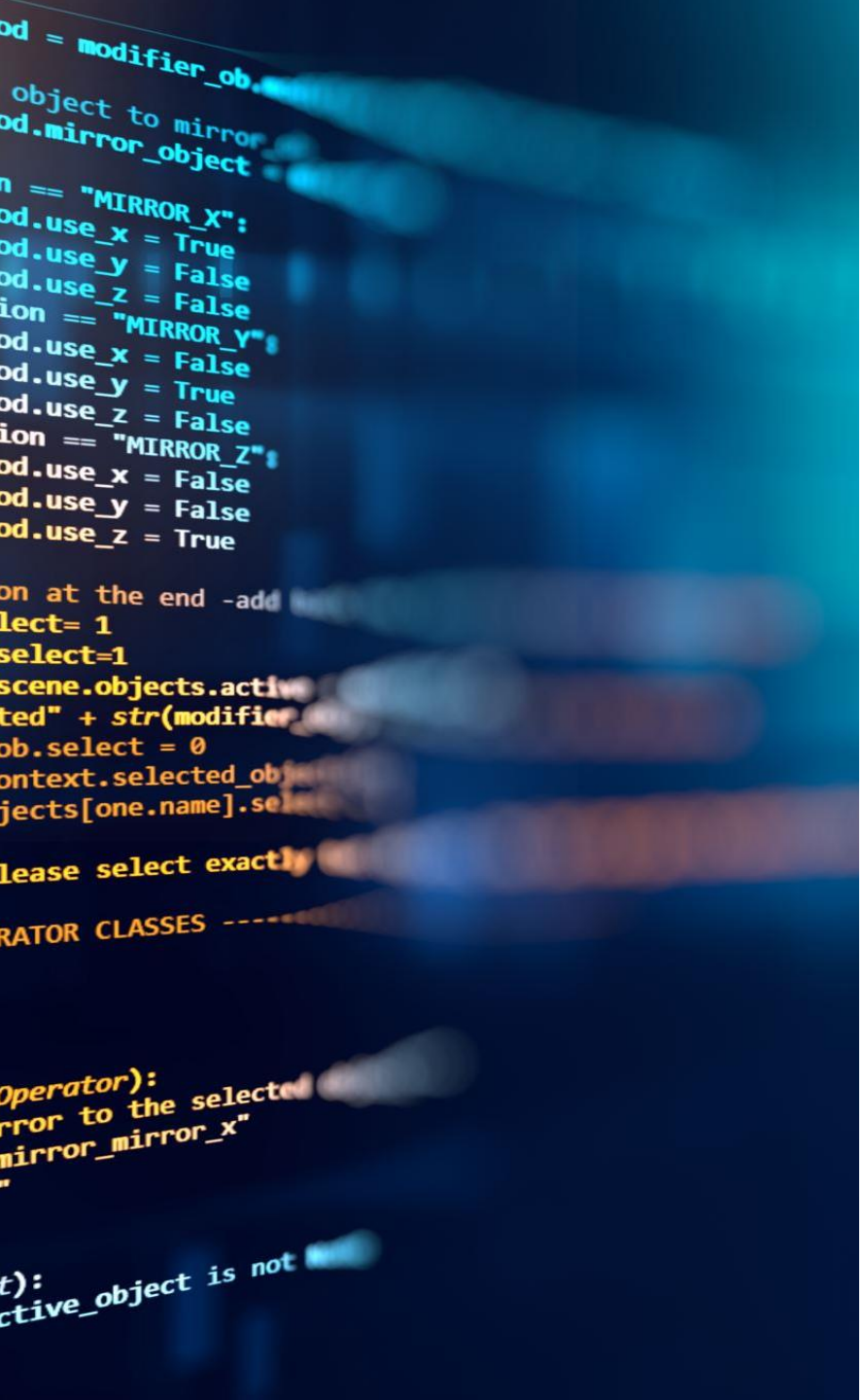
CODEQUEST

MILESTONE 2.0

TEAM A



MILESTONE 2.0 GOALS

Next Features to Implement

- Admin Panel: Allow teachers to manage quiz questions
- Leaderboard: Show top scores
- Improved UI/UX: Add animations, better styling
- More Question Categories: Python, SQL, Data Science (Not in Scope)
- User Empowerment: Own quiz creation by User (Not in Scope)
- Continued Testing: Test script writing (PyTest)

MILESTONE 2.0 FEATURE FULFILLED

Milestone 2.0 includes:

- Student login, Quiz taking and Leaderboard viewing
- Teacher login with Admin Panel to manage questions
- Custom quiz generation with filters
- Export to PDF, Email, Answer Toggle

