**CS440 Software Quality Assurance**

**HOS10A – Software Quality Assurance Plan**

06/05/2025 Developed by Clark Ngo

06/06/2025 Reviewed by Clark Ngo

School of Technology & Computing

City University of Seattle (CityU)

**Before You Start**

1. **Screenshots may be different from your environment.**
2. The directory path shown in the screenshots may be different from yours.
3. The steps might have subtle discrepancies. Please use your best judgment while completing each step in this cookbook-style tutorial.
4. Some steps may not be explained in detail. If you are not sure what to do:
5. Consult the resources from the course.
6. If you cannot solve the problem after a few tries (usually 15 -30 minutes), ask a TA for help.

**Learning Outcomes**

* Section 1: Preparing Your Environment
* Section 2: Exploring Software Quality Assurance Plan
* Section 3: Explaining the Code for Software Quality Assurance Plan
* Section 4: Submit your own Mini Software Quality Assurance Plan
* Section 5: Pushing Your Work to GitHub

**Section 1: Preparing Your Environment**

<https://cityuseattle.github.io/docs/git/github_codepsace/>

**Running the backend**

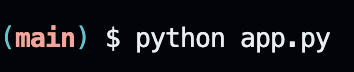
Change your directory to **backend,** install flask and flask-cors, and run it. In your terminal, type:

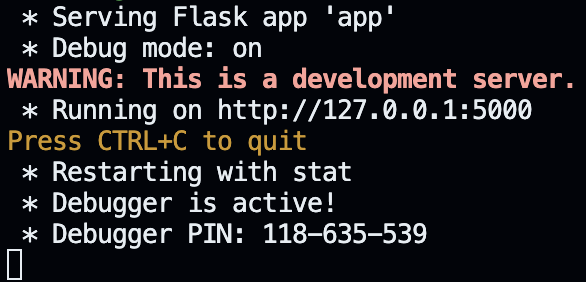
>>> cd backend

>>> pip install –r requirements.txt

* This command allows us to install multiple dependencies as specified in the text file.
* The -r flag in the pip install -r requirements.txt command stands for "requirement."
* It tells pip to install all the packages listed in the specified requirements.txt file.
* This is useful for setting up a project with all its dependencies in one go.

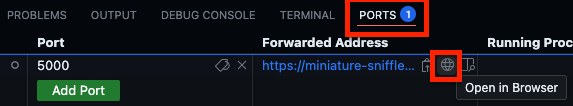




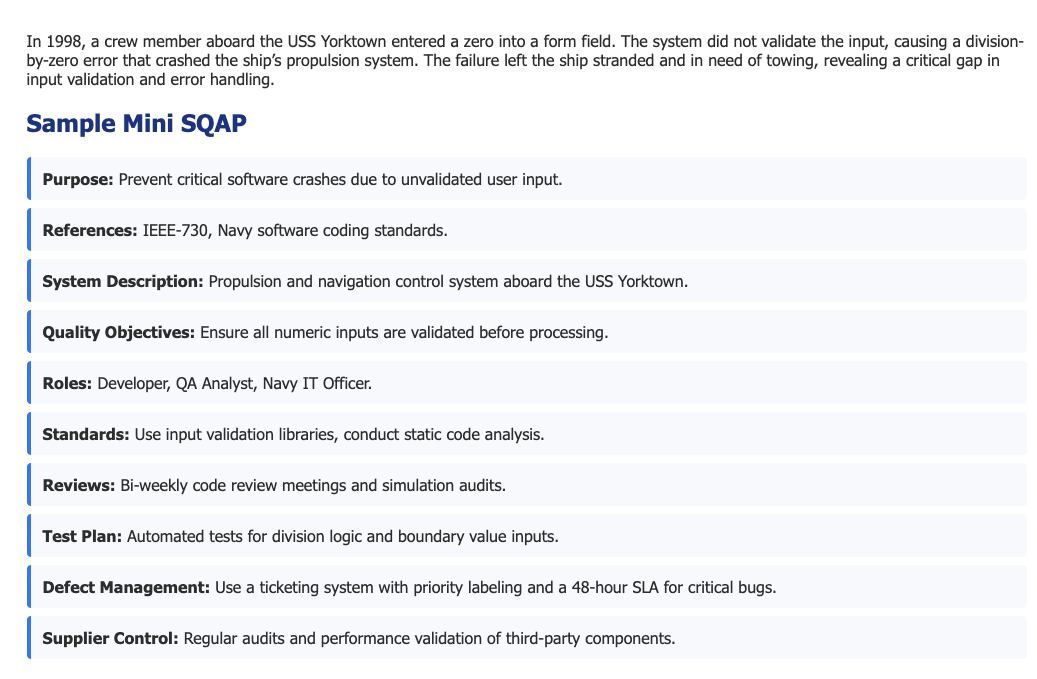


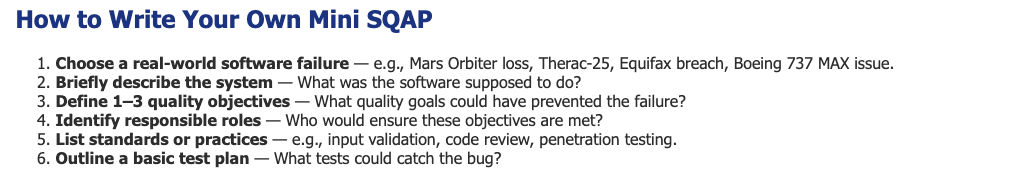
**Section 2: Exploring Software Quality Assurance Plan**

1. In your Ports tab in the row Port 5000, click the globe icon.



1. We should expect this page:





**✅ Purpose of an SQAP**

The main goal is to **prevent defects**, **ensure reliability**, and **verify that the software meets requirements**—not just through testing, but by embedding quality into every phase of development.

