

STAR APP SOLUTION

A Project

REPORT

On

ONLINE EXAMINATION SYSTEM

Submitted by

YASH TANWAR

Of

TEAM - B

Under the Guidance of

PREETI NEGI & SEJAL NARWAT MAM

Submitted on

MARCH 2025

Chapter 1. Introduction

1.1 Introduction

The Online Examination System is a web-based platform developed using Flask and PostgreSQL that enables efficient test management, user authentication, and evaluation. It allows users to register, take tests, and track their progress while administrators can manage tests, users, and reports.

1.2 Objectives

- Provide a seamless online test-taking experience.
- Implement user authentication with OTP verification.
- Allow administrators to manage users, test types, and test questions.
- Enable real-time evaluation and reporting of test results.
- Support file uploads for profile pictures and test questions.

1.3 Technologies Used

- **Backend:** Flask (Python)
- **Frontend:** HTML, CSS, JavaScript
- **Database:** PostgreSQL
- **Authentication:** Flask-Login, Flask-Mail
- **File Handling:** Werkzeug
- **Styling:** Tailwind CSS

Chapter 2. System Design & Analysis

2.1 System Design

User Authentication

- Registration with profile picture upload.
- OTP verification via email.
- Secure login using password hashing.
- Logout functionality.

User Management

- List and search users.
- Edit user details.
- Delete user accounts.
- Add new users manually.

Test Type Management

- Add, edit, and delete test types.
- Search test types.

Test Master Management

- Add test questions, including images.
- Store multiple-choice questions and correct answers.
- Delete test questions.

Test Allocation

- Assign specific tests to users.
- Remove allocated tests.

Test Attempt & Evaluation

- Users can take tests assigned to them.
- Fetch questions based on test type.
- Submit test results and auto-evaluate scores.

Reports & Statistics

- View the total number of students.

- Display test attempts and results (Passed/Failed counts).
- Generate reports based on user performance.

2.2 System Analysis

Database Schema

- **User Table:** Stores user details, passwords, and OTP status.
- **TestType Table:** Stores test category information.
- **TestMaster Table:** Stores questions and options.
- **AllocatedTest Table:** Tracks test assignments to users.
- **ExamAttempt Table:** Stores users' test attempts and scores.

System Workflow

1. A new user registers and uploads a profile picture.
2. The system generates an OTP and sends it via email.
3. The user verifies their account and logs in.
4. An administrator assigns a test type to the user.
5. The user selects a test and answers questions.
6. The system evaluates the test and stores the result.
7. Administrators generate reports based on test performance.

Conclusion

The Online Examination System provides a robust and secure platform for conducting online tests efficiently. It simplifies the management of users and test-related data while ensuring smooth user interactions through authentication, test allocation, and performance tracking.

Future Enhancements

- Implement AI-based question generation.
- Add real-time test monitoring.
- Introduce a timer-based test module.
- Provide detailed analytics on test performance.

Chapter 3. Implementations & Results

Registration Route

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        full_name = request.form['full_name']
        email = request.form['email']
        password = request.form['password']
        confirm_password = request.form['confirm_password']
        phone = request.form['phone']
        file = request.files['profile_picture']

        # Check if email already exists
        existing_user = User.query.filter_by(email=email).first()
        if existing_user:
            flash('Email already registered. Please login.', 'danger')
            return redirect(url_for('login'))

        if password != confirm_password:
            flash('Passwords do not match!', 'danger')
            return redirect(url_for('register'))

        hashed_password = generate_password_hash(password)
        otp = str(random.randint(100000, 999999))