USER STORIES FULFILLED 2.0

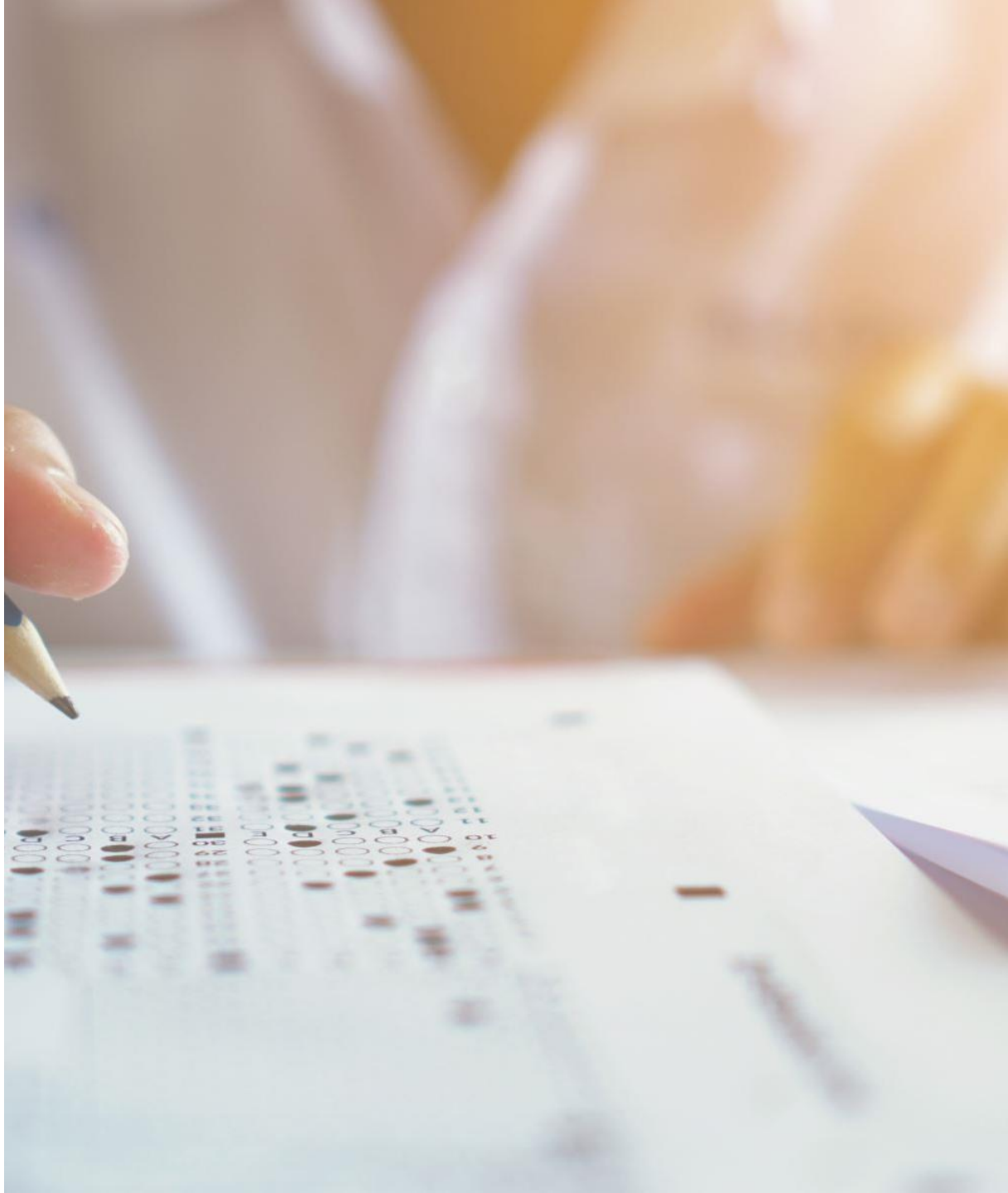
- **Admin Management (CRUD)**
  - As a teacher, I want to add, edit, and delete quiz questions from an admin panel so I can maintain up-to-date content.
- **Custom Quiz Filtering**
  - As a user, I want to filter quizzes by topic and difficulty so I can generate quizzes that match my learning goals.
- **Sharing/Exporting Features**
  - As a user, I want to export my quiz to PDF or send it via email so I can study offline or share with others.
- **User-Friendly UI**
  - As a student or teacher, I want a simple and clean interface so I can focus on the quiz without distractions.

