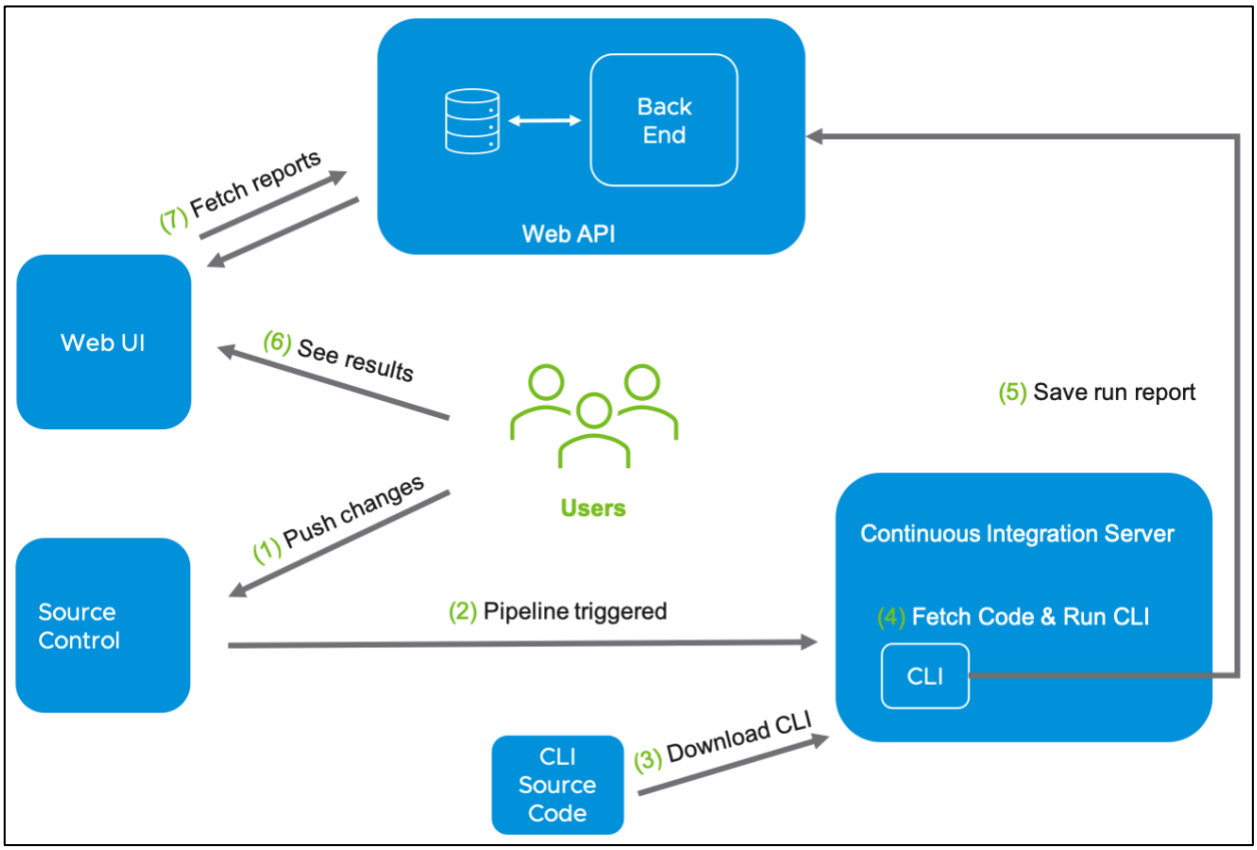
Universal Test Management System

For the latest several decades in software development it has been proven that automated testing is a part of every well-designed software development process. That's why the task is to create an automated system that: (1) **parses** a project test plan, (2) **runs** sets of tests, (3) **saves** their outcome, (4) **generates** a well-structured and compact report, and (5) **allows** users to see it through a web-based UI.

In this way we will be able to provide immediate feedback of the state of the software in any moment of its history and develop relatively fast and with confidence in our product!

Here is how the overall architecture will look like:



The task is broken down into 4 phases and each represents a completed state of the product. The more phases completed the more points will be earned. Some phases will build upon, remove, or alter requirements that are given in previous phases – this is expected and always the requirements from the **latest** developed phase must be applied.

Environment (Phase 1) – 10 points

Setup a **private** GitLab repository named "*utms-cli*". This repository will contain the main CLI runner which will be explained in the next phase. Private repositories will be visible only to repository owners and admins.