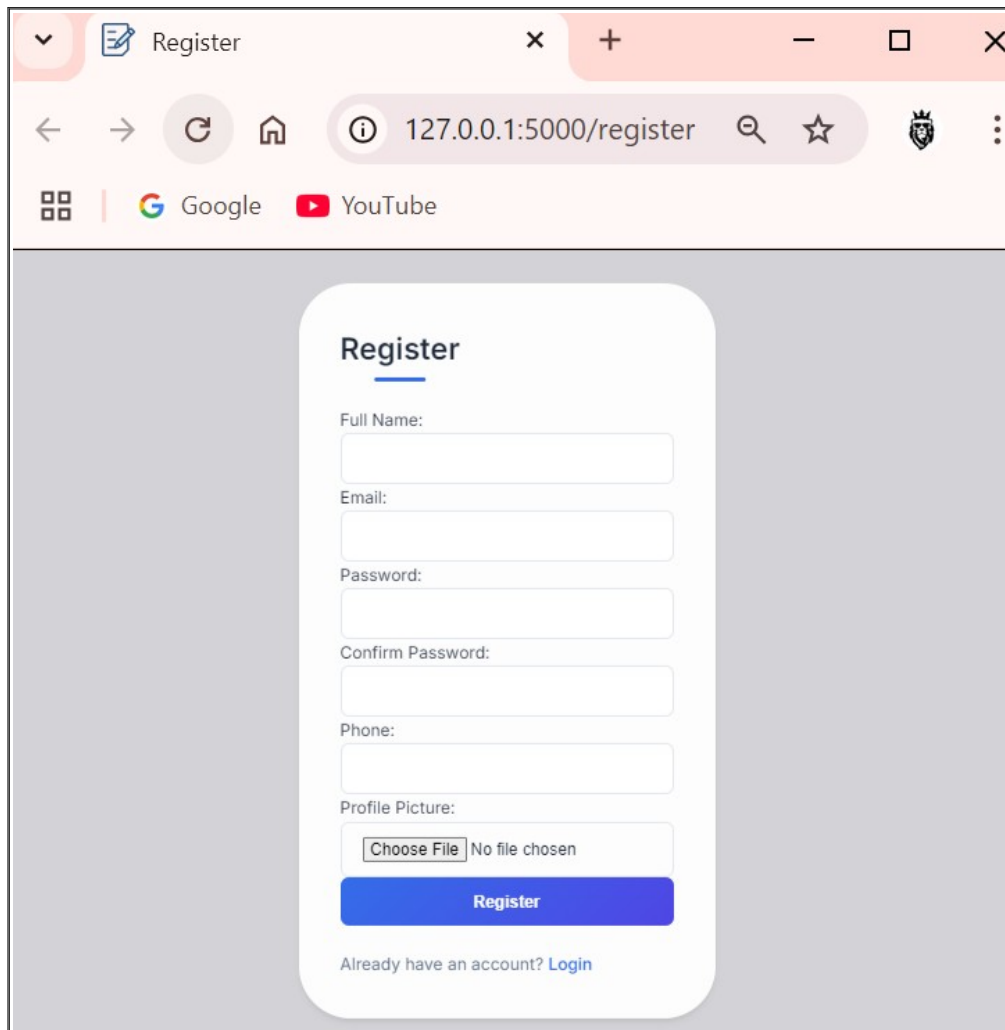
        # Handle profile picture upload
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            file.save(filepath)
        else:
            filename = 'default.png' # Default profile picture

        new_user = User(
            full_name=full_name,
            email=email,
            phone=phone,
            profile_picture=filename,
            otp=otp,
            is_verified=False
        )
        new_user.set_password(password)

        db.session.add(new_user)
        db.session.commit()

        send_otp(email, otp)
        flash('Registration successful! Please login and verify your email.', 'success')
        return redirect(url_for('login'))
```

```
return render_template('register.html')
```



The screenshot displays a web browser window with a single tab titled "Register". The address bar shows the URL "127.0.0.1:5000/register". Below the address bar, there are search engines for Google and YouTube. The main content area contains a white registration form with the following fields and elements:

- Full Name:** A text input field.
- Email:** A text input field.
- Password:** A text input field.
- Confirm Password:** A text input field.
- Phone:** A text input field.
- Profile Picture:** A file upload section with a "Choose File" button and the text "No file chosen".
- Register:** A prominent blue button.
- Already have an account? [Login](#)**: A link at the bottom of the form.

Fig 3.1.1 Registration Page

Login Route

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = User.query.filter_by(email=email).first()

        if user and check_password_hash(user.password_hash, password):
            if not user.is_verified:
                return redirect(url_for('verify', email=email))
            session['user_id'] = user.id
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid credentials!', 'danger')
    return render_template('login.html')
```