## **📄 What Does an SQAP Typically Include?**

Here's a breakdown of common components (based on **IEEE 730** standard):

1. **Purpose**
   1. Why the SQAP exists — the scope and objectives.
2. **References**
   1. Standards, policies, and documents the SQAP is based on (e.g., IEEE-730, ISO 25010, OWASP).
3. **Software Description**
   1. Overview of the system or product: its purpose, users, and components.
4. **Quality Objectives**
   1. Measurable goals like "99.9% uptime" or "100% input validation."
5. **Responsibilities**
   1. Roles and ownership (developers, QA engineers, project leads, vendors).
6. **Standards, Practices, and Conventions**
   1. What coding standards, documentation templates, or processes will be followed.
7. **Reviews and Audits**
   1. Types of reviews (code reviews, peer reviews, audit checkpoints) and their frequency.
8. **Test Plan Summary**
   1. High-level testing strategy (unit tests, integration tests, performance, security).
9. **Defect Management**
   1. How bugs will be logged, prioritized, tracked, and resolved.
10. **Supplier Control**
    1. If third-party vendors are involved, how their deliverables will be verified for quality.

## **🧠 Why Is an SQAP Important?**

* **Proactive, not reactive**: It helps prevent problems early instead of fixing them late.
* **Consistency**: Aligns the team on quality expectations and procedures.
* **Auditability**: Provides traceability and accountability — especially important for regulated industries (e.g., healthcare, aerospace).
* **Customer confidence**: Clients and stakeholders trust that there's a plan to maintain high quality.

The **USS Yorktown Division-by-Zero Crash (1998)** is a real and often-cited example of how **poor software quality assurance** can lead to **critical mission failure**.

## **🛳 What Happened?**

* The **USS Yorktown (CG-48)** was a U.S. Navy guided missile cruiser.
* In **September 1997**, during testing of a new computer-based control system called **Smart Ship**, a sailor **entered a zero** into a database field.
* The system **did not validate** the input.
* This led to a **division-by-zero error** in the ship's **Windows NT-based** software.
* The crash **shut down the propulsion system**, leaving the ship **dead in the water** for nearly 3 hours.

⚠️ The crew had to be towed back to shore — a major operational and reputational failure for a modern warship.

## **🧪 Why Is This Important?**

This incident is a **case study in software quality failures**, especially in mission-critical and military systems.

### **📌 Technical Failures:**

* No input validation (accepting zero as valid input).
* Poor exception handling (the crash wasn't caught or recovered).
* Over-reliance on untested COTS (Commercial Off-the-Shelf) software.

### **📌 Quality Assurance Gaps:**

* Lack of proper unit and boundary testing.
* Missing safeguards for human input errors.
* Inadequate fallback or fail-safe mechanisms.

## **📚 Lesson for QA and Engineering:**

|  |  |
| --- | --- |
| **Key Takeaway** | **Description** |
| Input Validation | Always validate user or sensor inputs, especially numeric fields. |
| Fault Tolerance | Mission-critical systems must gracefully handle errors. |
| Defensive Programming | Anticipate and prevent crashes using code-level safeguards. |
| Real-world Testing | Simulate extreme and edge-case scenarios (e.g., user enters 0). |
| QA Planning | An SQAP would include a requirement to prevent this (as shown in your project). |

## **🧠 Summary**

The **USS Yorktown crash** is a perfect example of how:

* A simple **software oversight** (no check for zero)…
* In a complex system (Navy propulsion)…
* Can lead to a **real-world operational disaster**.

**Section 3: Explaining the Code for Software Quality Assurance Plan**

This is a simple web application where:

1. Users (students) read a scenario about a software failure.
2. They view a sample Software Quality Assurance Plan (SQAP).
3. They fill out a form to create their own SQAP based on another scenario.
4. Submissions are stored in a local JSON file.

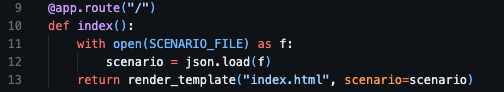
## **🔄 app.py – Server Logic**





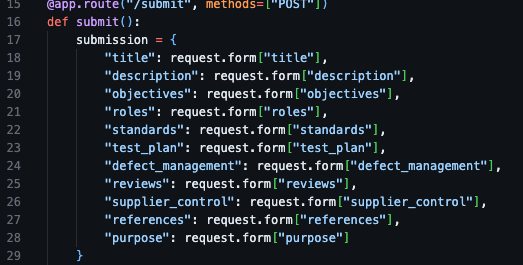
* **Flask** is the web framework.
* **render\_template** displays the HTML.
* **request** captures form data.
* **redirect** sends the user back to the homepage after form submission.
* **json + Path**: used to read/write scenario and submission data from disk.

