**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**PROGRAMŲ SISTEMŲ TESTAVIMAS (T120B162)**

**Unit testai**

Atliko: IFF – 4/3 gr. studentas

Rokas Vaitkevičius

**KAUNAS 2017**

Turinys

[Turinys 2](#_Toc493358816)

[1. Įvadas 3](#_Toc493358817)

[2. Tikslas 3](#_Toc493358818)

[3. Ranka generuoti testai 3](#_Toc493358819)

[4. Testai 3](#_Toc493358820)

[4.1 EventCases testai 3](#_Toc493358821)

[4.1.1 Testas 1 3](#_Toc493358822)

[4.1.2 Testas 2 5](#_Toc493358823)

[4.1.3 Testas 3 7](#_Toc493358824)

[4.1.4 Testas 4 9](#_Toc493358825)

[4.1.5 Testas 5 11](#_Toc493358826)

[4.1.6 Testas 6 13](#_Toc493358827)

[4.2 SportType testai 14](#_Toc493358828)

[4.2.1 Testas 1 14](#_Toc493358829)

[4.2.2 Testas 2 14](#_Toc493358830)

[4.3 Mappers testai 15](#_Toc493358831)

[4.3.1 Testas 1 15](#_Toc493358832)

[4.3.2 Testas 2 17](#_Toc493358833)

[4.3.3 Testas 3 18](#_Toc493358834)

[4.3.4 Testas 4 19](#_Toc493358835)

[5. Generuoti testai 20](#_Toc493358836)

[5.1 UserCases sugeneruotas testas 20](#_Toc493358837)

[5.2 VoterCases sugeneruotas kodas 21](#_Toc493358838)

[6. Testų dengimas 22](#_Toc493358839)

[7. Išvados 22](#_Toc493358840)

# Įvadas

Užduotis yra prašyti ir sugeneruoti testus. Taip pat patikrinti kiek kodo dengia tie testai. Laboratorinio darbo metu parašyti 8 integraciniai testai ir 4 unit testai.

# Tikslas

Susipažinti su testų rašymų ir automatiniu jų generavimu.

# Ranka generuoti testai

Testai parašyti testuoti .net API projekto veikimą. Testavimui naudotasi xUnit testavimo karkasu. Ranka parašyti visi testai, kurie buvo aprašyti testavimo plane.

# Testai

## EventCases testai

### Testas 1

1. Testas patikrina ar buvo sukurtas renginys.
2. Sukurti renginį ir jį paimti iš atmintyje laikomos duomenų bazės ir patikrinti ar jis egzistuoja.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| EventsValidationCollection – renginių kolekcija. | Egzistuojantis duomenų bazės objektas. |

Kodas

|  |
| --- |
| EventsValidationCollection.cs |
| using Events.Api.Dto.Events;  using Events.Api.Dto.Users;  using Events.Constants.Constants;  using Events.TestHelpers.Data;  using System;  namespace Events.EventRepositoryCasesTests.EventsCases.TestData  {  public class EventsValidationDataCollection : ClassDataBase<NewUser, NewEvent>  {  public EventsValidationDataCollection() :  base(new[]  {  Create("username", "email", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, null, 1, 1),  Create("usernames", "emailaaas", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", "https://fb", "http://image", 1, 1),  Create("usernamsse", "email@aaa", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, "http://image", 1, 1)  })  {  }  private static Tuple<NewUser, NewEvent> Create(string username, string email, string firstName, string lastName, string password,  string name, DateTime eventDate, string timeFrom, string timeTill, double price, string phoneNumber, string address,  string city, string country, string description, string facebookEventUrl , string imageUrl, int userId, int sportTypeId)  {  return new Tuple<NewUser, NewEvent>(  new NewUser  {  UserName = username,  Email = email,  FirstName = firstName,  LastName = lastName,  Password = password  },  new NewEvent  {  Name = name,  EventDate = eventDate,  TimeFrom = timeFrom,  TimeTill = timeTill,  Price = price,  PhoneNumber = phoneNumber,  Location = new Location  {  Address = address,  City = city,  Country = country  },  Description = description,  FacebookEventUrl = facebookEventUrl,  ImageUrl = imageUrl,  UserId = userId,  SportTypeId = sportTypeId  });  }  }  } |