Download the pre-configured Jenkins VM and run it in VirtualBox, following the included readme file. After the setup is done, you should be able to access the Jenkins server in your browser at http://localhost:8080.

Create another **private** repository named "*utms-sandbox*". Configure a **Jenkins pipeline** that will run for each commit to the repository, cloning the "*utms-sandbox*" repository as its first stage. Commit the pipeline to the repository, naming it "*Jenkinsfile"*. Also, see the example below of the "*testing.yaml"* and upload it to the root of repository as well.

Join the #final-task-2021-discussion slack channel.

NOTE: For any issues with creating repositories contact *vchomakov@vmware.com* through email or via Slack.

UTMS-CLI (Phase 2) – 40 points

Build CLI – 25 points

Create a **Command Line Interface** that parses a YAML file containing the testing configuration and executes tests based on it. Example of a *testing.yaml*:

```
project:
  name: My Software Project
  description: The description of the project
suites:
  - BackEndTests:
      - Test1:
          enabled: true
          command: echo Test
      - Test2:
          enabled: true
          command: echo Test 2
          description: This test does something too
  - UITests:
      - Test3:
          enabled: true
          command: echo Test 3
          description: Hello world
      - Test4:
          enabled: false
          command: echo Test 4
```

The CLI lifecycle is:

1. **Parse** the configuration file
2. **Run** each test in it that has been marked as "enabled"
    a. detect start and end time
    b. execute the *command* field as a **native OS command**
    c. if the execution returns code 0 – the test has **passed**; if any other code is received the test has **failed**
    d. save the output or/and error of the executed command as Base64 string
    e. save test case description if any
3. **Generate** a JSON report and print it on the console

The CLI must be bundled as a single executable. This means that it is expected that commands should be parsed from the console. Example usage:

```
java -jar ./utms-cli.jar --config ./testing.yaml
```

No additional input streams should be used. Any future reference to the execution of the CLI will be summarized to just *`./utms-cli`* for brevity.

The CLI must allow the following **arguments**:

Argument	Description	Mandatory	Default Value	Example Value
`--config, -c`	The path to the test configuration YAML file	No	`testing.yml`	`testing.yml`
`--run-id, -r`	An ID for the run. Specify to have the system match your CI runs	No (if missing, the server will create its own ID)	None	39

Examples:

Input	Output
`./utms-cli --config ./testing.yaml --run-id 3`	`{ "runId": 3, "project": { "name": "My Software Project", "description": "The description of the project" }, "status": "passed", "suites": [{ "name": "BackEndTests", "status": "passed", "tests": { "Test1": { "description": null, "output": "VGVzdA==", "error": "", "status": "passed", "startDate": "2021-04-13T11:39:30.438441Z", "endDate": "2021-04-13T11:39:30.439093Z" }, "Test2": { "description": "This test does something too", "output": "VGVzdCAy", "error": "", "status": "passed",`

	"startDate": "2021-04-13T11:39:30.442684Z",
	"endDate": "2021-04-13T11:39:30.443858Z"
	}
	}
	},
	{
	"name": "UITests",
	"status": "passed",
	"tests": {
	"Test4": {
	"description": null,
	"output": null,
	"error": null,
	"status": "skipped",
	"startDate": null,
	"endDate": null
	},
	"Test3": {
	"description": "Hello world",
	"output": "VGVzdCAz",
	"error": "",
	"status": "passed",
	"startDate": "2021-04-13T11:39:30.447784Z",
	"endDate": "2021-04-13T11:39:30.449034Z"
	}
	}
	}
	]
	}

./utms-cli --config ./failing-test-testing.yaml --run-id 3	{
	"runId": 3,
	"project": {
	"name": "My Software Project",
	"description": "The description of the project"
	},

```
  "status": "failed",
  "suites": [
    {
      "name": "BackEndTests",
      "status": "passed",
      "tests": {
        "Test1": {
          "description": null,
          "output": "VGVzdA==",
          "error": "",
          "status": "passed",
          "startDate": "2021-04-13T11:39:30.438441Z",
          "endDate": "2021-04-13T11:39:30.439093Z"
        },
        "Test2": {
          "description": "This test does something
too",
          "output": "VGVzdCAy",
          "error": "",
          "status": "passed",
          "startDate": "2021-04-13T11:39:30.442684Z",
          "endDate": "2021-04-13T11:39:30.443858Z"
        }
      }
    },
    {
      "name": "UITests",
      "status": "failed",
      "tests": {
        "Test4": {
          "description": null,
          "output": null,
          "error": null,
          "status": "skipped",
          "startDate": null,
```