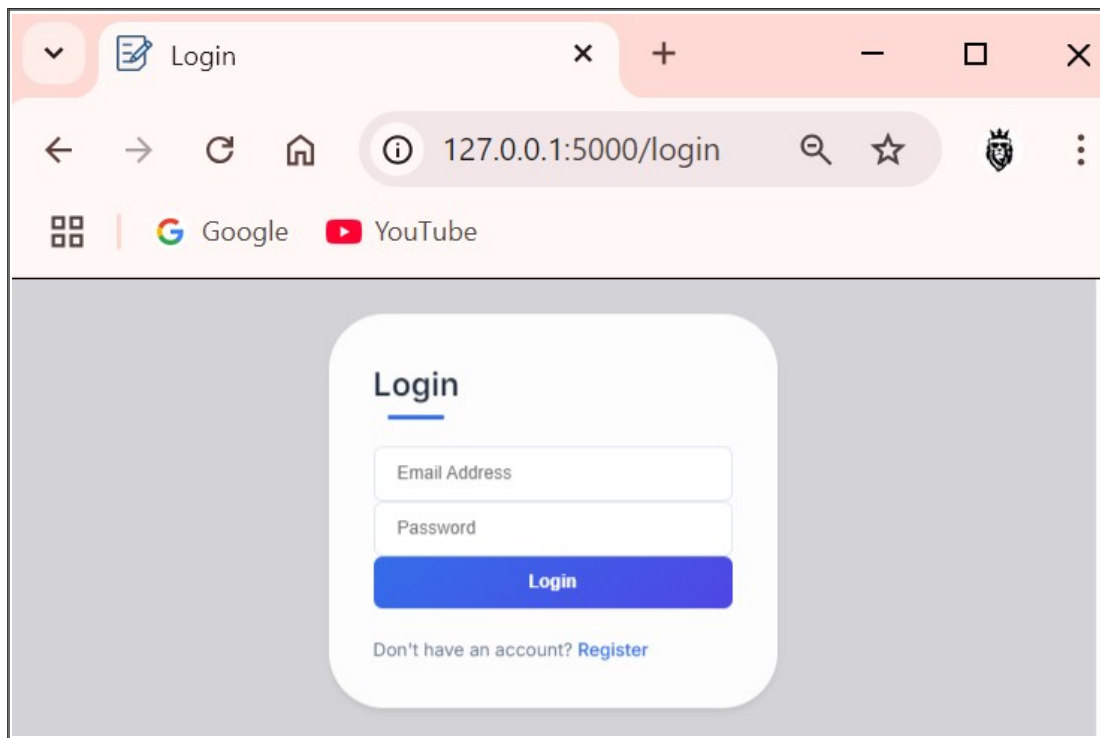


Fig 3.1.2 Login Page

Verify OTP Route

```
@app.route('/verify', methods=['GET', 'POST'])
def verify():
    email = request.args.get('email')
    user = User.query.filter_by(email=email).first()

    if not user:
        flash("User not found!", "danger")
        return redirect(url_for('register'))

    if request.method == 'POST':
        entered_otp = request.form.get('otp')
        if user.otp == entered_otp:
            user.is_verified = True
            db.session.commit()
            session['user_id'] = user.id
            flash("Verification successful! Welcome to your dashboard.", "success")
            return redirect(url_for('dashboard'))
        else:
            flash("Invalid OTP. Please try again.", "danger")
            return redirect(url_for('verify', email=email))

    return render_template('verify.html', email=email)
```