A MESSAGE FROM OUR CLIENT

- Amazing job.
- Thank you for demoing the project.
- We really need the ability to compose custom quizzes.
- We are not talking about writing our own questions, we are not qualified to do that (yet).
- We would like to be able to click a button to generate a quiz.
- We should be able to specify a difficulty level as well as a topic area, but that should be optional - we may just want to click the create quiz button

SUMMARY OF CLIENT REQUEST

- We need the ability to compose custom quizzes.
- We should be able to specify a difficulty level as well as a topic area, but that should be optional.
- We may just want to click the create quiz button.



CLIENT REQUEST TO USER STORIES 1

As a teacher, I want to click a "Generate a Quiz" button so that I can easily create a quiz without writing questions myself.

As a teacher, I want to filter by difficulty so I can customize the quiz based on student levels.

As a teacher, I want to filter by topic to focus the quiz on specific subject areas.

As a teacher, I want to set the number of questions I want in the quiz so I can control quiz length.

CLIENT REQUEST TO USER STORIES 2

As a teacher, I want to preview the generated quiz so I can verify the questions before using it.

As a teacher, I want to show or hide correct answers and explanations so I can customize visibility based on my needs.

As a teacher, I want to export the quiz to PDF so I can print or distribute offline.

As a teacher, I want to email the quiz to myself for record-keeping or sharing with others.

WORKING CODE SNIPPETS

Here are some sample code snippets behind the key features we will demo.

```
<div class="form-group mb-3">
    <input type="text" name="teacher_code" class="form-control" placeholder="Teacher Code (optional)">
    <small class="form-text text-muted">Only fill this if you're registering as a teacher.</small>
</div>
```

Registration form includes optional teacher code for role assignment.

ANSWER TOGGLING LOGIC

```
<!-- Toggle Answer Visibility -->
{% if show_answers %}
<a href="{{ url_for('preview_quiz', category=category, topic=topic, limit=limit, show_answers=0) }}"
   class="btn btn-outline-warning">🙈 Hide Answers</a>
{% else %}
<a href="{{ url_for('preview_quiz', category=category, topic=topic, limit=limit, show_answers=1) }}"
   class="btn btn-success">✅ Show Answers</a>
{% endif %}
```

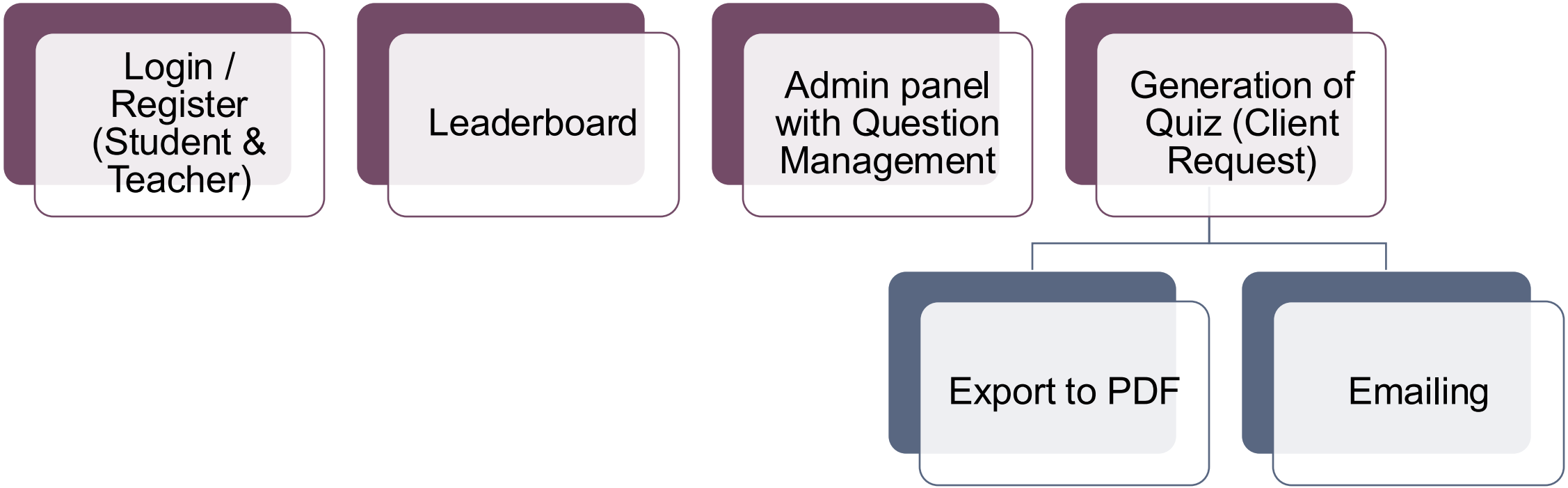
Dynamic rendering of answers using show answers flag.