	<pre>"endDate": null
	},
	"Test3": {
	"description": "Hello world",
	"output": "VGVzdCAz",
	"error": "",
	"status": "failed",
	"startDate": "2021-04-13T11:39:30.447784Z",
	"endDate": "2021-04-13T11:39:30.449034Z"
	}
	}
	}
	]
	}</pre>

`./utms-cli --config ./not-existing-testing.yaml`	<pre>{
	"error": "Configuration file not found."
	}</pre>

`./utms-cli --config ./not-valid-testing.yaml`	<pre>{
	"error": "Configuration file is not valid."
	}</pre>

`./utms-cli --config ./valid-testing.yaml –run-id not-a-number`	<pre>{
	"error": "Run Id is not valid."
	}</pre>

Note: the *command* field represents a native OS command that must be executed for each test.

Setup pipelines – 15 points

Add a Jenkins pipeline for *utms-cli* that builds the CI. It should build the CLI as an artifact.

Create a *Jenkinsfile* in utms-cli. The pipeline should consist of three stages:

1. Fetch the utms-cli source code
2. Build the utms-cli source code
3. Publish the UTMS CLI application as an artifact

The new pipeline should run for each commit to the utms-cli repository, producing a new artifact on success.

Once this has been done, update the Jenkinsfile in the "*utms-sandbox*" repository, adding a new stage that copies the "*utms-cli*" artifact from the "*utms-cli*" pipeline. Then add a third stage, "*Run tests*", that uses the imported CLI to read *testing.yml* and print the requests to standard output.

BONUS TASKS* – 5 points per argument

Add the following arguments:

Argument	Description	Mandatory	Default Value	Example Value
`--suite-name, -sn`	Which exact suite to run	No	None	`UITests`
`--test-name, -tn`	Which exact test to run	No	None	`Test4`

Examples:

Input	Output
`./utms-cli --config ./testing.yaml –suite-name BackEndTest`	`{ "runId": 1, "project": { "name": "My Software Project", "description": "The description of the project" }, "status": "passed", "suites": [{ "name": "BackEndTests", "status": "passed", "tests": { "Test1": { "output": "VGVzdA==", "error": "", "status": "passed", "startDate": "2021-04-13T11:39:30.438441Z", "endDate": "2021-04-13T11:39:30.439093Z" }, "Test2": { "description": "This test does something too", "output": "VGVzdCAy", "error": "", "status": "passed", "startDate": "2021-04-13T11:39:30.442684Z", "endDate": "2021-04-13T11:39:30.443858Z" }`

	}
	},
	{
	"name": "UITests",
	"status": "skipped",
	"tests": {
	"Test4": {
	"output": null,
	"error": null,
	"status": "skipped",
	"startDate": null,
	"endDate": null
	},
	"Test3": {
	"description": "Hello world",
	"output": null,
	"error": null,
	"status": "skipped",
	"startDate": null,
	"endDate": null
	}
	}
	}
	]
	}

./utms-cli --config ./testing.yaml –suite-name BackEndTest2	{
	"error": "Test suite not found."
	}

./utms-cli --config ./testing.yaml –suite-name Test4
# note test is skipped in config 	
	{
	"runId": 1,
	"project": {
	"name": "My Software Project",
	"description": "The description of the project"
	},
	"status": "failed",
	"suites": [

```
    {
      "name": "BackEndTests",
      "status": "skipped",
      "tests": {
        "Test1": {
            "output": null,
            "error": null,
            "status": "skipped",
            "startDate": null,
            "endDate": null
          },
        "Test2": {
          "description": "This test does something
too",
          "output": null,
          "error": null,
          "status": "skipped",
          "startDate": null,
          "endDate": null
        }
      }
    },
    {
      "name": "UITests",
      "status": "failed",
      "tests": {
        "Test4": {
          "output": "",
          "error": "",
          "status": "failed",
          "startDate": "2021-04-13T11:39:30.447784Z",
          "endDate": "2021-04-13T11:39:30.449034Z"
        },
        "Test3": {
          "description": "Hello world",
```

	<pre>"output": null,
	"error": null,
	"status": "skipped",
	"startDate": null,
	"endDate": null
	}
	}
	}
	]
	}</pre>

BONUS TASK* – 10 points

Write unit tests to achieve at least 80% coverage for the CLI **OR**
Implement static code analysis for the CLI.