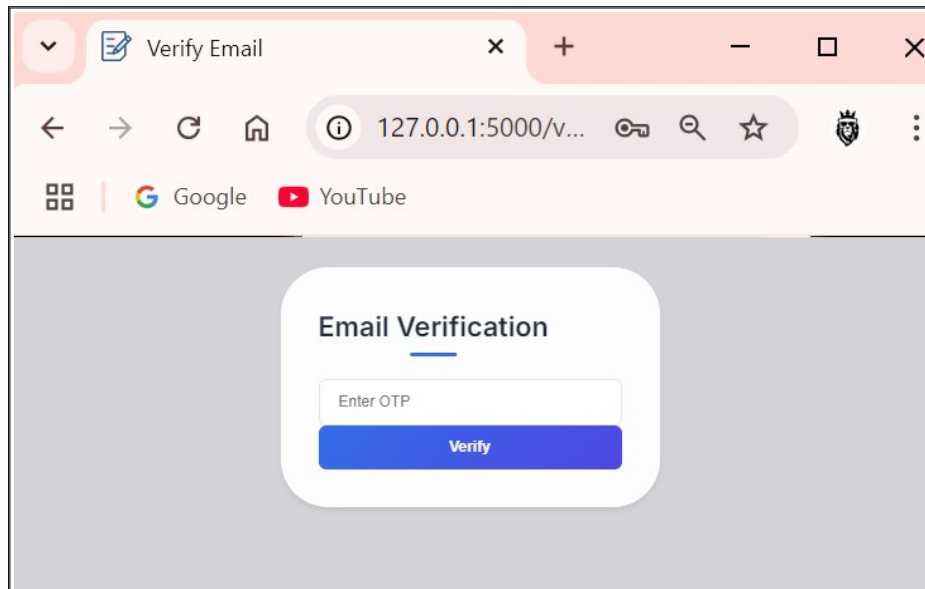


Fig 3.1.3 Verification Page

Dashboard Route

```
@app.route('/dashboard')
def dashboard():
    if 'user_id' not in session:
        return redirect(url_for('login'))

    user = User.query.get(session['user_id'])
    if not user:
        flash("User not found!", "danger")
        return redirect(url_for('login'))

    # Dashboard statistics
    total_students = User.query.count()
    new_students = User.query.filter(User.is_verified == True).count()
    test_types = test_types = TestType.query.count()
    total_attempts = ExamAttempt.query.count()
    passed_attempts = ExamAttempt.query.filter_by(status='Passed').count()
    failed_attempts = ExamAttempt.query.filter_by(status='Failed').count()

    return render_template('dashboard.html',
        user=user,
        total_students=total_students,
        new_students=new_students,
        test_types=test_types,
        total_attempts=total_attempts,
        passed_attempts=passed_attempts,
        failed_attempts=failed_attempts
    )
```

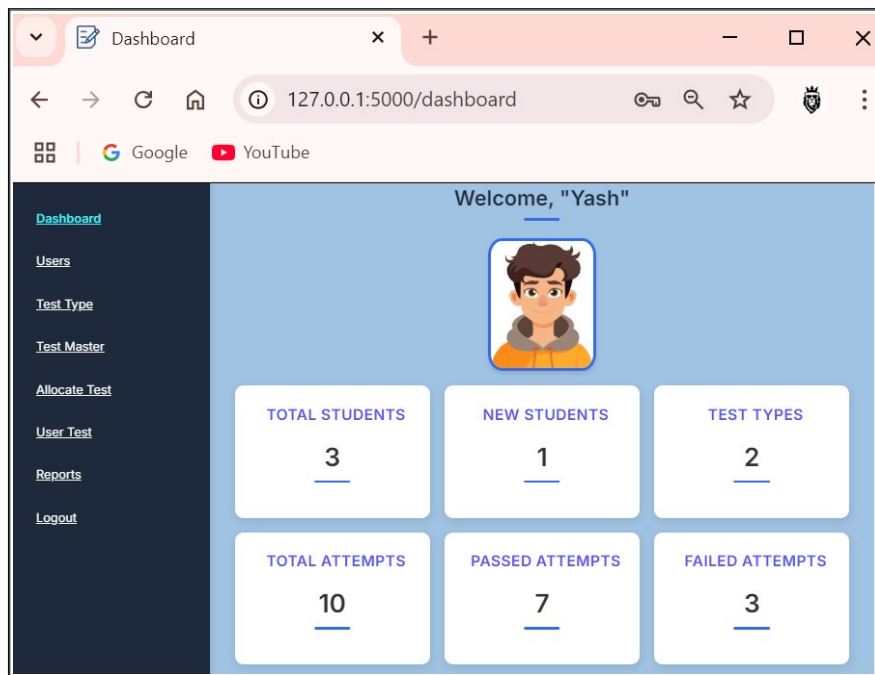



Fig 3.1.4 Dashboard Page

Users List & Search Route

```
@app.route('/users', methods=['GET', 'POST'])
def users():
    search_query = request.form.get('search', "") # Get search input
    if search_query:
        users = User.query.filter(User.full_name.ilike(f"%{search_query}%")).all()
    else:
        users = User.query.all()
    return render_template('users.html', users=users)
```

