LESSON 62

Testing and Validation

80 minutes (2 periods) | Python Web Backend | Outcome 2.19 | Outcome 2.20 | Outcome 2.21 | Outcome 2.22 | Outcome 1.22 | Outcome 3.2 | Outcome 3.3

WHAT THIS LESSON DOES

In this lesson, you'll explore the essentials of testing and validation. Learn to build a simple Flask app, test it manually, and add validation to manage invalid data, ensuring your app runs smoothly and handles errors effectively.

CURRICULUM OUTCOMES

Outcome	What the specification asks for
2.19	test solutions and decisions to determine their short-term and long-term outcomes
2.20	identify and fix/debug warnings and errors in computer code and modify as required
2.21	critically reflect on and identify limitations in completed code and suggest possible improvements
2.22	explain the different stages in software testing
1.22	read, write, test, and modify computer programs
3.2	create a basic relational database to store and retrieve a variety of forms of data types
3.3	use appropriate programming languages to develop an interactive website that can display information from a database that meets a set of users' needs

WHAT STUDENTS SHOULD BE ABLE TO DO

- Understand the importance of testing to ensure application functionality.
- Recognise the role of validation in maintaining data integrity.
- Develop skills to perform manual testing for identifying app issues.
- Implement input validation to prevent errors and crashes.
- Apply error-handling techniques to improve user experience.

BEFORE THE LESSON

- Run through the whole build yourself first: create the venv, pip install flask, run create_db.py, then app.py. The filter or a missing venv is what will stall the class.
- Confirm Flask installs from the terminal on the school network. If pip is blocked, arrange an offline wheel or a shared virtual environment in advance.
- Have create_db.py and the base app.py saved somewhere the class can copy from, so a typo in the SQL does not eat the period.

Testing and Validation

HOW THE LESSON RUNS

- Introduction
- Set Up Your Project
- Create Sample Database
- Build Basic Flask App
- Manual Testing
- Add Input Validation
- Challenge
- Wrap Up

TEACHING MOVES

- 2. Set Up Your Project. Walk the venv creation on the projector once. Check the terminal prompt shows (.venv) before anyone runs pip, or Flask installs to the wrong place and imports fail.
- 3. Create Sample Database. Run create_db.py once only. Running it twice throws a table-already-exists error and confuses everyone. Have them confirm users.db appeared in the folder before moving on.
- 4. Build Basic Flask App. Stress the parameterised query with the ? placeholders. Do not let anyone build the INSERT by joining strings, this is where SQL injection lives and it matters at Higher Level.
- 5. Manual Testing. Insist they run all three tests before touching any fix. The point is that an app that looks fine on good input is not tested until you feed it bad input. Let the TypeError crash happen.
- 6. Add Input Validation. Have students predict which of their three failing tests each new check catches before they retest. Note isdigit rejects negatives and decimals, which is why the int check follows it.
- 7. Challenge. Let them place the length check themselves rather than showing it. Then have them re-run the earlier tests to prove the new rule did not break the working cases.

WHAT TO WATCH FOR

- Students think an app that works when they type sensible data is finished and correct. Point back to the invalid age test. Their app looked perfect until someone typed abc. Correct means it behaves for bad input too, which is the whole reason for the manual tests.
- Assuming form data arrives as the right type, so age is treated as a number without checking. Show that request.form['age'] is always a string. That is why isdigit is checked before int(age), and why abc crashed before validation was added.
- Believing the CHECK constraint in the database makes app-level validation unnecessary. Explain the two are layers, not alternatives. The constraint protects the data but throws an ugly crash. Validation gives the user a readable message before the database is touched.

DIFFERENTIATION

- Emerging: If the venv or Flask install is fighting them, pair them onto a working machine so they still do the testing and validation work, which is the actual objective. Give them the three test cases written out and have them just run and report the result, before asking them to fix anything.
- Developing: Ask them to explain, in words, what each if condition rejects before they retest, so they reason about the checks rather than copying them.
- Proficient: Push them to validate age against a sensible upper bound and to strip whitespace from name, then justify each rule. Have them write their test cases as a short table of input and expected result. This is exactly the testing evidence they will need for the Coursework Project.