|  |
| --- |
| Testas |
| [Theory, ClassData(typeof(EventsValidationDataCollection))]  public async void ItFindsCreatedEvent\_WhenEventIsCreated(NewUser newUser, NewEvent newEvent)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var userCases = iocContainer.GetService<IUserCases>();  var eventCases = iocContainer.GetService<IEventCases>();  // Act:  var userId = await userCases.CreateUser(newUser);  newEvent.UserId = userId;  var eventId = await eventCases.CreateEvent(newEvent);  // Assert:  var retrievedEvent = await eventCases.GetEventById(eventId);  Assert.NotNull(retrievedEvent);  } |

### Testas 2

1. Testas patikrina ar teisingai užpildomi visi duomenų laukai.
2. Sukuriamas renginys ir lyginami jau sukurto renginio duomenys su duotais testo duomenimis.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| EventsValidationCollection – renginių kolekcija. | Renginio objektas su teisingais duomenimis. |

Kodas

|  |
| --- |
| EventsValidationCollection.cs |
| using Events.Api.Dto.Events;  using Events.Api.Dto.Users;  using Events.Constants.Constants;  using Events.TestHelpers.Data;  using System;  namespace Events.EventRepositoryCasesTests.EventsCases.TestData  {  public class EventsValidationDataCollection : ClassDataBase<NewUser, NewEvent>  {  public EventsValidationDataCollection() :  base(new[]  {  Create("username", "email", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, null, 1, 1),  Create("usernames", "emailaaas", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", "https://fb", "http://image", 1, 1),  Create("usernamsse", "email@aaa", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, "http://image", 1, 1)  })  {  }  private static Tuple<NewUser, NewEvent> Create(string username, string email, string firstName, string lastName, string password,  string name, DateTime eventDate, string timeFrom, string timeTill, double price, string phoneNumber, string address,  string city, string country, string description, string facebookEventUrl , string imageUrl, int userId, int sportTypeId)  {  return new Tuple<NewUser, NewEvent>(  new NewUser  {  UserName = username,  Email = email,  FirstName = firstName,  LastName = lastName,  Password = password  },  new NewEvent  {  Name = name,  EventDate = eventDate,  TimeFrom = timeFrom,  TimeTill = timeTill,  Price = price,  PhoneNumber = phoneNumber,  Location = new Location  {  Address = address,  City = city,  Country = country  },  Description = description,  FacebookEventUrl = facebookEventUrl,  ImageUrl = imageUrl,  UserId = userId,  SportTypeId = sportTypeId  });  }  }  } |

|  |
| --- |
| Testas |
| [Theory, ClassData(typeof(EventsValidationDataCollection))]  public async void ItHasTheSameData\_WhenEventIsCreated(NewUser newUser, NewEvent newEvent)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var userCases = iocContainer.GetService<IUserCases>();  var eventCases = iocContainer.GetService<IEventCases>();  // Act:  var userId = await userCases.CreateUser(newUser);  newEvent.UserId = userId;  var eventId = await eventCases.CreateEvent(newEvent);  // Assert:  var retrievedEvent = await eventCases.GetEventById(eventId);  Assert.Equal(newEvent.Name, retrievedEvent.Name);  Assert.Equal(newEvent.EventDate, retrievedEvent.EventDate);  Assert.Equal(newEvent.TimeFrom, retrievedEvent.TimeFrom);  Assert.Equal(newEvent.TimeTill, retrievedEvent.TimeTill);  Assert.Equal(newEvent.Price, retrievedEvent.Price);  Assert.Equal(newEvent.PhoneNumber, retrievedEvent.PhoneNumber);  Assert.Equal(newEvent.Location.Address, retrievedEvent.Location.Address);  Assert.Equal(newEvent.Location.City, retrievedEvent.Location.City);  Assert.Equal(newEvent.Location.Country, retrievedEvent.Location.Country);  Assert.Equal(newEvent.Description, retrievedEvent.Description);  Assert.Equal(newEvent.FacebookEventUrl, retrievedEvent.FacebookEventUrl);  Assert.Equal(newEvent.ImageUrl, retrievedEvent.ImageUrl);  } |