Back End (Phase 3) – 50 points

Design the database – 15 points

Entity	Description	Members
Project	• A project is the largest-scale entity, holding meta information on a project and all of its test runs. • Projects are created by the CLI through the web API the first time the CLI sends data about them. • A project's user-friendly name and description will be updated if they change in the project's configuration file. However, if the project ID changes, the system will create a new project with this ID instead	• A unique identifier (e.g., project_1) • A unique user-friendly name (e.g., "Project 1") • A description • A set of **Test Runs**
Test Run	• A set of test suites that have been executed within a test session • Its ID is either explicitly given by the CLI (so the back end can match it with a CI run ID), or is auto assigned by the back end (if not provided by the CLI call)	• An ID for the test run (unique for a given project) • The status of running the suite (in progress, passed, failed, skipped) • A set of **Test Suites**
Test Suite	• A set of tests that have been run within a test run	• A name (a test suite name is unique for a project, but not unique for the whole system) • The status of running the suite (passed, failed, skipped) •
Test Case	• An actual test case that was performed as a part of the suite • While a name will usually be common between different test runs, it's important to note the test may change over time. Unlike with projects where we create a project only once and then only update it, a test case is basically just a name—all of its other	• A name (unique per suite for a given project) • A description of the test • Whether the test was enabled • The status of running the test (started, passed, failed, skipped) • The console output from running the test

	values would be different for each of its iterations.	• Test start time • Test end time • Error output if the test failed

Develop project endpoints – 10 points

Endpoint	Method	User	Details
`/projects`	GET	UI	Gets all projects.
`/projects/:project_name`	GET	CLI	Gets projects by name.
`/projects`	POST	CLI	Registers a project by giving name and a description.

Develop test run endpoints – 15 points

Endpoint	Method	User	Details
`/projects/:project_id/runs`	GET	UI/CLI	Gets all runs for a project.
`/projects/:project_id/runs`	POST	CLI	Creates a new run. When a run is finished by the CLI send the given report to the backend to persist the outcome.
`/projects/:project_id/runs/:run_id`	GET	Project members (through the web UI)	Return the details of a test run for a project. The response should have the overall run status, names of the suites and the tests within, and their statuses.

Integrate with CLI – 10 points

Deploy the UTMS server locally. Extend the CLI to be able to send new project and test run requests. **Add the following arguments**:

Argument	Description	Mandatory	Default Value	Example Value
`--server, -s`	At which address the CLI to send the report. This should point a UTMS web API and follow the contracts given in the previous sub-sections.	Yes	None	`http://localhost:8080/api`

Make sure that after the CLI generates the report, it sends a POST request to the **server** where it is expected that this report will be persisted.

Remove the following arguments:

Argument	Description	Mandatory	Default Value	Example Value
~~`--run-id, -r`~~	~~An ID for the run. Specify to have the system match your CI runs~~	~~No (if missing, the server will create its own ID)~~	~~None~~	~~39~~

* The run id now should be set when the run is actually persisted in the database.