GENERATE QUIZ PREVIEW

```python
@app.route('/admin/generate', methods=['GET'])
@login_required
def preview_quiz():
    if current_user.role != 'teacher':
        flash("Access denied", "danger")
        return redirect(url_for('index'))

    # 🔍 Get filter inputs
    category = request.args.get('category', '').strip().lower()
    topic = request.args.get('topic', '').strip().lower()
    limit = request.args.get('limit', '10').strip()  # ✅ Default to 10 if missing
    show_answers = request.args.get('show_answers', '1') == '1'
    export = request.args.get('export', '').strip().lower() == 'pdf'

    # 🔍 Query construction
    conn = get_db_connection()
    query = "SELECT * FROM quiz_questions WHERE 1=1"
    params = []

    if category:
        query += " AND LOWER(category) = ?"
        params.append(category)

    if topic:
        query += " AND LOWER(topic) LIKE ?"
        params.append(f"%{topic}%")

    query += " ORDER BY RANDOM()"
    if limit and limit.isdigit():
        query += " LIMIT ?"
        params.append(int(limit))

    questions = conn.execute(query, tuple(params)).fetchall()
    conn.close()
```

Flask route handling quiz generation and session-based exports.

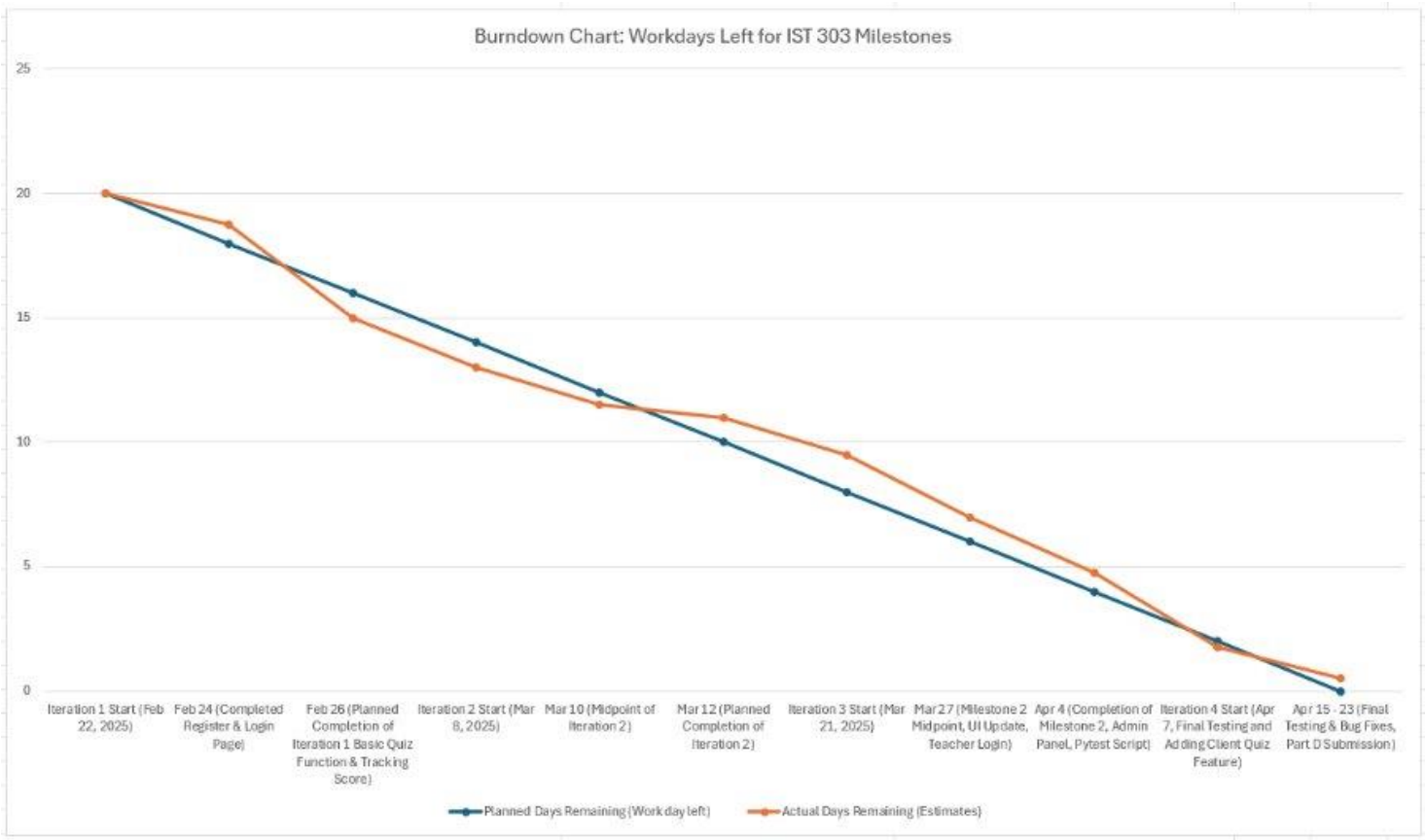
DEMO


Login / Register (Student & Teacher)
Leaderboard
Admin panel with Question Management
Generation of Quiz (Client Request)
Export to PDF
Emailing