### Testas 3

1. Testas patikrina ar pasirodo sistema iškelia klaidos pranešimą, kai bent vienas iš laukų yra neteisingas.
2. Bandyti sukurti renginį, kurie kažkuris iš laukų yra neteisingas, tikėtis kad bus iškeltas klaidos pranešimas.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| EventsInvalidDataCollection – renginių kolekcija kurios bent vienas laukas yra neteisingas. | Sistema iškels klaidos pranešimą dėl neteisingo lauko. |

Kodas

|  |
| --- |
| EventsInvalidDataCollection.cs |
| using Events.Api.Dto.Events;  using Events.Api.Dto.Users;  using Events.Constants.Constants;  using Events.TestHelpers.Data;  using System;  namespace Events.EventRepositoryCasesTests.EventsCases.TestData  {  public class EventsInvalidDataCollection : ClassDataBase<NewUser, NewEvent>  {  public EventsInvalidDataCollection() :  base(new[]  {  Create("username", "email", "firstname", "latname", "paswrd", null, DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, null, 1, 1),  Create("usernames", "emailaaas", "firstname", "latname", "paswrd", "Event", DateTime.Now, null, "12", 12.05,  "8666566", "street", "city", "country", "description", "https://fb", "http://image", 1, 1),  Create("usernamsse", "email@aaa", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", null, 12.05,  "8666566", "street", "city", "country", "description", null, "http://image", 1, 1),  Create("usernamsse", "email@aaa", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", null, 12.05,  null, "street", "city", "country", "description", null, "http://image", 1, 1)  })  {  }  private static Tuple<NewUser, NewEvent> Create(string username, string email, string firstName, string lastName, string password,  string name, DateTime eventDate, string timeFrom, string timeTill, double price, string phoneNumber, string address,  string city, string country, string description, string facebookEventUrl , string imageUrl, int userId, int sportTypeId)  {  return new Tuple<NewUser, NewEvent>(  new NewUser  {  UserName = username,  Email = email,  FirstName = firstName,  LastName = lastName,  Password = password  },  new NewEvent  {  Name = name,  EventDate = eventDate,  TimeFrom = timeFrom,  TimeTill = timeTill,  Price = price,  PhoneNumber = phoneNumber,  Location = new Location  {  Address = address,  City = city,  Country = country  },  Description = description,  FacebookEventUrl = facebookEventUrl,  ImageUrl = imageUrl,  UserId = userId,  SportTypeId = sportTypeId  });  }  }  } |

|  |
| --- |
| Testas |
| [Theory, ClassData(typeof(EventsInvalidDataCollection))]  public async void ItThrowsArgumentNullException\_WhenEventIsCreated(NewUser newUser, NewEvent newEvent)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var userCases = iocContainer.GetService<IUserCases>();  var eventCases = iocContainer.GetService<IEventCases>();  // Act:  var userId = await userCases.CreateUser(newUser);  newEvent.UserId = userId;  // Assert:  await Assert.ThrowsAsync<ArgumentNullException>(async () =>  await eventCases.CreateEvent(newEvent));  } |

### Testas 4

1. Testas patikrina ar atnaujinus renginio duomenis, duomenys atsinaujina
2. Sukurti renginį, tada tą renginį atnaujinti ir tada patikrinti ar duomenys atsinaujino.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| EventsUpdateDataCollection – renginių kolekcija kurioje yra du objektai, vienas pradinių duomenų, kitas atnaujinamų duomenų. | Renginio duomenys po atnaujinimo turi teisingus duomenis. |

