

I. Verwendete Hilfsmittel

Nr.	KI-Basierte Hilfsmittel	Einsatzform, Verwendungszweck	Betroffene Teile der Arbeit	Bemerkungen
1	ChatGPT, https://chatgpt.com	Frontend Additions & Tweaks	style.ccs	Used AI to add to the css file, make some adjustments for responsiveness and better layout. See prompt used below
2	Claude Sonnet 3.7 https://claude.ai/chat/	Backend fluidity	models.py	Fixed the database relationship errors & add better data validation
3	Claude Sonnet 3.7 https://claude.ai/chat/	Script creation to add students to a classroom	add_test_students.py	Completely written by Claude, see prompts below
4	ChatGPT, https://chatgpt.com	Final assessment and help to understand bug fix	views.py	Used AI to shorten sections and help with bug fixes. Multiple prompts

Prompts for AI use:

1. add_test_students.py

„Create a Python Flask script that adds 10 test students (st1@test.com to st10@test.com) to a "Full-Stack Web Development SoSe-2025" classroom. Include database setup, password hashing, error handling, and join existing students to the classroom if they already exist. Password should be "12345678" for all students.“

Response:

```
1. #CREATED WITH CHAT GPT AND CHANGED INFORMATION
2. # DO NOT DELETE THIS COMMENT
3. """
4. Script to add 10 test students to the database and join them to the
   "History 101" classroom.
5. Run this script from the root directory of your project.
6. """
7.
8. import sys
9. import os
10. from werkzeug.security import generate_password_hash
11.
12. # Add the project root to the Python path
13. sys.path.insert(0, os.path.dirname(os.path.abspath(__file__)))
14.
15. from website import create_app, db
16. from website.models import User, Classroom
17.
18. def add_test_students():
19.     """Add 10 test students and join them to History 101 classroom"""
20.
21.     app = create_app()
22.
23.     with app.app_context():
24.         # Find the History 101 classroom
25.         classroom = Classroom.query.filter_by(name="Full-Stack Web
           Development SoSe-2025").first()
26.
27.         if not classroom:
28.             print("Error: Could not find classroom named 'Full-Stack Web
           Development SoSe-2025'")
29.             print("Available classrooms:")
30.             for c in Classroom.query.all():
31.                 print(f"    - {c.name}")
32.             return
33.
34.         print(f"Found classroom: {classroom.name} (Join Code:
           {classroom.join_code})")
35.
36.         # Create 10 test students
37.         students_added = 0
38.         students_joined = 0
39.
40.         for i in range(1, 11):
41.             email = f"st{i}@test.com"
42.
43.             # Check if student already exists
44.             existing_student = User.query.filter_by(email=email).first()
45.
46.             if existing_student:
```

```

47.         print(f"Student {email} already exists")
48.
49.         # Check if already joined the classroom
50.         if existing_student not in classroom.students:
51.             classroom.students.append(existing_student)
52.             students_joined += 1
53.             print(f" - Joined {email} to {classroom.name}")
54.         else:
55.             print(f" - {email} already in {classroom.name}")
56.     else:
57.         # Create new student
58.         new_student = User(
59.             email=email,
60.             first_name=f"Student {i}",
61.             password=generate_password_hash("12345678",
method='pbkdf2:sha256'),
62.             role='student',
63.             learning_points=0
64.         )
65.
66.         db.session.add(new_student)
67.         db.session.flush() # Flush to get the ID
68.
69.         # Join the classroom
70.         classroom.students.append(new_student)
71.
72.         students_added += 1
73.         students_joined += 1
74.         print(f"Created and joined student: {email}")
75.
76.         # Commit all changes
77.         try:
78.             db.session.commit()
79.             print(f"\nSuccess!")
80.             print(f" - {students_added} new students created")
81.             print(f" - {students_joined} students joined to
'{classroom.name}'")
82.             print(f" - Total students in classroom:
{classroom.students.count()}")
83.
84.             print(f"\nAll students can now log in with password:
12345678")
85.             print(f"Classroom join code: {classroom.join_code}")
86.
87.         except Exception as e:
88.             db.session.rollback()
89.             print(f"Error committing to database: {e}")
90.
91. if __name__ == "__main__":
92.     add_test_students()

```

2. style.css

Prompt:

"I need a CSS framework for a quiz app with a dashboard, statistics display, and leaderboard. It should have a modern UI with cards, stats displays, and a responsive layout. Please provide structural classes without any specific values or implementations."