AGILE PROCESS & BURNDOWN CHART

- Used Agile development with 4 iterations, each targeting milestone goals.
- Weekly team meetings and GitHub issues helped track progress and unblock challenges.
- Iteration planning adapted to evolving client needs, especially for the custom quiz feature.
- Frequent Git commits ensured version control and collaboration.
- Burndown chart tracks planned vs actual progress.
- Iteration 4 was added to address the new client feature request.

BURNDOWN RATE

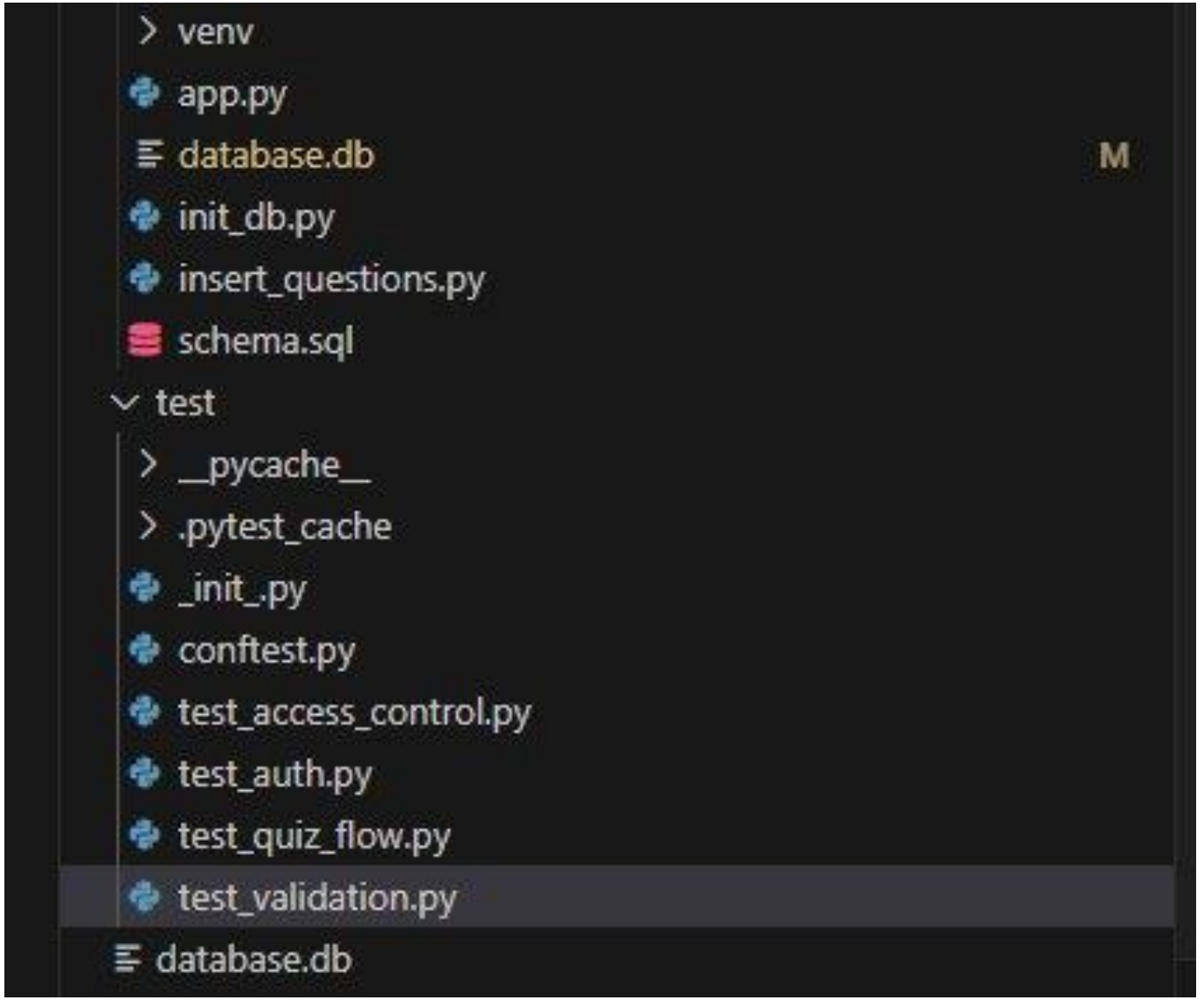


ITERATION BREAKDOWN

ITERATION	KEY DELIVERABLES
Iteration 1	Register/Login, Basic Quiz
Iteration 2	Quiz Flow, Score, Feedback
Iteration 3	Admin Panel, Question CRUD
Iteration 4	Custom Quiz Generator (Client Request) + Export