Kodas

|  |
| --- |
| EventsUpdateDataCollection.cs |
| using Events.Api.Dto.Events;  using Events.Api.Dto.Users;  using Events.Constants.Constants;  using Events.TestHelpers.Data;  using System;  namespace Events.EventRepositoryCasesTests.EventsCases.TestData  {  public class EventsUpdateDataCollection : ClassDataBase<NewUser, NewEvent, EventUpdate>  {  public EventsUpdateDataCollection() :  base(new[]  {  Create("username", "email", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, null, 1, 1, "Eventas", DateTime.Now,  "10", "12", 12.05,  "8666566", "streetas", "cityas", "countryas", "descriptionas", null, "http://image", 1),  Create("username", "email", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, null, 1, 1, "Eventas", DateTime.Now,  "10", "12", 12.05,  "8666566", "streetas", "cityas", "countryas", "descriptionasass", null, "http://image", 1),  Create("username", "email", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, null, 1, 1, "Eventas", DateTime.Now,  "10", "12", 12.05,  "866656as6", "streets", "citysssas", "countryas", "descriptionas", null, "http://image", 1),  })  {  }  private static Tuple<NewUser, NewEvent, EventUpdate> Create(string username, string email, string firstName,  string lastName, string password, string name, DateTime eventDate, string timeFrom, string timeTill,  double price, string phoneNumber, string address, string city, string country, string description,  string facebookEventUrl, string imageUrl, int userId, int sportTypeId, string uName, DateTime uEventData,  string uTimeFrom, string uTimeTill, double uPrice, string uPhoneNumber, string uAddress,  string uCity, string uCountry, string uDescription, string uFacebookUrl, string uImageUrl, int uSportTypeId)  {  return new Tuple<NewUser, NewEvent, EventUpdate>(  new NewUser  {  UserName = username,  Email = email,  FirstName = firstName,  LastName = lastName,  Password = password  },  new NewEvent  {  Name = name,  EventDate = eventDate,  TimeFrom = timeFrom,  TimeTill = timeTill,  Price = price,  PhoneNumber = phoneNumber,  Location = new Location  {  Address = address,  City = city,  Country = country  },  Description = description,  FacebookEventUrl = facebookEventUrl,  ImageUrl = imageUrl,  UserId = userId,  SportTypeId = sportTypeId  },  new EventUpdate  {  Name = uName,  EventDate = uEventData,  TimeFrom = uTimeFrom,  TimeTill = uTimeTill,  Price = uPrice,  PhoneNumber = uPhoneNumber,  Location = new Location  {  Address = uAddress,  City = uCity,  Country = uCountry  },  Description = uDescription,  FacebookEventUrl = uFacebookUrl,  ImageUrl = uImageUrl,  SportTypeId = uSportTypeId  });  }  }  } |

|  |
| --- |
| Testas |
| [Theory, ClassData(typeof(EventsUpdateDataCollection))]  public async void ItHasUpdatedData\_WhenEventIsUpdated(NewUser newUser, NewEvent newEvent, EventUpdate updatedEvent)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var userCases = iocContainer.GetService<IUserCases>();  var eventCases = iocContainer.GetService<IEventCases>();  // Act:  var userId = await userCases.CreateUser(newUser);  newEvent.UserId = userId;  var eventId = await eventCases.CreateEvent(newEvent);  await eventCases.UpdateEvent(eventId, updatedEvent);  // Assert:  var retrievedEvent = await eventCases.GetEventById(eventId);  Assert.Equal(updatedEvent.Name, retrievedEvent.Name);  Assert.Equal(updatedEvent.EventDate, retrievedEvent.EventDate);  Assert.Equal(updatedEvent.TimeFrom, retrievedEvent.TimeFrom);  Assert.Equal(updatedEvent.TimeTill, retrievedEvent.TimeTill);  Assert.Equal(updatedEvent.Price, retrievedEvent.Price);  Assert.Equal(updatedEvent.PhoneNumber, retrievedEvent.PhoneNumber);  Assert.Equal(updatedEvent.Location.Address, retrievedEvent.Location.Address);  Assert.Equal(updatedEvent.Location.City, retrievedEvent.Location.City);  Assert.Equal(updatedEvent.Location.Country, retrievedEvent.Location.Country);  Assert.Equal(updatedEvent.Description, retrievedEvent.Description);  Assert.Equal(updatedEvent.FacebookEventUrl, retrievedEvent.FacebookEventUrl);  Assert.Equal(updatedEvent.ImageUrl, retrievedEvent.ImageUrl);  } |