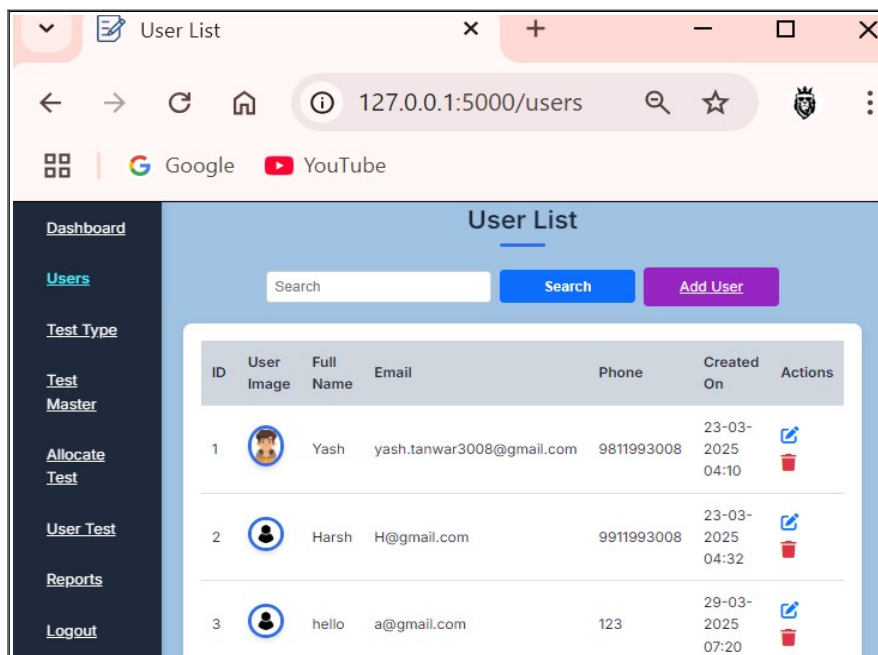


Fig 3.1.5 User Page

#Test Type Route

```
@app.route('/test-type', methods=['GET', 'POST'])
def test_type():
    if request.method == 'POST':
        search_query = request.form.get('search', "").strip() # Get search input

        if search_query:
            test_types = TestType.query.filter(TestType.test_type.ilike(f'{search_query}%')).all()
        else:
            test_types = TestType.query.all()
    else:
        test_types = TestType.query.all()

    return render_template('test_type.html', test_types=test_types)
```

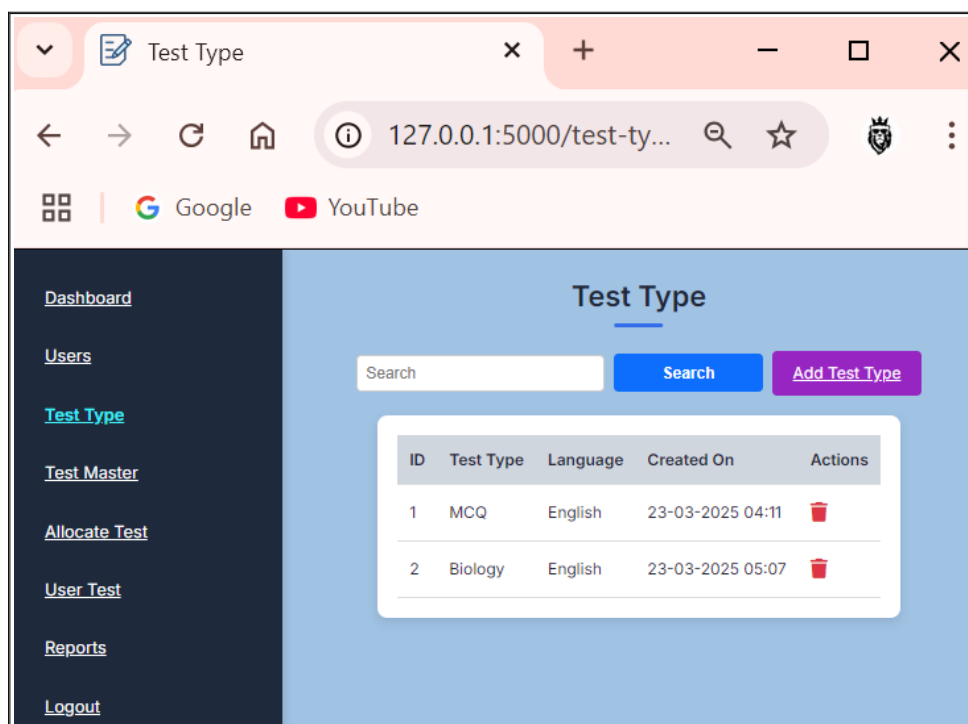


Fig 3.1.6 Test Type Page

Test Master Route

```
UPLOAD_FOLDER = 'static/uploads/questions'
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

@app.route('/test-master', methods=['GET', 'POST'])
def test_master():
    if request.method == 'POST':
        search_query = request.form.get('search', "").strip()
        test_master = TestMaster.query.filter(TestMaster.question.ilike(f'{search_query}%')).all()
    else:
        test_master = TestMaster.query.all()
    return render_template('test_master.html', test_master=test_master)
```

