Edit

Testing CreatePerson Method

You can skip this section if you don't interest in **automated testing**.

We can create a unit test method to test CreatePerson method as shown below:

```
[Fact]
public async Task Should_Create_Person_With_Valid_Arguments()
{
    //Act
    await _personAppService.CreatePerson(
        new CreatePersonInput
        {
            Name = "John",
            Surname = "Nash",
            EmailAddress = "john.nash@abeautifulmind.com"
        });

    //Assert
    UsingDbContext(
        context =>
        {
            var john = context.Persons.FirstOrDefault(p => p.EmailAddress == "john.
            john.ShouldNotBe(null);
            john.Name.ShouldBe("John");
        });
}
```

Test method also written using **async/await** pattern since calling method is async. We called CreatePerson method, then checked if given person is in the database. **UsingDbContext** method is a helper method of **AppTestBase** class (which we inherited this unit test class from). It's used to easily get a reference to DbContext and use it directly to perform database operations.

This method successfully works since all required fields are supplied. Let's try to create a test for **invalid arguments**:

```
{
    //Act and Assert
    await Assert.ThrowsAsync<AbpValidationException>(
        async () =>
              {
                  await _personAppService.CreatePerson(
                      new CreatePersonInput
                      {
                          Name = "John"
                      });
              });
}
```

We did not set **Surname** property of CreatePersonInput despite it being **required**. So, it throws **AbpValidationException** automatically. Also, we can not send null to CreatePerson method since validation system also checks it. This test calls CreatePerson with invalid arguments and asserts that it throws AbpValidationException. See validation document for more information.

Next

- Creating Modal for New Person