### Testas 5

1. Testas patikrins ar renginio būsena pasikeitė iškvietus tam skirtą metodą.
2. Sukurti renginį, iškviesti metodą keičiantį būseną, patikrinti ar būsena teisinga.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| EventsValidationCollection – renginių kolekcija. | Renginio laukelis cancelled po funckijos iškvietimo pasikeitė. |

Kodas

|  |
| --- |
| EventsValidationCollection.cs |
| using Events.Api.Dto.Events;  using Events.Api.Dto.Users;  using Events.Constants.Constants;  using Events.TestHelpers.Data;  using System;  namespace Events.EventRepositoryCasesTests.EventsCases.TestData  {  public class EventsValidationDataCollection : ClassDataBase<NewUser, NewEvent>  {  public EventsValidationDataCollection() :  base(new[]  {  Create("username", "email", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, null, 1, 1),  Create("usernames", "emailaaas", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", "https://fb", "http://image", 1, 1),  Create("usernamsse", "email@aaa", "firstname", "latname", "paswrd", "Event", DateTime.Now, "10", "12", 12.05,  "8666566", "street", "city", "country", "description", null, "http://image", 1, 1)  })  {  }  private static Tuple<NewUser, NewEvent> Create(string username, string email, string firstName, string lastName, string password,  string name, DateTime eventDate, string timeFrom, string timeTill, double price, string phoneNumber, string address,  string city, string country, string description, string facebookEventUrl , string imageUrl, int userId, int sportTypeId)  {  return new Tuple<NewUser, NewEvent>(  new NewUser  {  UserName = username,  Email = email,  FirstName = firstName,  LastName = lastName,  Password = password  },  new NewEvent  {  Name = name,  EventDate = eventDate,  TimeFrom = timeFrom,  TimeTill = timeTill,  Price = price,  PhoneNumber = phoneNumber,  Location = new Location  {  Address = address,  City = city,  Country = country  },  Description = description,  FacebookEventUrl = facebookEventUrl,  ImageUrl = imageUrl,  UserId = userId,  SportTypeId = sportTypeId  });  }  }  } |

|  |
| --- |
| Testas |
| [Theory, ClassData(typeof(EventsValidationDataCollection))]  public async void ItsFieldCancelHasChanged\_WhenMethodIsCalled(NewUser newUser, NewEvent newEvent)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var userCases = iocContainer.GetService<IUserCases>();  var eventCases = iocContainer.GetService<IEventCases>();  // Act:  var userId = await userCases.CreateUser(newUser);  newEvent.UserId = userId;  var eventId = await eventCases.CreateEvent(newEvent);  var oldRetrievedEvent = await eventCases.GetEventById(eventId);  await eventCases.ChangeEventState(eventId);  var newRetrievedEvent = await eventCases.GetEventById(eventId);  // Assert:  Assert.NotEqual(oldRetrievedEvent.Canceled, newRetrievedEvent.Canceled);  } |

### Testas 6

1. Testas patikrins ar teisingai injectintas interfeisas.
2. Iškviesti pagalbinę klasę, kuri iškvies DI, tada patikrinti ar EventCases egzistuoja.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
|  | Teisingai inject‘intas EventsCases interfeisas. |

Kodas

|  |
| --- |
| Testas |
| [Fact]  public void ItResolve\_IEventsService()  {  // Prepare:  var services = new ServiceCollection();  services.AddDbContext<EventsDbContext>(options => options.UseInMemoryDatabase());  services.AddAllDependencies();  var iocContainer = services.BuildServiceProvider();  // Act:  var cases = iocContainer.GetService<IEventCases>();  // Assert:  Assert.NotNull(cases);  } |

