1. Overall Grading Logic

- **Total available points: 130**
- **Passing threshold (grade 2): ≥ 50 points**
- **Best grade (grade 5): ≥ 100 points**
  (You cannot get more than 5, but you can collect more than 100 pts.)
- **Grades:**
  - **0–49 pts → 1 (fail)**
  - **50–69 pts → 2 (pass)**
  - **70–84 pts → 3**
  - **85–99 pts → 4**
  - **100+ pts → 5** (with extra conditions below)

Extra conditions (non-negotiable):

To **pass (grade 2 or higher)**:

1. Project **builds and runs tests with gradle test**.
2. **Basic config** is used (no hardcoded base URLs everywhere).
3. There is a **basic README** explaining how to run tests.
4. There is at least **some API testing and some UI testing** (not just one of them).

To get **grade 5** (top grade):

1. You have **≥ 100 points** total.
2. You have **API tests and UI tests with meaningful coverage**:
   - At least **18 pts from API** category (60% of max).
   - At least **18 pts from UI** category (60% of max).
3. You implemented **at least one of logging / reporting**, and specifically:

- **Allure reporting integrated** (local or via CI) **OR**
- At least **basic logging** in tests/framework.

4. You implemented **Jenkins CI integration** (at least 4 pts from CI category).

The rest is flexible: **you choose what to implement to reach your target grade**.

2. Category Overview (Points per Area)

Category, Max points
Core setup & Gradle/JUnit/Config 8
API tests (coverage + E2E flows) 30
UI tests (coverage + POM + E2E) 30
Framework design & config 16
Code quality, structure, readability 8
CI / Jenkins 14
Git usage 6
Reporting & logging (Allure, logs) 10
README / documentation 8
Total 130

3. Detailed Rubric by Category

3.1 Core Setup & Execution (8 pts)

ID	Task	Diff.	Pts	Notes / Requirements
C1	Project builds and runs tests with gradle test	Easy	4	**Required for passing.** JUnit tests must actually run.
C2	**Central configuration** is used (e.g., base URLs, env, browser)	Easy	2	No hardcoded URLs/paths in all tests. **Required for pass.**
C3	JUnit tags/categories + running specific suites via Gradle property (e.g., gradle test -Ptag=ui)	Medium	2	Bonus for flexible execution. Proper tagging for API/UI/etc.

3.2 API Tests – Coverage & E2E (30 pts)

You will be testing the provided **demo webshop API** using **RestAssured**. Points are based on **coverage and variety**, not raw test count.

ID	Task	Diff.	Pts	Explanation / Example
A	Basic API coverage: at least **5 tests** on different endpoints, checking status codes and 1–2 response fields	Easy	6	Full **CRUD happy-path coverage** for at least **3** one main resource (e.g., product, order) E.g., GET product list, GET product by id, POST order, etc.
A	Uses **RestAssured** consistently (request building instead of raw HTTP, no copy-paste of configuration)	Easy	2	given().baseUri(...).when().get(...).then(... is the norm.
A		Medium	8	Create–Read–Update–Delete tested;

	Mediu		assertions on responses for each step.	
A				
4	**Negative tests** (at least 3) – validation, 404,	unauthorized, etc.	**Easy 2**	E.g., missing required fields, invalid IDs, wrong auth.
A	Use of **RequestSpecification/ResponseSpecific ion or helper methods** to avoid repetition		m4	Common headers/base URI, auth update → verify → delete)
A	**Use of POJOs** for request and response bodies; map responses into Java objects and assert values from those objects		**m3**	Mediu Clean object-based testing instead of response.jsonPath().get("…").
A	An **end-to-end API flow** that chains multiple calls and assertions (e.g., create → verify →		Hard 5	One or more flows that reflect real user/API usage.

For grade 5: Aim for A1–A6; you'll want at least **18 pts** here.

3.3 UI Tests – Coverage, POM, E2E (30 pts)

You will be testing the specified **demo webshop UI** using **Selenium**.

Again, points are **coverage-based**.

ID	Task	Diff.	Pts	Explanation / Example
				on different pages / features
U1 U2		U3 U4		Use of **reasonable locators** (id, name, CSS, short XPath), not fragile mega XPaths
				Component-level coverage of key areas (≈10
		Basic UI tests: at least	**5 tests**	

meaningful tests): login, search, product details, cart, simple validations	product page, verify price displayed, etc.	E2E.
Proper use of **waits** (implicit or explicit), avoiding flakiness and Thread.sleep abuse	Easy 2 No //*[@id='something']/div[3]/div[2]/span[1] everywhere.	Medium 3 Use WebDriverWait/ExpectedConditions or similar.

Basic Page Object Model

Easy 5 E.g., login, open homepage, check title, open U5	Medium 7 Many small tests are fine; they don't all need to be	Tests call page methods instead of directly using WebDriver everywhere.
(POM) – each page has its own class with methods	**BasePage + inheritance**: Medium 4	Add a private field in your BasePage: private final string neptuneCode = Neptun.InsertYourCodeHere(). This field **must not be used anywhere**.
U6 U7	to parent class used by ≥2 pages	navigation.
common functionality extracted	**UI end-to-end flows** (≥ 2 flows) across multiple pages, asserting final result Hard 5 E.g., BasePage with driver, waits, common	Hard 4 E.g., login → add product to cart → go to cart; or full checkout flow if possible.