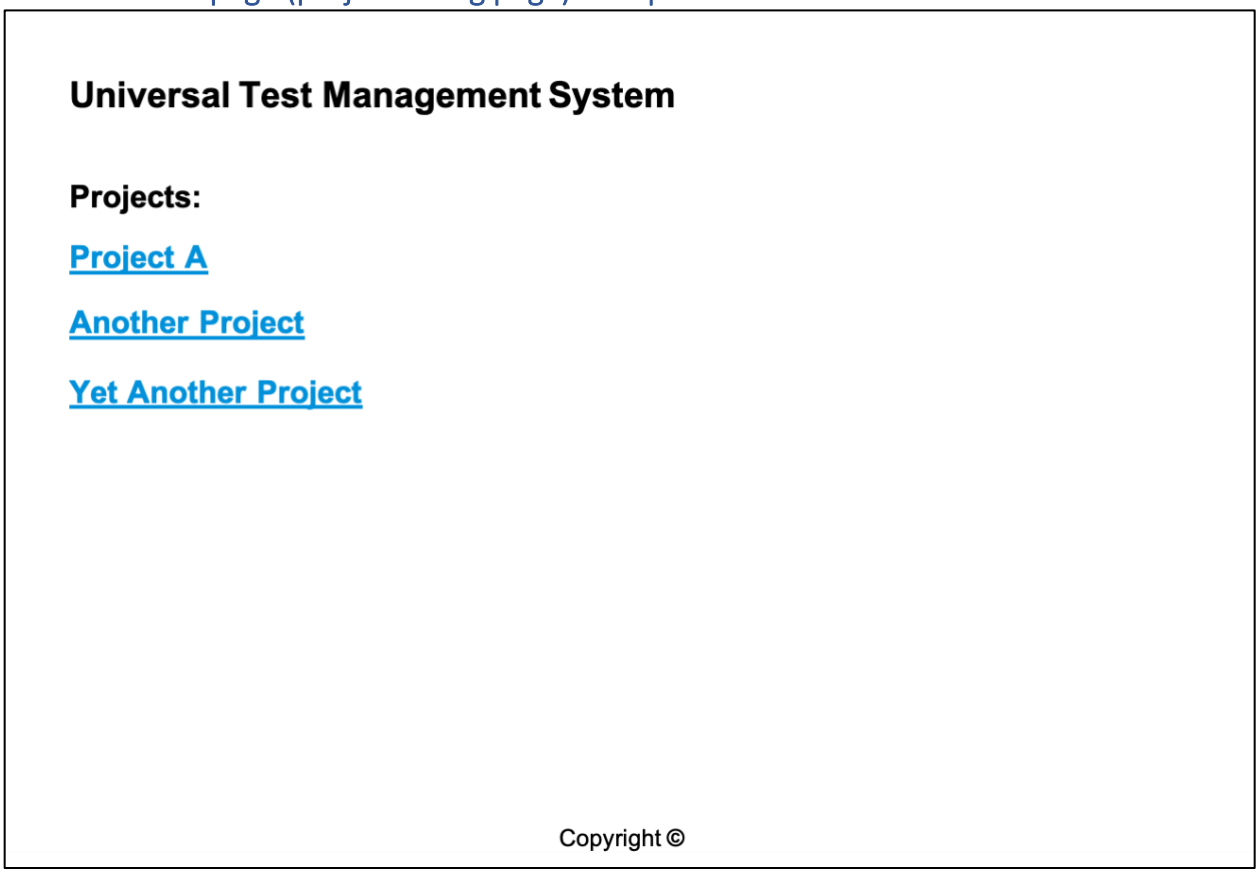
BONUS TASKS* – 10 points

Introduce a new argument:

Argument	Description	Mandatory	Default Value	Example Value
`--debug, -d`	This is an optional flag which when true it should print the report to the console instead of sending it to the server.	No	`False`	`True`