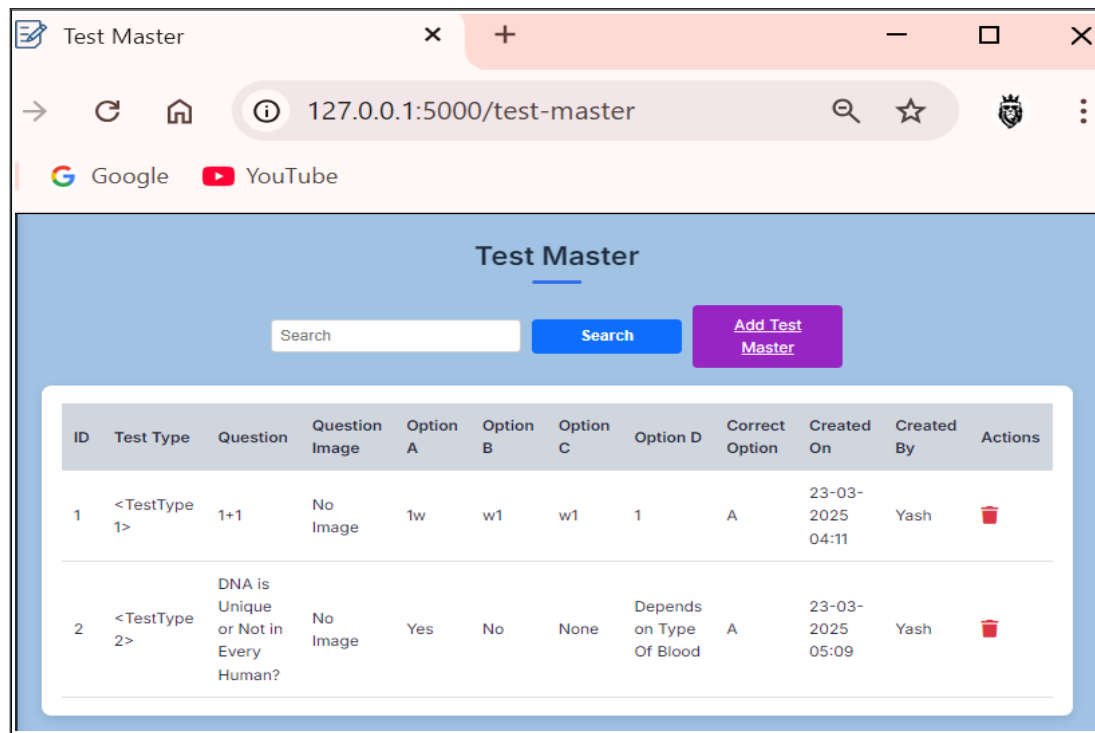


Fig 3.1.7 Test Master Page

Allocate Test Route

```
@app.route('/allocate_test', methods=['GET', 'POST'])
def allocate_test():
    if request.method == 'POST':
        user_id = request.form.get('user_id')
        test_type_id = request.form.get('test_type_id')

        if not user_id or not test_type_id:
            flash('Please select both user and test type.', 'error')
            return redirect(url_for('allocate_test'))

        allocation = AllocatedTest(user_id=user_id, test_type_id=test_type_id)
        db.session.add(allocation)
        db.session.commit()
        flash('Test allocated successfully!', 'success')
        return redirect(url_for('allocate_test'))

    users = User.query.all()
    test_types = TestType.query.all()
    allocated_tests = AllocatedTest.query.order_by(AllocatedTest.assigned_at.desc()).all()

    return render_template('allocate_test.html', users=users, test_types=test_types,
        allocated_tests=allocated_tests)
```

