_created=now,author_id=current_user.id)
                db.session.add(effort)
            db.session.commit()
    return redirect('/master')
```

5. Alert Configuration – ***master_alert_home()***

```
@views.route("/master-alert-home",methods=['GET','POST'])
@login_required
def master_alert_home():
    alerts = MasterAlertConfig.query.all()
    statuses = Status.query.all()
    if request.method == 'POST':
        ticket_status = request.form.get('ticket_status')
        alert_subject = request.form.get('alert_subject')
        alert_body = request.form.get('alert_body')
        body_type = request.form['body_type']
        recipients = request.form.get('recipients')
        if(len(alert_subject)>500):
            flash("Alert Subject exceeded 500 characters",category="error")
            return redirect('/master-alert-home')
        curr = MasterAlertConfig.query.filter_by(ticket_status=ticket_status).first()
        if(curr):
            curr.alert_subject = alert_subject
            curr.alert_body = alert_body
            curr.body_type = body_type
            curr.recipients = recipients
```

6. Logo Upload – ***master_logo_home()***

```
@views.route("/master-logo",methods=['POST'])
@login_required
def master_logo():
    target = os.path.join(APP_ROOT, 'static/')

    if not os.path.isdir(target):
        os.mkdir(target)

    file = request.files["file"]
    filename = "logo.png"
    destination = "/".join([target, filename])
    file.save(destination)
    flash("Logo Uploaded",category="succeess")
    return render_template('master_logo.html',user=current_user,logopath=logopath)
```

7. Reset Database – ***master_reset_home()***

```
@views.route("/master-reset",methods=['GET'])
@login_required
def master_reset():
    now = time.strftime("%d/%B/%Y %H:%M:%S")
    MasterAlertConfig.query.delete()
    MasterAlertAudit.query.delete()
    Comment.query.delete()
    File.query.delete()
    TicketEffortMap.query.delete()
    TicketQuestionMap.query.delete()
    Effort.query.delete()
    Question.query.delete()
    Status.query.delete()
    Ticket.query.delete()
    User.query.filter_by(usertype="reporter").delete()
    User.query.filter_by(usertype="assignee").delete()
    newreset = MasterResetHistory(reset_by=current_user.id,reset_on=now)
    db.session.add(newreset)
    db.session.commit()
    return redirect('/master-reset-home')
```

- The Front-End html files (***master_ticketcode.html, master_status.html, master_question.html, master_effort.html, master_alert.html, master_logo.html, master_reset.html***) are found under the ***templates*** folder.

12 Additional Notes

- All .html pages extend the ***base.html*** template, which contains the Navigation Bar.
- The ***config.ini*** file contains the basic configuration details for Database connection, Email Delivery credentials and path for logo image.
- The logo image file is stored in the ***static*** folder and can be uploaded from the Master Setup.