1. One test per test class (see VerifySecureZoneViaSecureZoneLoginFormTest)
2. All tests should inherit from BaseTest. There is not much in BaseTest right now but it will provide us with extension points if we want to add functionality to all tests at once.
3. One or more models per test to hold data necessary for the test. Model classes should extend DataServiceModelBase. For example VerifySecureZoneViaSecureZoneLoginFormTest has two models.

* VerifySecureZoneLoginBaseTest.TestModel model;
* Use internal classes for models that represent the data for a specific class.
* AdminLoginModel loginModel;
* Use shared classes for models that can be reused.
* Take care when setting up persistence flag on model fields. Persistence set to true (enabled by default) means that those values are saved and retrieved from the service. This should be set only on those fields for which we create data in @Before test. For example CreateNewSecureZoneTest creates only two pages in @Before test and hence only those page names are persistent. The rest of the data is generated and recreated in each test.

1. @Before methods should setup the environment (to see if it is the case to create data should check model.shouldCreateTestData).

* There are three kinds of setup: full setup, partner setup and site setup.
* Full Setup: creates a partner, creates a site, and creates all site data
* Partner Setup: assumes a partner is already created and just creates a site and the corresponding site data
* Site Setup: only creates site data

1. Currently we have only sample tests that do site setup, and need some login data to start with. This common login data is provided by the configuration of GlobalConfigTest

* How does a test reads the emailAddresss and password from GlobalConfigTest?
* Use @DataServiceModel(fallbackConfigTest = GlobalConfigTest.class) for loginModel. The loginModel values will be copied from GlobalConfigTest if they are not stored against the current test.
* How do we populate the values for GlobalConfigTest?
* Run GlobalConfigTest after replacing the values in its datamodel with those that you want in the database. At first run the values will be saved in the data service.

1. To force the creation of new data run the test with parameter forceCreateData=true. This will generate new data even if we have existing data in the data service.

* mvn test –DforceCreateData=true –Dtest= VerifySecureZoneViaSecureZoneLoginFormTest

1. Admin Workflows:

* Commands that are used to prepare the setup should be exposed as workflows.
* Workflows are series of steps (decorated with @StepGroup) that are rerunable and always start by navigating the admin menu. Their goal is not to test functionality but to be reusable in order to easily setup tests.

1. One test per test class (see VerifySecureZoneViaSecureZoneLoginFormTest)
2. All tests should inherit from BaseTest. There is not much in BaseTest right now but it will provide us with extension points if we want to add functionality to all tests at once.
3. One or more models per test to hold data necessary for the test. Model classes should extend DataServiceModelBase. For example VerifySecureZoneViaSecureZoneLoginFormTest has two models.

* VerifySecureZoneLoginBaseTest.TestModel model;
* Use internal classes for models that represent the data for a specific class.
* AdminLoginModel loginModel;
* Use shared classes for models that can be reused.
* Take care when setting up persistence flag on model fields. Persistence set to true (enabled by default) means that those values are saved and retrieved from the service. This should be set only on those fields for which we create data in @Before test. For example CreateNewSecureZoneTest creates only two pages in @Before test and hence only those page names are persistent. The rest of the data is generated and recreated in each test.

1. @Before methods should setup the environment (to see if it is the case to create data should check model.shouldCreateTestData).

* There are three kinds of setup: full setup, partner setup and site setup.
* Full Setup: creates a partner, creates a site, and creates all site data
* Partner Setup: assumes a partner is already created and just creates a site and the corresponding site data
* Site Setup: only creates site data

1. Currently we have only sample tests that do site setup, and need some login data to start with. This common login data is provided by the configuration of GlobalConfigTest

* How does a test reads the emailAddresss and password from GlobalConfigTest?
* Use @DataServiceModel(fallbackConfigTest = GlobalConfigTest.class) for loginModel. The loginModel values will be copied from GlobalConfigTest if they are not stored against the current test.
* How do we populate the values for GlobalConfigTest?
* Run GlobalConfigTest after replacing the values in its datamodel with those that you want in the database. At first run the values will be saved in the data service.

1. To force the creation of new data run the test with parameter forceCreateData=true. This will generate new data even if we have existing data in the data service.

* mvn test –DforceCreateData=true –Dtest= VerifySecureZoneViaSecureZoneLoginFormTest

1. Admin Workflows:

* Commands that are used to prepare the setup should be exposed as workflows.
* Workflows are series of steps (decorated with @StepGroup) that are rerunable and always start by navigating the admin menu. Their goal is not to test functionality but to be reusable in order to easily setup tests.