TESTING AND CODE COVERAGE

- We used Pytest for backend testing



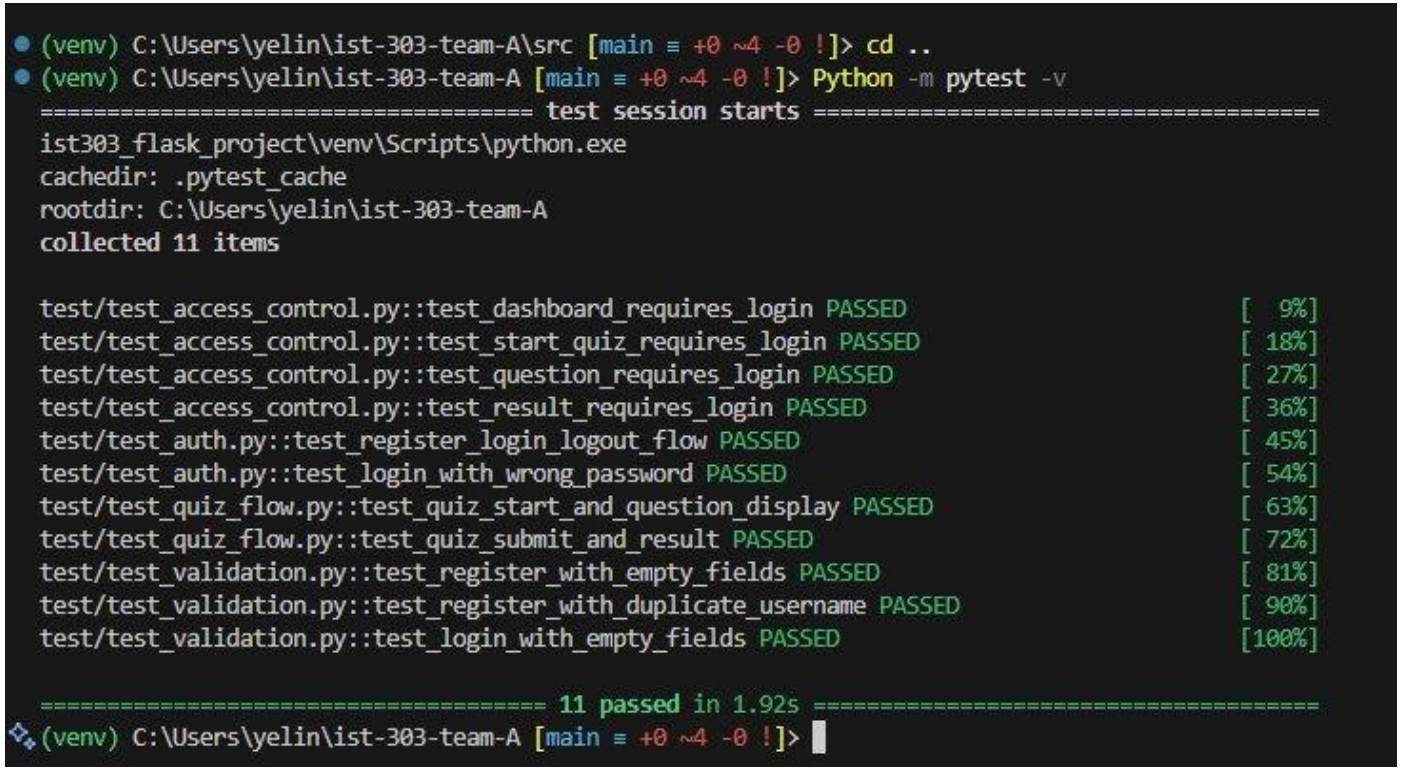
```python
def test_quiz_start_and_question_display(client):
    register_and_login(client)

    response = client.get('/start_quiz/easy', follow_redirects=True)
    assert b'Question' in response.data or b'select the correct answer' in response.data
    assert b'Option A' in response.data
```

Test simulates quiz start and verifies question rendering.

```python
def test_register_with_duplicate_username(client):
    # First registration
    client.post('/register', data={
        'username': 'duplicateuser',
        'password': 'abc123'
    }, follow_redirects=True)

    # Second registration with same username
    response = client.post('/register', data={
        'username': 'duplicateuser',
        'password': 'differentpass'
    }, follow_redirects=True)

    assert b'Username already exists' in response.data
```

Validation edge cases like duplicate registration and empty login fields.