Framework Structure Outline:

```css

```
1. /* Theme Variables */
2. :root {
3. --primary: [COLOR];
4. --secondary: [COLOR];
5. --success: [COLOR];
6. --warning: [COLOR];
7. --danger: [COLOR];
8. /* Base colors */
9. --bg: [COLOR];
10. --surface: [COLOR];
11. --text: [COLOR];
12. --text-muted: [COLOR];
13. --border: [COLOR];
14. }
15.
16. /* Core Layout Classes */
17. .container { }
18. .container-sm { }
19. .row { }
20. .col-12 { }
21. .col-lg-8 { }
22. .col-lg-4 { }
23. .col-md-6 { }
24.
25. /* Quiz Dashboard Components */
26. .dashboard-grid { }
27. .dashboard-card { }
28. .dashboard-quiz-grid { }
29. .dashboard-quiz-card { }
30. .dashboard-quiz-content { }
31.
32. /* Statistics Components */
33. .stats-row { }
34. .stat-card { }
35. .stat-card-primary { }
36. .stat-card-secondary { }
37.
38. /* Leaderboard Components */
```

```

39. .leaderboard-container { }
40. .leaderboard-item { }
41. .student-info { }
42. .score-display { }
43.
44. /* UI Components */
45. .btn { }
46. .card { }
47. .alert { }
48. .badge { }
49. .form-group { }
50.
51. /* Utility Classes */
52. .d-flex { }
53. .text-center { }
54. .mb-3 { }
55. .w-100 { }
56.
57. /* Responsive Breakpoints */
58. @media (max-width: 768px) {
59. .dashboard-grid { }
60. .stats-row { }
61. .leaderboard-item { }
62. }

```

This structure outlines all major components needed for the quiz application while leaving implementation details open. It follows the same organization as your current file but without specific values, ready for customization.

### 3. Models.py

```

Prompt:
" I need a database model structure for a gamified quiz app. It should include level-based user management, a classroom system, question-and-answer quizzes, and a reward shop. Please describe the core models and their relationships. "

```

```

Model Structure Outline:

```

```

```python
# Core User System
class User(db.Model):
    """
    Manages users (teachers/students) with gamification elements
    - Basic user info
    - Learning points/level system
    - Classroom relationships
    - Customization (avatars)
    """

# Quiz System
class Quiz(db.Model):
    """

```

```

Core quiz structure
- Quiz metadata
- Teacher relationship
- Question collection
- Classroom assignments
"""

class Question(db.Model):
    """
    Quiz questions
    - Question content
    - Point value
    - Answer options
    """

class Answer(db.Model):
    """
    Answer options
    - Answer text
    - Correctness flag
    """

# Classroom Management
class Classroom(db.Model):
    """
    Virtual classroom space
    - Basic info
    - Teacher/student relationships
    - Quiz assignments
    - Join system
    """

class StudentQuizAttempt(db.Model):
    """
    Tracks student quiz performances
    - Score tracking
    - Classroom context
    """

# Reward System
class ShopItem(db.Model):
    """
    Purchasable items/rewards
    - Cosmetic items
    - Special features
    - Point costs
    """

class UserPurchase(db.Model):
    """
    Tracks item purchases
    - User ownership
    - Purchase history
    """

# Relationships to consider:
# - Users <-> Classrooms (many-to-many)

```

```
# - Classrooms <-> Quizzes (many-to-many)
# - Users <-> Shop Items (through purchases)
# - Questions -> Answers (one-to-many)
....
```

This outline focuses on:

- Core functionality of each model
- Key relationships between models
- Gamification elements
- Extensibility points

It provides a foundation for implementing specific features while keeping the structure flexible for future additions.

4. Views.py

```
### Prompt:
```

```
" I need a high-level overview of Flask's view functions for a gamified quiz application. It should include all options for teachers and students, such as classroom management, quiz creation/taking, and shop functionality. Outline the route, authentication requirements, core logic flow, and key database operations for each view. Use Python-style function signatures with docstrings describing their purpose. "
```

```
### Unimplemented Views.py Structure:
```

```
```python
```

```
Core View Structure
```

```
from flask import Blueprint, render_template, request, redirect, url_for, flash, jsonify
```

```
from flask_login import login_required, current_user
```

```
from .models import Classroom, User, Quiz, Question, Answer, StudentQuizAttempt, ShopItem, UserPurchase
```

```
from . import db
```

```
views = Blueprint('views', __name__)
```

```
=====
```

```
HELPER FUNCTIONS
```

```
=====
```

```
def require_role(role: str):
```

```
 """Check if current user has required role"""
```

```
 pass
```

```
def get_classroom_or_404(classroom_id: int, teacher_check=True):
```

```
 """Retrieve classroom with ownership validation"""
```

```
 pass
```

```
def get_quiz_or_404(quiz_id: int, teacher_check=True):
```

```
 """Retrieve quiz with ownership validation"""
```

```
 pass
```

```
=====
```

```
CORE APPLICATION VIEWS
```

```
=====
```

```
@views.route('/')
```

```
@login_required
```

```
def home():
```

```
 """
```

```

Main dashboard view
- Students: Shows available quizzes
- Teachers: Shows management options
"""
pass