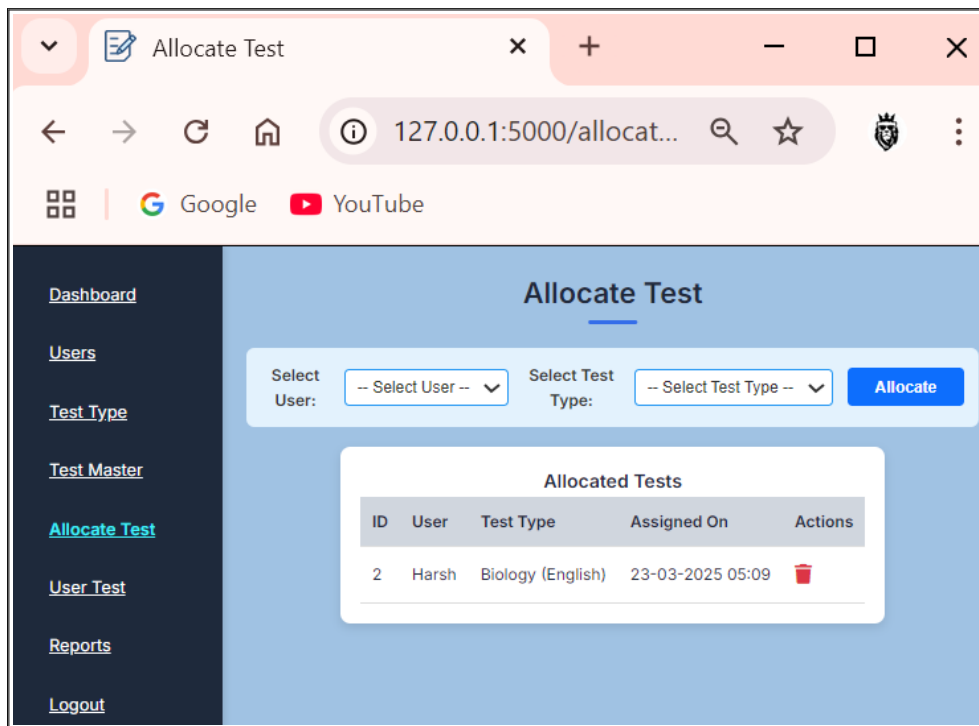


Fig 3.1.8 Allocate Test Page

User Test Route

```
@app.route('/user_test')
def user_test():
    test_types = TestType.query.all()
    return render_template('user_test.html', test_types=test_types)
```

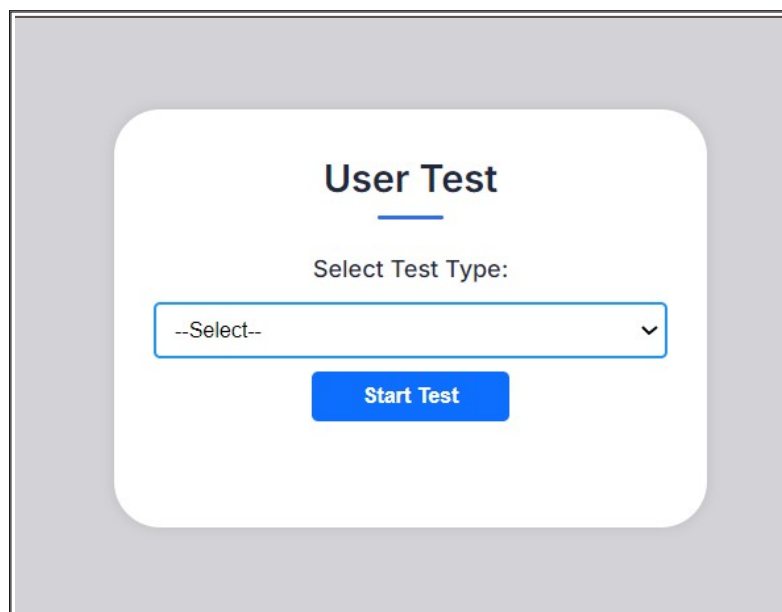


Fig 3.1.9 User Test Page

#Reports Route

```
@app.route('/reports')
```

```
def reports():
```

```
    exam_attempts = ExamAttempt.query.join(User).join(TestType).all()
```

```
    total_students = User.query.count()
```

```
    pass_count = ExamAttempt.query.filter_by(status='Passed').count()
```

```
    fail_count = ExamAttempt.query.filter_by(status='Failed').count()
```

```
    return render_template('reports.html',  
                           exam_attempts=exam_attempts,  
                           total_students=total_students,  
                           pass_count=pass_count,  
                           fail_count=fail_count)
```