PYTEST



- We implemented automated testing using pytest.
- A total of 11 tests were created across 3 categories:
  - Access control tests (e.g., login required to access quiz pages)
  - Authentication tests (Register, Login, Logout Validation)
  - Quiz flow tests (Question navigation, scoring, results)
- All tests passed successfully (11/11) with pytest -v

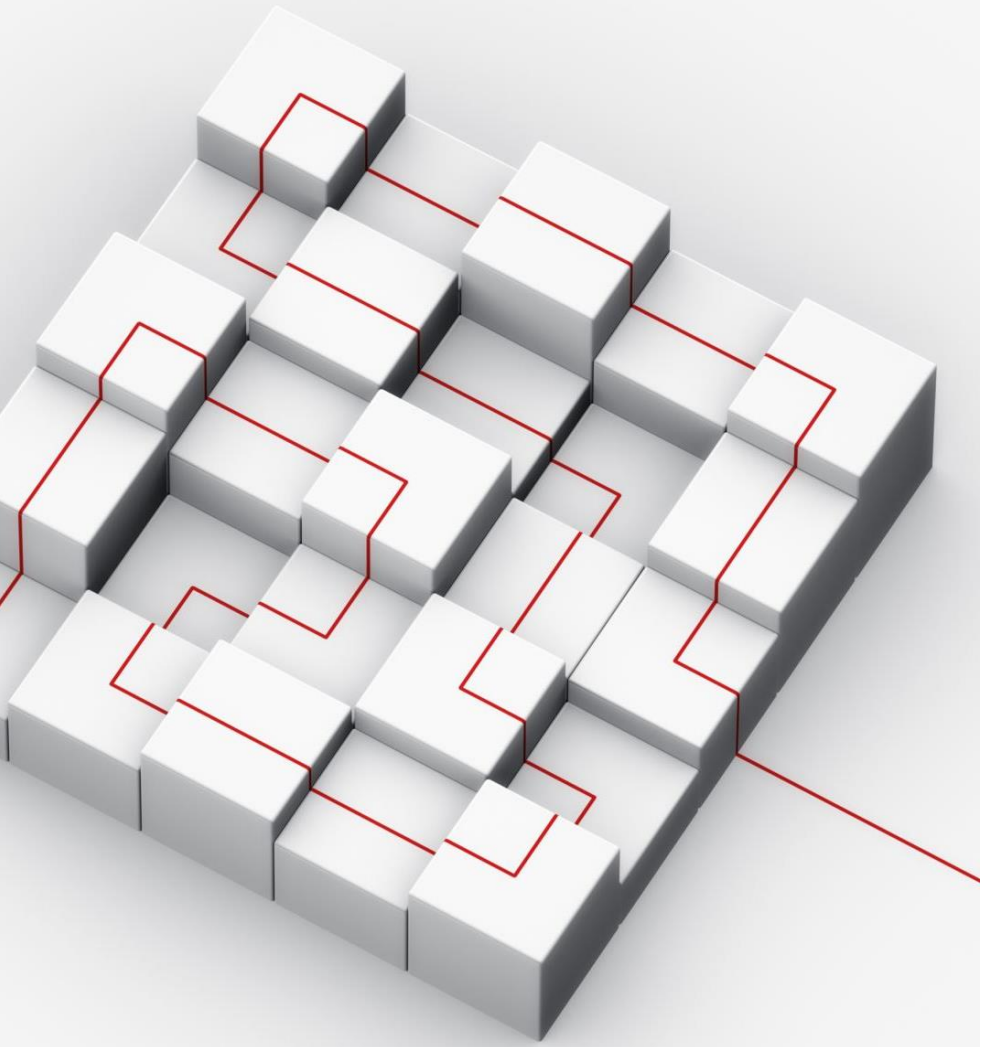
WHAT WE TESTED?

- Manually tested all form inputs, routes and exports
  - Registration
  - Login
  - Quiz Scoring logic
  - Role-based access
  - Custom Quiz Filtering
- Code coverage includes key routes and logic (80% +)

WHAT WE LEARNED

Successes:

- Developed a fully working Flask quiz app with separate **Student and Teacher roles.**
- Created a dynamic **Admin Panel** with full **CRUD** capabilities for quiz management.
- Implemented **custom quiz generation** with filters (difficulty, topic, number of questions).
- Built **answer toggling** logic using Flask routes and session flags.
- Enabled **Export to PDF** and **Email Quiz** features with clean formatting.
- Followed Agile workflow and adapted to real-time **client change requests.**
- Achieved **100% test pass rate** across auth, quiz flow, and validation modules.



