Core

Package: `e2e/src/test/java/eu/nets/test/core`

This package contains the core components of the test suite, creating a framework tailored to handle tests for MPA.

Drivers

- `MpaDriver`: extension of Selenium WebDriver. MpaDriver is used as a wrapper, to encapsulate and customize Appium driver features.
- `MpaAndroidDriver`: implementation of MpaDriver which extends Appium AndroidDriver.
- `MpaIOSDriver`: implementation of MpaDriver which extends Appium IOSDriver.

Using MpaDriver as a common interface, tests can launch the required driver dinamically, based on the value of the environmental variable PLATFORM (see .env file), so that each test can be written once and run on both mobile platforms. Moreover, the MpaDriver interfaces forces the developer to implement the same methods for both iOS and Android drivers, adapting the logic to fit each mobile platform, allowing a specific method to work on both platforms.

Regarding custom functionalities, for example, the `safeClick` method combines the `wait` functionality and the `click` method included in Appium drivers, allowing to wait for an element to be visible before trying to click it, accounting for possible delays and/or performance issues, making the tests more stable and consistent. The same strategy is used for other custom methods in MPA drivers.

Enums

- `AndroidSnapshot`: used to map MPA Android snapshots, which allow to save the state of the Android emulator, enabling the developer to skip steps of a specific MPA flow (e.g., registration to the app), in order to start a test from a specific step and make the test execution faster and more efficient. It provides methods to check, create, save, load and delete snapshots.
- `MpaLanguage`: set of languages supported by MPA via Lokalise. By combining this language list with the filters in the test data, the developer can control which languages the tests will be executed on. It includes unsupported languages as well (identified by the

constructor flag `isSupportedByMpa`), to keep track of languages that are planned to be added to MPA or handle test scenarios where the user's phone language is set to a language not supported by MPA. It provides methods to capitalize texts, map Lokalise xml folders and handle language dictionaries and related in-app labels.

- `MpaWidget`: used to map MPA UI elements (i.e, widgets) on both mobile platforms. It provides methods to identify widgets through different id or xpath strategies (e.g., resource id, accessibility id, xpath with index, xpath with attribute).
- `PathKey`: set of common directories used across the test suite. It provides methods to obtain a path as a `Path`, `String` or `File` object.

Exceptions

- `LokaliseMismatchException`: extension of `RuntimeException`, to be thrown when a mismatch occurs between expected Lokalise translations and actual in-app texts. It provides a custom message to highlight the mismatch.
- `UnsupportedPlatformException`: extension of `UnsupportedOperationException`, to be thrown when the developer tries to run tests on an unsupported mobile platform. Since the test suite supports both Android and iOS, at the moment this exception is only used to identify mistakes in the selected platform name (see .env file), and could helpful if any new mobile platform will be launched in the future.
- `UnupportedOsException`: extension of `UnsupportedOperationException`, to be thrown when the developer tries to run tests on an unsupported operating system. Currently, the supported OSs are Windows and MacOS.

AbstractFlow

Abstract class aiming to provide a common template for flow tests. It is the base class for all MPA UI tests executed via Appium. It provides shared setup and teardown logic, Appium driver initialization, environment handling, and integration with tools such as Allure, Lokalise, and emulator/simulator management.

This class is intended to be extended by specific test classes targeting Android or iOS flows (see related documentation).

Responsibilities

- Launch and shut down Android emulators or iOS simulators based on the platform under test.
- Initialize the appropriate Appium driver (`MpaAndroidDriver` or `MpaIOSDriver`).
- Download and load Lokalise translation bundles.
- Set Allure test parameters and manage Allure custom reports.
- Manage driver teardown after each test.
- Provide utility hooks like TestCallbackExtension for custom extension-based logic.

Fields

- `driver` : instance of `MpaDriver` , initialized per test run.
- `service` : Appium server instance launched with custom args.
- `lokaliseBundle` : contains downloaded translations, mapped by language.
- `reportDir` : directory path where Allure custom reports are stored.
- `testCallbackExtension` : JUnit 5 extension to hook into test events and enhance logging/reporting.
- `APP_PIN` , `APP_PIN_CHANGE` , `GMAIL_APP_PSW` : credentials and configuration for app login and email validation.

Driver initialization methods (`launchDriver`)

These overloads provide flexibility when initializing the driver:

- Automatically detects platform via `EnvUtil` .
- Supports Appium driver capability `"appium:app"` via `setAppiumApp` .
- Supports localization via `language` and `locale` .

JUnit Lifecycle methods

- `beforeAll()` : sets up emulator/simulator, starts Appium server, downloads Lokalise bundle.
- `beforeEach()` : applies a 20s delay before every test, to avoid driver/service/emulator/simulator connection issues.
- `afterEach()` : quits the Appium driver after each test and resets driver state.

- `afterAll()`: stops emulator/simulator, stops Appium service, generates Allure report, and clears previous results.

TestCallbackExtension

Implementation of JUnit interface AfterTestExecutionCallback. It overrides the method `afterTestExecution`, which is called immediately after an individual test has been executed, but before any user-defined teardown methods (e.g., @AfterEach methods) have been executed for that test. This allows to perform end-of-test operations, such as taking a screenshot only when the test has failed. The information regarding the outcome of a test is provided by the `ExtensionContext` object passed to `afterTestExecution` as an input parameter.

This class can be customized further, by implementing other interfaces included in the package `org.junit.jupiter.api.extension`, which allow to perform the callback at different stages of a test (before, after, etc.).