## SportType testai

### Testas 1

1. Testas patikrins ar paleidus sistemą sukuriamas bent vienas sporto tipas.
2. Paleidus sistemą tikimasi kad duomenų bazėje atsiras sporto tipų duomenys.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
|  | Egzistuoja bent vienas renginio tipas. |

Kodas

|  |
| --- |
| Testas |
| [Fact]  public async void ItFindsCreatedSportTypes()  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var sportTypeCases = iocContainer.GetService<ISportTypeReadOnlyCases>();  // Act:  var sportTypes = await sportTypeCases.GetAllSportTypes();  // Assert:  Assert.NotNull(sportTypes);  } |

### Testas 2

1. Testas patikrins ar teisingai injectintas interfeisas.
2. Iškviesti pagalbinę klasę, kuri iškvies DI, tada patikrinti ar SportTypesCases egzistuoja.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
|  | Teisingai inject‘intas SportTypesCases interfeisas. |

Kodas

|  |
| --- |
| Testas |
| [Fact]  public void ItResolve\_ISportTypesService()  {  // Prepare:  var services = new ServiceCollection();  services.AddDbContext<EventsDbContext>(options => options.UseInMemoryDatabase());  services.AddAllDependencies();  var iocContainer = services.BuildServiceProvider();  // Act:  var cases = iocContainer.GetService<ISportTypeReadOnlyCases>();  // Assert:  Assert.NotNull(cases);  } |

## Mappers testai

### Testas 1

1. Testas patikrins ar teisingai veikia renginių mapper’is.
2. Paimti renginių mapperį ir sumapinti duomenis iš poco į dto.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| EventPocoToDtoDataCollection – renginio poco ir tikimosi gauti dto duomenų rinkinys. | Mapperis teisingai sumappin‘o duomenis iš poco į dto. |

Kodas

|  |
| --- |
| EventPocoToDtoDataCollection.cs |
| using Events.Repository.Pocos;  using Events.TestHelpers.Data;  using System;  namespace Events.MappersTests.TestData  {  public class EventPocoToDtoDataCollection : ClassDataBase<Event>  {  public EventPocoToDtoDataCollection() : base(new []  {  Create(1, "Event", DateTime.Now, "10", "12", 12.05, "8666566", "street", "city", "country",  "description", null, null, DateTime.Now, false),  Create(2, "Eventassss", DateTime.Now, "10", "12", 1.05, "866657766", "street", "city", "country",  "description", null, "aasas", DateTime.Now, false),  Create(3, "Event", DateTime.Now, "12", "16", 2.05, "8666566", "streetas", "cityas", "countsssry",  "description", "aasasda", null, DateTime.Now, false),  })  {  }  private static Event Create(int eventId, string name, DateTime eventData, string timeFrom, string timeTill,  double price, string phoneNumber, string address, string city, string country, string description,  string faceBookUrl, string imageUrl, DateTime dateUpdated, bool canceled)  {  return new Event  {  EventId = eventId,  Name = name,  EventDate = eventData,  TimeFrom = timeFrom,  TimeTill = timeTill,  Price = price,  PhoneNumber = phoneNumber,  Address = address,  City = city,  Country = country,  Description = description,  FacebookEventUrl = faceBookUrl,  ImageUrl = imageUrl,  DateUpdated = dateUpdated,  Canceled = canceled  };  }  }  } |