CHALLENGES

Designing the Questions Table & Schema
- *Creating a normalized and flexible structure for quiz questions with fields like difficulty, topic, and explanations required multiple iterations.*

Generating Initial Questions with insert_questions.py
- *Ensuring question integrity, diverse difficulty levels, and avoiding duplicates during database population took several refinements.*

Leaderboard Score Logic
- *Tracking scores across sessions, maintaining unique usernames, and dynamically updating leaderboard rankings required precise route control.*

Session & Route State Mismatch for Answer Toggle
- *Handling dynamic answer display (show/hide) without breaking route/session continuity needed query param strategies and fallback handling.*

PDF Export Instability
- *PDFs initially failed due to long content, unicode issues, and rendering errors. We fixed this by refining templates and adjusting margins.*

LESSONS LEARNED

Clarify Client Needs Early	**Avoid rework by locking requirements upfront.**
Use Sessions Wisely	**Helps preserve user state and toggle options cleanly.**
Evolve UI with Features	**User experience should adapt with functionality.**
Understand Flask Architecture	**We learned how routing, schema, templates, and logic tie together.**
Agile Helps Manage Change	**Allowed us to respond quickly to client's new quiz generation request.**
Visualize User Flows First	**Sketching quiz flow helped us design a smoother experience.**

THANK YOU!

Team A

MAIN USER STORIES USED IN OUR PROJECTS

User Registration & Login

- *As a new user, I want to register for an account so I can use the quiz app.*
- *As a student, I want to log in with my credentials so I can take quizzes.*
- *As a teacher, I want to log in with my teacher credentials so I can manage questions.*

MAIN USER STORIES USED IN OUR PROJECTS

Taking a Quiz (Student Features)

- *As a student, I want to view available quizzes so I can choose one to take.*
- *As a student, I want to answer multiple-choice questions so I can test my knowledge.*
- *As a student, I want to review my answers before submitting the quiz.*
- *As a student, I want to submit the quiz and see my score instantly.*
- *As a student, I want to see my score compared to others on a leaderboard.*
- *As a student, I want to view a leaderboard to see top performers.*
- *As a student, I want to view my previous quiz attempts so I can track my progress over time.*
- *As a student, I want to see a timer while taking a quiz so I can manage my time.*

MAIN USER STORIES USED IN OUR PROJECTS

Teacher Admin Features

- *As a teacher, I want to add new quiz questions so I can grow the question pool.*
- *As a teacher, I want to edit existing questions so I can correct or improve them.*
- *As a teacher, I want to delete questions that are outdated or incorrect.*
- *As a teacher, I want to filter questions by topic or difficulty to find them quickly.*
- As a teacher, I want to be restricted to teacher-only features to keep student access secure.

MAIN USER STORIES USED IN OUR PROJECTS

- **Custom Quiz Generation (Client Request in Iteration 4)**
- *As a user, I want to generate a quiz with a button click so I don't need to select questions manually.*
- *As a user, I want to optionally filter generated quizzes by difficulty.*
- *As a user, I want to optionally filter generated quizzes by topic.*
- *As a user, I want to choose how many questions appear in the generated quiz.*
- *As a user, I want to preview the quiz before exporting or sharing.*

MAIN USER STORIES USED IN OUR PROJECTS

Sharing & Exporting

- *As a user, I want to toggle showing or hiding answers in the quiz preview.*
- *As a user, I want to export the quiz to PDF so I can print or save it.*
- *As a user, I want to email the quiz for distribution or review.*

UI & Project Maintenance

- *As a user, I want the app to look clean and responsive so I can use it easily on any device.*
- *As a developer, I want clear GitHub documentation and structure so others can understand and run the app.*
- As a user, I want to be logged out or notified when my session expires for security reasons.

FUTURE PIPELINE: HOSTING CODEQUEST ONLINE

Render.com

- Free tier (500 hrs/month for web services)
- Auto-deploys directly from GitHub
- Comes with HTTPS and custom domain support
- Highly beginner-friendly interface

Steps to Deploy:

- Connect to GitHub Repo
- Choose "Web Service"
- Use commands
  - pip install –r requirements.txt
  - gunicorn app:app

Why Render?

- Simple to use – no DevOps needed
- Smooth deployment process
- Great transition from classroom project to real-world app