For grade 5: You should reach **at least 18 pts** here as well, with **at least some E2E tests (U7)**.

3.4 Framework Design & Config (16 pts)

ID Task Diff. Pts Explanation

F1	**Extended configuration**: separate config for API and UI (different base URLs, etc.), loaded from file(s) Medium 4	**Separated setup/teardown** for UI and API using JUnit E.g., config.properties, application.yml, or similar. UI setup uses WebDriver; API setup uses RestAssured; they don't mix.
F2	(@Before/@After/@BeforeClass/@AfterClass), no duplication annotations Medium 4	

F3	**Reusable helpers** for test data or repeated logic (e.g., creating a product/order, login helper)	Medium	3	through JUnit/Gradle/Selenium setup Methods that are reused in multiple tests instead of copy-paste. Not mandatory but rewarded if clean and used.
F4	Optional **utilities layer** (e.g., utils for random data, date handling, etc.)	Medium	2	Hard 3 Doesn't need to be perfect; showing that tests can run in parallel is enough.
F5	Basic **parallel execution support** (API or UI)			

3.5 Code Quality, Structure, Readability (8 pts)

This is **not** a Java course, but the code must be maintainable and understandable.

ID	Task	Diff.	Pts	Explanation
Q1	**Reasonable project structure**: packages by layer (api, ui, tests, pages, config, etc.)	Easy	3	No "everything in one package/file" chaos.
Q2	**Readability**: meaningful method/variable names, small methods, minimal copy-paste	Easy	3	You should be able to understand what a test does in a quick read.
Q3	**Basic OOP principles**: usage of classes and encapsulation (esp. pages & helpers)	Medium	2	No everything-static; tests use objects and page classes where appropriate.

3.6 CI / Jenkins Integration (14 pts)

Jenkins is an **important part of the exam**. There are two main ways to integrate: manual job or Jenkinsfile (you can do both and stack points).

ID	Task	Diff.	Pts	Explanation

	Jenkins job (manual configuration) that J1 checks out the repo and runs tests; documented in README (with screenshots)	Medium 4	Job must run gradle test (or equivalent). Screenshots/steps should be in README.
J2	**Jenkinsfile** stored in the repository that defines a pipeline to build and run tests	Hard 6	Declarative or scripted pipeline that can be used directly by Jenkins.
J3	Pipeline step to **deploy and start the API system under test** before running tests	Hard 2	Could be docker-compose up, Gradle task, script, etc., documented.
J4	Jenkins pipeline integrates **Allure report plugin** and shows results in the Jenkins job	Hard 2	Builds Allure report as part of the job; visible as a post-build action / link.

For **grade 5**, aim for at least J1 + J2 and ideally J3; J4 is a nice bonus.

3.7 Git Usage (6 pts)

Git is evaluated on **how** you use it, not on the number of commits.

ID	Task	Diff.	Pts	Explanation
G1	Multiple **logical commits** instead of a single “big dump”	Easy	2	Each commit should represent a small step or change (new tests, new feature, fix, etc.).
G2	**Meaningful commit messages**	Easy	2	“Add login tests”, “Implement product API negative tests”, not just “update” or “fix”.
G3	Use of **at least one feature branch** plus main/master	Medium	2	E.g., a feature/ui-tests or feature/ci branch merged into main.

3.8 Reporting & Logging (Allure, Logs) (10 pts)

For the **top grade**, you must implement at least one of these (Allure recommended).

ID	Task	Diff.	Pts	Explanation
R1	**Allure report integrated locally**: running tests produces Allure results, and README explains how to generate/view the report	Medium	6	E.g., gradle test → allure serve / allure generate instructions.
R2	**Basic logging** in framework/tests using Logger (java.util.logging, Log4j, SLF4J, etc.)	Easy	3	Logs important events: starting test, navigating to page, calling API, etc.
R3	**Advanced reporting use**: attaching extra data (logs, screenshots) to Allure or custom log levels	Hard	1	E.g., screenshot on failure attached to Allure, custom step annotations, etc.

Having **R1 or R2** plus enough points elsewhere satisfies the “logging/reporting” requirement for grade 5.

3.9 README / Documentation (8 pts)

We explicitly grade your README.

ID	Task	Diff.	Pts	Explanation
D1	**Basic README** including at least: how to run the tests (gradle test) and prerequisites (Java, etc.)	Easy	3	**Required for passing.** Someone should be able to clone and run from README alone.
D2	**Detailed README** including the following sections:	Medium	5	All items below expected for full points:

- How to run **UI tests** separately — E.g., with tags or specific Gradle params.
- How to run **API tests** separately
- How to run tests in **Jenkins** (job or pipeline)

• List of **dependencies & tools** (Java version, Gradle, Selenium, RestAssured, Allure, Jenkins)

• Short description of **project structure** (packages, layers)

• Short description of **framework architecture** (POM, helpers, config concept)

• (Optional but nice) **Screenshots** of reports / Jenkins / UI flows