Web Based User Interface (Phase 4) – 50 points

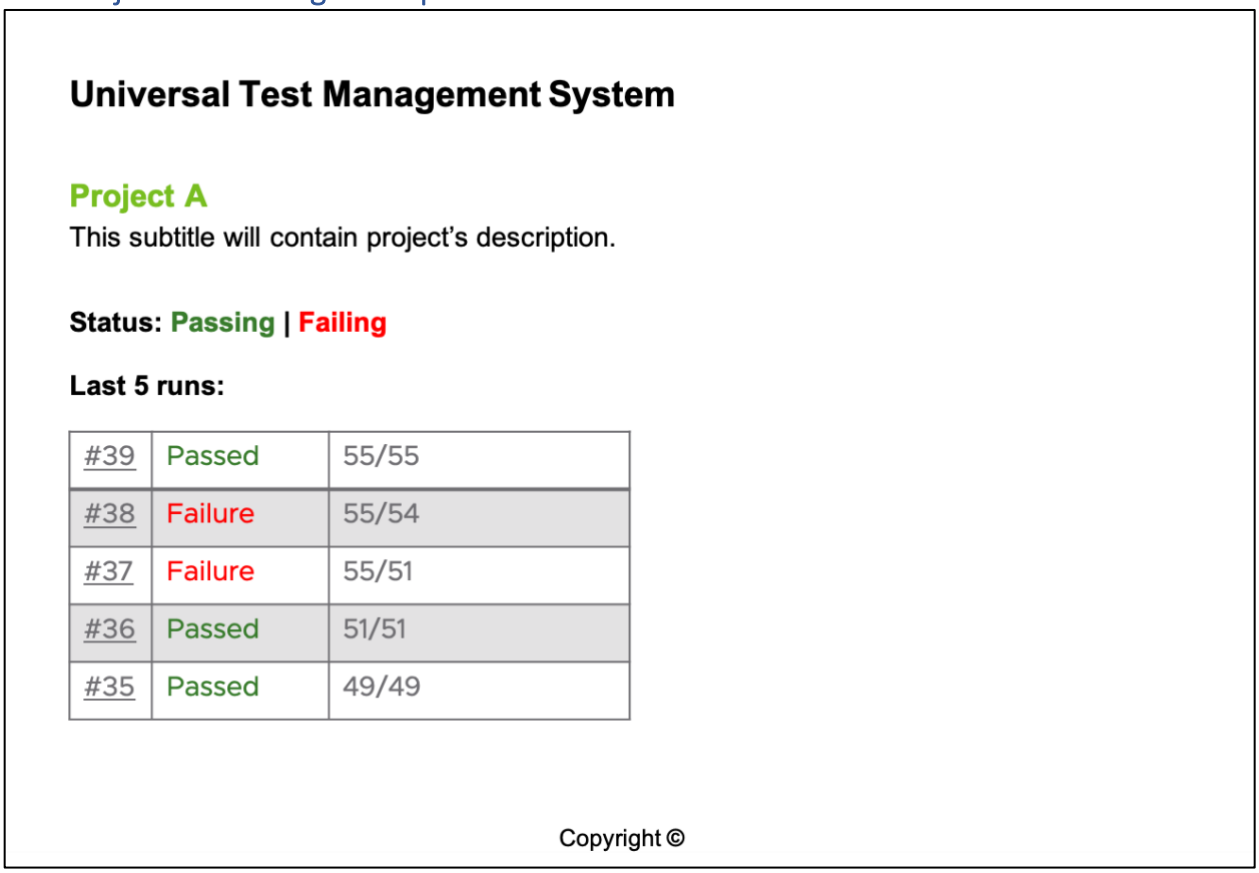
Add the home page (project listing page) – 15 points



Requirements:

- "*Universal Test Management System*" should be clickable and point to home page ("/")
- home page should list all projects
- each project should be a link that leads to their details page
- "*Universal Test Management System*" and copyright footer should be visible on every page

Add Project Details Page – 15 points



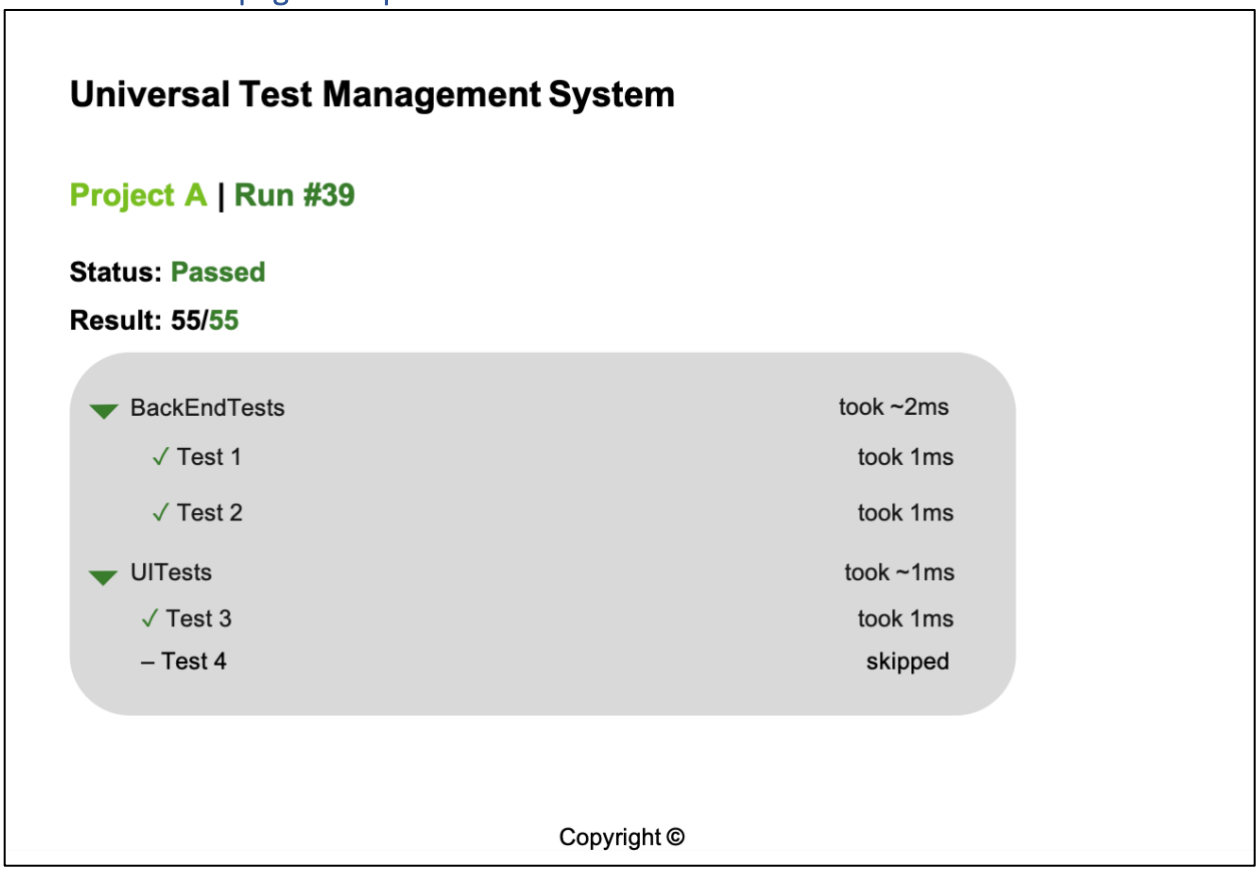
Requirements:

- show project name and description
- compute the "Status" field based on the **latest** test run
- show 5 last runs with their id, status, and pass rate (all **enabled** test / successful test)
- each id should be a link and it should lead to test run details

BONUS TASK* – 10 points

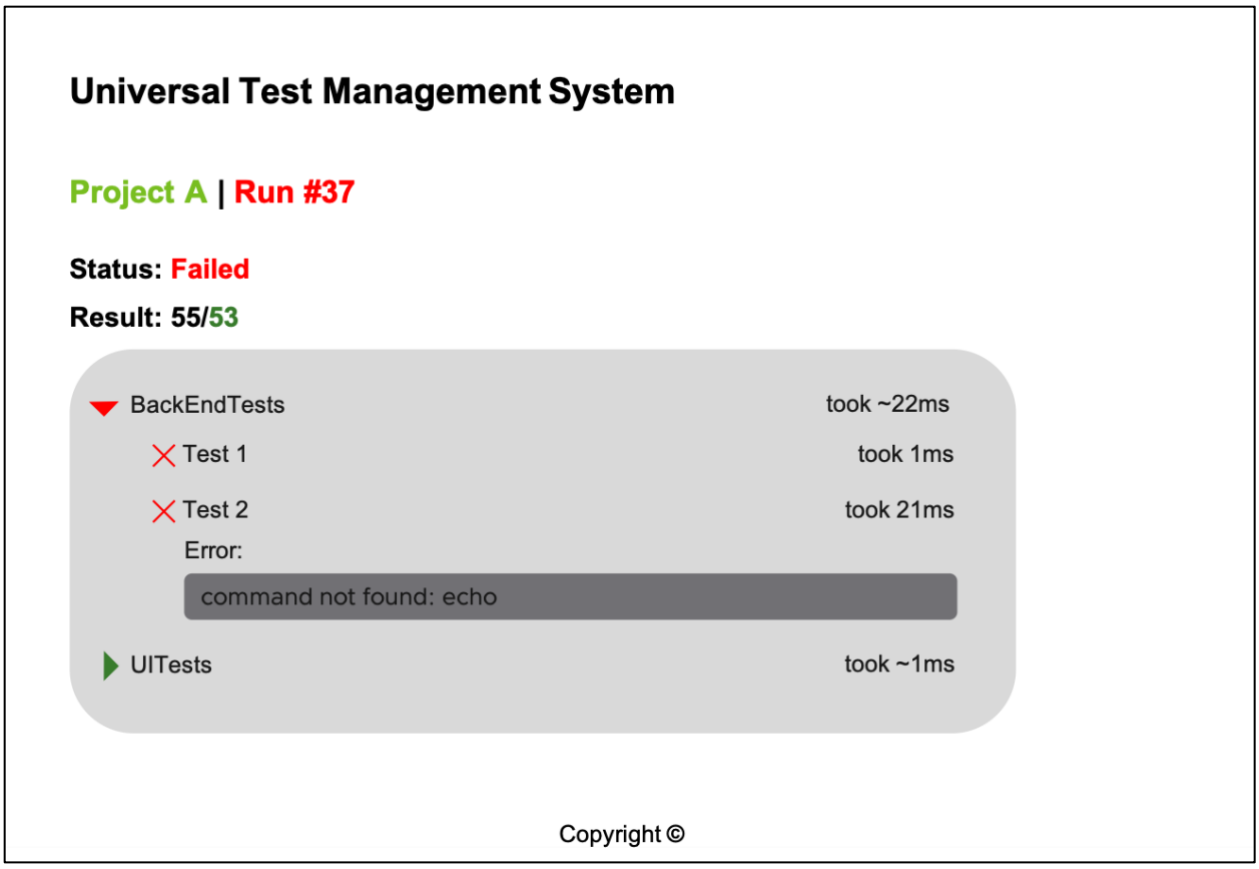
Add a "Load 5 More" button below the "Last 5 runs" table and upon click show the prior 5 runs before that. You should be able to click "Load 5 More" as many times as there are runs available.