1. One test per test class (see VerifySecureZoneViaSecureZoneLoginFormTest)
2. All tests should inherit from BaseTest. There is not much in BaseTest right now but it will provide us with extension points if we want to add functionality to all tests at once.
3. One or more models per test to hold data necessary for the test. Model classes should extend DataServiceModelBase. For example VerifySecureZoneViaSecureZoneLoginFormTest has two models.

* VerifySecureZoneLoginBaseTest.TestModel model;
* Use internal classes for models that represent the data for a specific class.
* AdminLoginModel loginModel;
* Use shared classes for models that can be reused.
* Take care when setting up persistence flag on model fields. Persistence set to true (enabled by default) means that those values are saved and retrieved from the service. This should be set only on those fields for which we create data in @Before test. For example CreateNewSecureZoneTest creates only two pages in @Before test and hence only those page names are persistent. The rest of the data is generated and recreated in each test.

1. @Before methods should setup the environment (to see if it is the case to create data should check model.shouldCreateTestData).

* There are three kinds of setup: full setup, partner setup and site setup.
* Full Setup: creates a partner, creates a site, and creates all site data
* Partner Setup: assumes a partner is already created and just creates a site and the corresponding site data
* Site Setup: only creates site data

1. Currently we have only sample tests that do site setup, and need some login data to start with. This common login data is provided by the configuration of GlobalConfigTest

* How does a test reads the emailAddresss and password from GlobalConfigTest?
* Use @DataServiceModel(fallbackConfigTest = GlobalConfigTest.class) for loginModel. The loginModel values will be copied from GlobalConfigTest if they are not stored against the current test.
* How do we populate the values for GlobalConfigTest?
* Run GlobalConfigTest after replacing the values in its datamodel with those that you want in the database. At first run the values will be saved in the data service.

1. To force the creation of new data run the test with parameter forceCreateData=true. This will generate new data even if we have existing data in the data service.

* mvn test –DforceCreateData=true –Dtest= VerifySecureZoneViaSecureZoneLoginFormTest

1. Admin Workflows:

* Commands that are used to prepare the setup should be exposed as workflows.
* Workflows are series of steps (decorated with @StepGroup) that are rerunable and always start by navigating the admin menu. Their goal is not to test functionality but to be reusable in order to easily setup tests.

1. One test per test class (see VerifySecureZoneViaSecureZoneLoginFormTest)
2. All tests should inherit from BaseTest. There is not much in BaseTest right now but it will provide us with extension points if we want to add functionality to all tests at once.
3. One or more models per test to hold data necessary for the test. Model classes should extend DataServiceModelBase. For example VerifySecureZoneViaSecureZoneLoginFormTest has two models.

* VerifySecureZoneLoginBaseTest.TestModel model;
* Use internal classes for models that represent the data for a specific class.
* AdminLoginModel loginModel;
* Use shared classes for models that can be reused.
* Take care when setting up persistence flag on model fields. Persistence set to true (enabled by default) means that those values are saved and retrieved from the service. This should be set only on those fields for which we create data in @Before test. For example CreateNewSecureZoneTest creates only two pages in @Before test and hence only those page names are persistent. The rest of the data is generated and recreated in each test.

1. @Before methods should setup the environment (to see if it is the case to create data should check model.shouldCreateTestData).

* There are three kinds of setup: full setup, partner setup and site setup.
* Full Setup: creates a partner, creates a site, and creates all site data
* Partner Setup: assumes a partner is already created and just creates a site and the corresponding site data
* Site Setup: only creates site data

1. Currently we have only sample tests that do site setup, and need some login data to start with. This common login data is provided by the configuration of GlobalConfigTest

* How does a test reads the emailAddresss and password from GlobalConfigTest?
* Use @DataServiceModel(fallbackConfigTest = GlobalConfigTest.class) for loginModel. The loginModel values will be copied from GlobalConfigTest if they are not stored against the current test.
* How do we populate the values for GlobalConfigTest?
* Run GlobalConfigTest after replacing the values in its datamodel with those that you want in the database. At first run the values will be saved in the data service.

1. To force the creation of new data run the test with parameter forceCreateData=true. This will generate new data even if we have existing data in the data service.

* mvn test –DforceCreateData=true –Dtest= VerifySecureZoneViaSecureZoneLoginFormTest

1. Admin Workflows:

* Commands that are used to prepare the setup should be exposed as workflows.
* Workflows are series of steps (decorated with @StepGroup) that are rerunable and always start by navigating the admin menu. Their goal is not to test functionality but to be reusable in order to easily setup tests.