@views.route('/profile')
@login_required
def profile():
 """User profile with level progression"""
 pass

=====
QUIZ MANAGEMENT VIEWS
=====

@views.route('/take-quiz/<int:quiz_id>/<int:classroom_id>', methods=['GET', 'POST'])
@login_required
def take_quiz(quiz_id, classroom_id):
 """
 Student quiz interface
 GET: Display quiz questions
 POST: Process quiz answers and calculate score
 """
 pass

@views.route('/teacher/quiz/<int:quiz_id>/manage_questions', methods=['GET', 'POST'])
@login_required
def teacher_manage_quiz_questions(quiz_id):
 """
 Quiz question management
 GET: Show existing questions
 POST: Add new question
 """
 pass

@views.route('/teacher/question/<int:question_id>/manage_answers', methods=['GET', 'POST'])
@login_required
def teacher_manage_question_answers(question_id):
 """
 Answer management for questions
 GET: Show existing answers
 POST: Add new answer
 """
 pass

=====
CLASSROOM MANAGEMENT
=====

@views.route('/teacher/classrooms')
@login_required
def teacher_classrooms():
 """List all classrooms owned by teacher"""
 pass

@views.route('/teacher/create_classroom', methods=['GET', 'POST'])

```



```

@login_required
def create_classroom():
 """
 Classroom creation
 GET: Show creation form
 POST: Process new classroom
 """
 pass

@views.route('/teacher/edit_classroom/<int:classroom_id>', methods=['GET', 'POST'])
@login_required
def edit_classroom(classroom_id):
 """
 Classroom management hub
 - Update classroom details
 - Manage quizzes
 - View leaderboard
 - Handle student roster
 """
 pass

@views.route('/student/join_classroom', methods=['GET', 'POST'])
@login_required
def student_join_classroom():
 """
 Classroom joining interface
 GET: Show join form
 POST: Process join code
 """
 pass

@views.route('/student/my_classrooms')
@login_required
def student_my_classrooms():
 """List classrooms student has joined"""
 pass

=====
SHOP & REWARDS SYSTEM
=====

@views.route('/shop')
@login_required
def shop():
 """Display available shop items"""
 pass

@views.route('/shop/buy/<int:item_id>')
@login_required
def buy_item(item_id):
 """Process item purchase transaction"""
 pass

@views.route('/shop/select-avatar/<avatar_icon>')
@login_required
def select_avatar(avatar_icon):
 """Set selected avatar for user"""

```

```

pass

=====
API ENDPOINTS
=====

@views.route('/api/students')
def api_students():
 """JSON data of student information"""
 pass

@views.route('/api/shop-items')
def api_shop_items():
 """JSON data of shop items"""
 pass

=====
SUPPORTING VIEWS
=====

@views.route('/teacher/kick_student/<int:classroom_id>/<int:student_id>', methods=['POST'])
@login_required
def kick_student(classroom_id, student_id):
 """Remove student from classroom"""
 pass

@views.route('/teacher/quiz/<int:quiz_id>/upload_to_classroom/<int:classroom_id>')
@login_required
def upload_quiz_to_classroom(quiz_id, classroom_id):
 """Publish quiz to classroom"""
 pass

@views.route('/teacher/quiz/<int:quiz_id>/review/<int:classroom_id>')
@login_required
def quiz_review_details(quiz_id, classroom_id):
 """Detailed quiz results analysis"""
 pass
...

Key Relationships and Flow:
1. **Authentication Flow**:
- All routes require `@login_required`
- Role-based access control via `require_role()`

2. **Classroom Lifecycle**:


```

mermaid
graph TD
A[Create Classroom] --> B[Edit Classroom]
B --> C[Add Quizzes]
C --> D[Student Joins]
D --> E[Take Quiz]

```


...

3. **Quiz Workflow**:


```

mermaid
graph LR

```


```

```
A[Create Quiz] --> B[Add Questions]
B --> C[Add Answers]
C --> D[Publish to Classroom]
D --> E[Students Take Quiz]
'''
```

4. **Shop Flow**:

```
'''mermaid
graph LR
A[Earn LP] --> B[Browse Shop]
B --> C[Purchase Items]
C --> D[Select Avatar]
'''
```

This structure provides:

- Clear route definitions with HTTP methods
- Authentication requirements
- Core functionality descriptions
- Key database interactions
- Relationships between views
- Separation of concerns between user roles
- API endpoint definitions
- Support for gamification elements

The unimplemented version maintains the complete routing structure while removing implementation details, serving as both documentation and development roadmap.