### **🔗 @app.route("/") – Homepage**

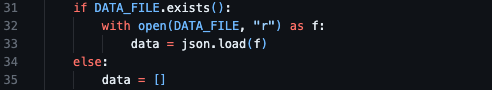


* Reads scenario.json, which contains the **scenario description** and **sample SQAP**.
* Passes this data to the index.html template.

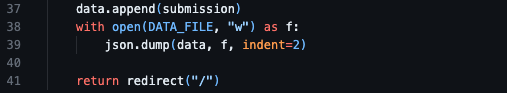
### **📨 @app.route("/submit", methods=["POST"]) – Submission Logic**



* Captures each form field from the HTML form using request.form.
* Builds a dictionary called submission.



* Checks if a file submissions.json already exists.
* If it does, loads current data.
* If not, initializes an empty list.

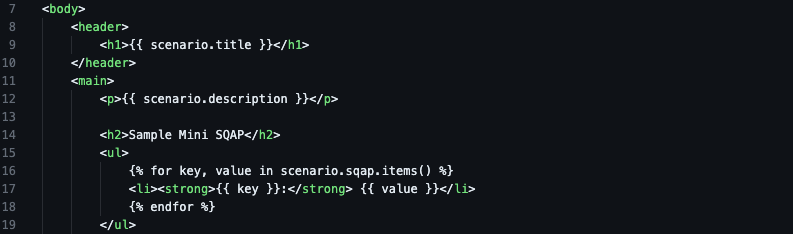


* Appends the new submission to the list.
* Saves the updated list to submissions.json.
* Sends the user back to the home page (which reloads the form).

## **🖼 templates/index.html – Frontend Page**

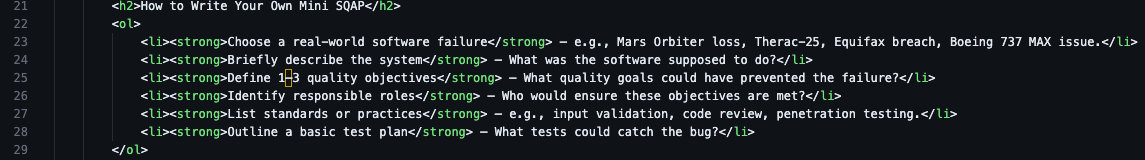
This template does three things:

### **✅ 1. Displays the scenario and sample SQAP:**



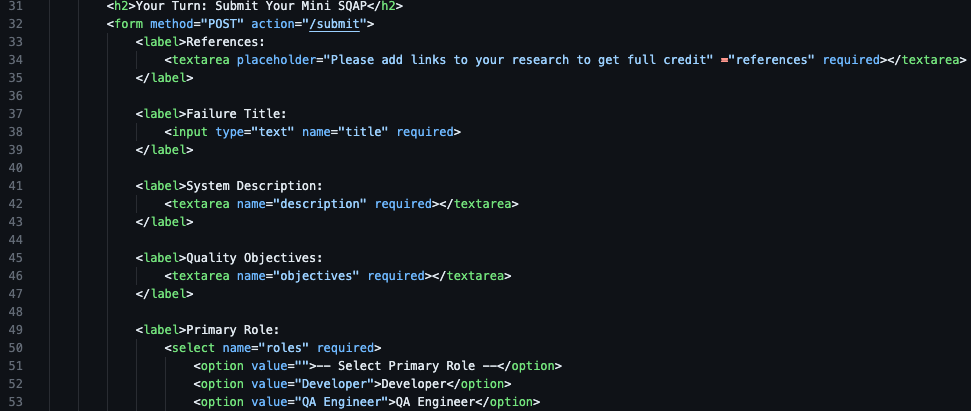
* These variables (scenario.title, scenario.sqap, etc.) come from the Python dictionary passed via render\_template().

### **🧑‍🎓 2. Provides context to help students write their own:**



* Just static instructional content to guide students before filling the form.

### **📝 3. The submission form:**



* Uses method POST and sends the form to /submit.

Includes a mix of:

* Text fields (<input type="text">)
* Text areas (<textarea>)
* Dropdown menus (<select>)

Each field is name="...", which Flask uses to read values via request.form["..."].

## **🔒 Local Storage**

All submissions are saved to a local file:

* submissions.json

**Section 4: Submit your own Mini Software Quality Assurance Plan**

**As a review, we head back to interacting with the app**

1. In the terminal, type:
   1. pip install –r requirements.txt
   2. python app.py
2. Visit:
   1. http://<codespacesurl> from your backend ports tab
3. **Execute these to receive full credit/grade**:
   1. Software Quality Assurance all fields have values
   2. **References will be verified**

**Section 5: Pushing Your Work to GitHub**

<https://cityuseattle.github.io/docs/git/codespaces_submission/>