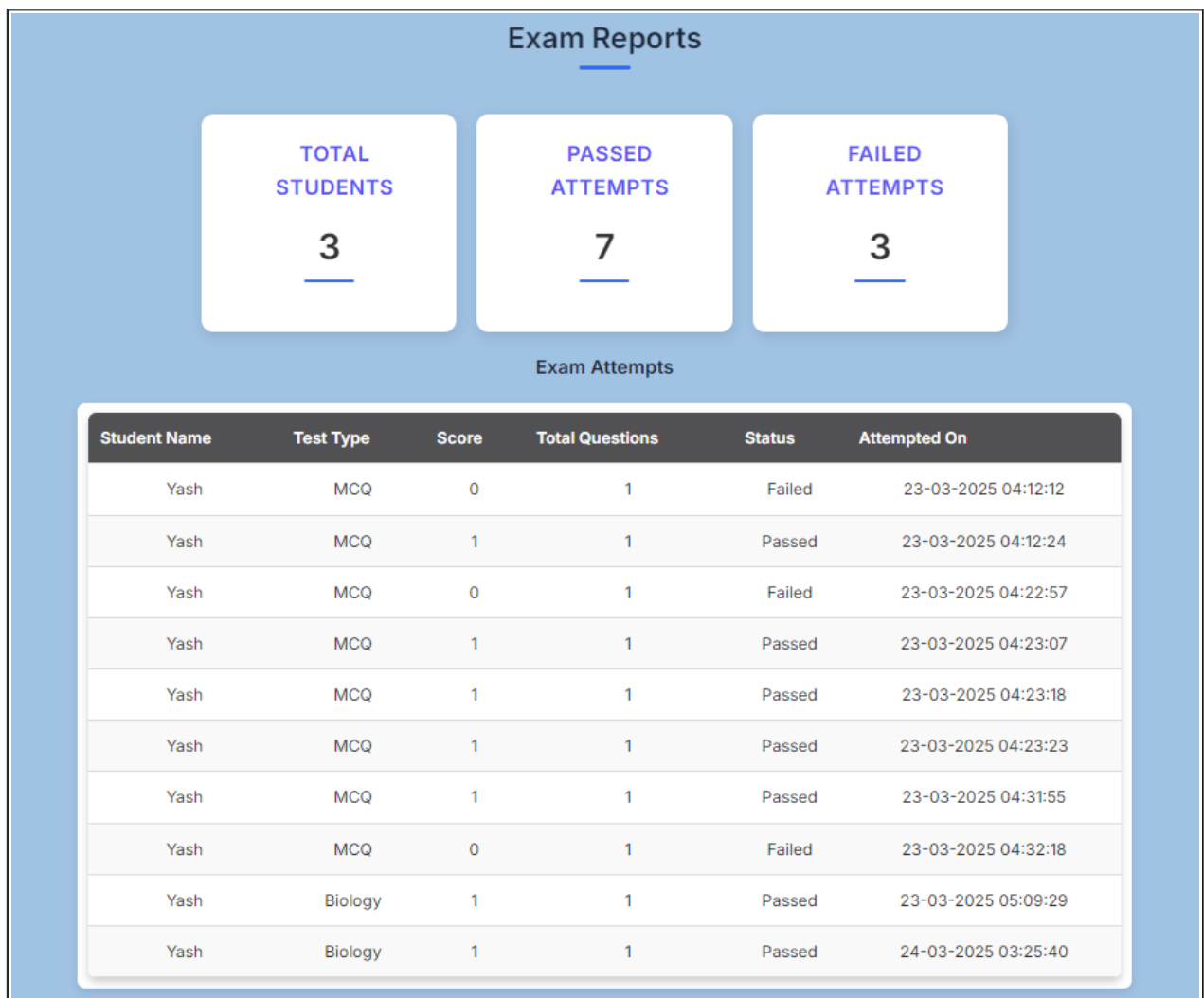


Fig 3.1.10 Reports Page

Chapter 4. Errors Faced During Project Development

1. Database Errors : Missing or incorrect column names during queries, often caused by model updates without running migrations.

Solution: Ensured all column name is executed after model changes.

2. SMTP Errors SMTP Exception: Failed email delivery for OTP due to incorrect email credentials or blocked ports.

Solution: Verified SMTP server, port settings, and correct Gmail authentication settings.

3. Login Issues Invalid Email or Password : Users faced login failures even with valid credentials due to bcrypt hash mismatch.

Solution: Confirmed password hashing and check if data is stored correctly in the database.

4. Form Validation Errors Key Error : Missing form fields during registration or test submission.

Solution: Implemented robust input validation using try-except blocks and display detailed error messages after reading some articles in websites like stack overflow.

5. Search and Filter Issues No Results Found : Queries returned no results even when matching data was present.

Solution: Ensure case-insensitive search using SQLAlchemy's `ilike()` method.

6. Reports errors : user test did not found.

Solution : Corrected commands for fetching data from existing tables.

7. Route errors : Page did not found.

Solution : Corrected routes for fetching the existing Html file from Template.

8. Data Delete In Tables : Can not Delete Database is used in Another Table.

Solution : Manually Deleted Data.