Add the test run page – 25 points



Requirements:

- show project name, run number, run status, and ratio of **total executed tests** / **successful tests**
- display all test suites with their *name, status,* and *approximate duration* – the latter is calculated as a sum of all test cases' durations.
- provide option to "expand" a test suite and see more details about executed cases
- when expanded see test case **name, status, and duration**



Requirements:

- if the test has failed with an error, display the error

Async Execution (Optional Phase 5):

Test run status API/CLI/UI Integration** – 20 points

Endpoint	Method	User	Details
`/projects/:project_id/runs/:run_id`	PUT	CLI	When a test run is first started, the back end will create it in an "started" status. When all tests have run or the CLI has decided so, it should close the test session by doing a request to this endpoint. Action could be "passed", "failed", or "skipped." Since test runs are immutable, the back end should allow setting the status just once. If a follow-up request arrives, the status should remain the same.

You may need to modify the */project/:project_id/run POST* API to allow CLI to update run 2 times (one on run start and on run end). Make changes to the UI to reflect the new statuses.

Test case status API/CLI Integration*** – 25 points

Endpoint	Method	User	Details
`/projects/:project_id/runs/:run_id/suites/:suite_id/tests/:test_name`	POST	CLI	Create a test case as it starts. The request should include the parameters of the test—whether it is enabled (if not, it would directly be marked as skipped on the back end), the command that should be run, its owners. The start time will be calculated by the back end. The test will be created in "*started*" status, so we can view it in real time in the browser.
`/projects/:project_id/runs/:run_id/suites/suite_id/tests/:test_name/actions`	PUT	CLI	Mark a test case as complete. Action would be "*passed*", "*failed*", or "*skipped*". Once set, the back end should not allow the status to be changed. This request can also contain the log of the command that was run. The end time for the test is calculated by the back end.

Make sure this endpoint is consumed by the API as well. Make changes to the UI to reflect these new statuses.

Presentation – 30 points

You will be given a **total 20-minutes** slot to present your solution. Make sure you have everything running and be prepared to make a short demo. After or during that the examining committee will ask questions regarding different implementation or design details. Make sure your **demo** is structured and does not last longer than **12 minutes** (the rest will be used fulfilled by the committee with questions). PowerPoint presentation (or any other) is optional.

Click here to see the schedule. An individual invitation will be sent to every student. If you don't have one yet email *rstankova@vmware.com*.

FAQ

I have questions regarding the task, who to write to?

- General – #final-task-2021-discussion (slack channel)
- UI - Georgi Stoimenov (stoimenovg), Tihomir Mateev (tmateev)
- CI/CD – Ivo Marinkov (imarinkov)
- CLI/BE (Spring/DB) - Georgi Stoimenov (stoimenovg), Vasil Chomakov (vchomakov)

When is the end date for development?

- Make sure you have submitted all your changes prior to 17/05/2021 0:00. Any commits after that are considered late and will not be included in the evaluation. Any local changes that were not committed but still presented will result in severe point decrease.

What if I cannot complete the task (due to some reason)?

- Write an email to *dgindeva@vmware.com* and *stoimenovg@vmware.com*. It is important to communicate this with us as it is normal and happens. Didi most likely will follow up with you to have a friendly chat ☺.

What if I cannot present at the scheduled time?

- Write an email to *dgindeva@vmware.com* and *stoimenovg@vmware.com* and state your reason. If it is important enough, we will re-schedule for the next few days.