|  |
| --- |
| Kodas |
| [Theory, ClassData(typeof(EventPocoToDtoDataCollection))]  public void ItChecksIf\_EventPocoIsMappedToDtoCorrectly(EventPoco eventPoco)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var mapper = iocContainer.GetService<IMapper>();  // Act:  var mappedDto = mapper.Map<EventPoco, EventDto>(eventPoco);  // Assert:  Assert.Equal(eventPoco.EventId, mappedDto.EventId);  Assert.Equal(eventPoco.Name, mappedDto.Name);  Assert.Equal(eventPoco.EventDate, mappedDto.EventDate);  Assert.Equal(eventPoco.TimeFrom, mappedDto.TimeFrom);  Assert.Equal(eventPoco.TimeTill, mappedDto.TimeTill);  Assert.Equal(eventPoco.Price, mappedDto.Price);  Assert.Equal(eventPoco.PhoneNumber, mappedDto.PhoneNumber);  Assert.Equal(eventPoco.Address, mappedDto.Location.Address);  Assert.Equal(eventPoco.City, mappedDto.Location.City);  Assert.Equal(eventPoco.Country, mappedDto.Location.Country);  Assert.Equal(eventPoco.Description, mappedDto.Description);  Assert.Equal(eventPoco.FacebookEventUrl, mappedDto.FacebookEventUrl);  Assert.Equal(eventPoco.ImageUrl, mappedDto.ImageUrl);  } |

### Testas 2

1. Testas patikrins ar teisingai veikia renginių sporto tipų mapper’is.
2. Paimti sporto tipų mapperį ir sumapinti duomenis iš poco į dto.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| SportTypePocoToDtoDataCollection –poco ir tikimosi gauti dto duomenų rinkinys. | Mapperis teisingai sumappin‘o duomenis iš poco į dto. |

Kodas

|  |
| --- |
| SportTypePocoToDtoDataCollection.cs |
| using Events.Repository.Pocos;  using Events.TestHelpers.Data;  namespace Events.MappersTests.TestData  {  public class SportTypePocoToDtoCollection : ClassDataBase<SportType>  {  public SportTypePocoToDtoCollection() :  base(new []  {  Create(1, "Tennis"),  Create(2, "Basketball"),  Create(3, "Baseball"),  })  {  }  private static SportType Create(int sportTypeId, string name)  {  return new SportType  {  SportTypeId = sportTypeId,  Name = name  };  }  }  } |

|  |
| --- |
| Testas |
| [Theory, ClassData(typeof(SportTypePocoToDtoCollection))]  public void ItChecksIf\_SportTypePocoIsMappedToDtoCorrectly(SportTypePoco sportTypePoco)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var mapper = iocContainer.GetService<IMapper>();  // Act:  var mappedDto = mapper.Map<SportTypePoco, SportTypeDto>(sportTypePoco);  // Assert:  Assert.Equal(sportTypePoco.SportTypeId, mappedDto.SportTypeId);  Assert.Equal(sportTypePoco.Name, mappedDto.Name);  } |

### Testas 3

1. Testas patikrins ar teisingai veikia vartotojų mapper’is.
2. Paimti vartotojo dto ir pažiūrėti ar jis bus teisingai sumappintas į poco

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| UserDtoToPocoDataCollection – poco ir tikimosi gauti dto duomenų rinkinys. | Mapperis teisingai sumappin‘o duomenis iš dto į poco. |

Kodas

|  |
| --- |
| UserDtoToPocoDataCollection.cs |
| using Events.Repository.Pocos;  using Events.TestHelpers.Data;  namespace Events.MappersTests.TestData  {  public class UserPocoToDtoDataCollection : ClassDataBase<User>  {  public UserPocoToDtoDataCollection() :  base(new[]  {  Create(1, "username", "email", "firstnamssse", "latnameassa", "paswrdasas"),  Create(2, "usernames", "emailaaas", "firstname", "latname", "paswrd"),  Create(3, "usernamsse", "email@aaa", "firstname", "latname", "paswrd123123")  })  {  }  private static User Create(int userId, string username, string email, string firstName, string lastName, string password)  {  return new User  {  UserId = userId,  UserName = username,  Email = email,  FirstName = firstName,  LastName = lastName,  Password = password  };  }  }  } |

|  |
| --- |
| Testas |
| [Theory, ClassData(typeof(UserPocoToDtoDataCollection))]  public void ItChecksIf\_UserPocoIsMappedToDtoCorrectly(UserPoco userPoco)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var mapper = iocContainer.GetService<IMapper>();  // Act:  var mappedDto = mapper.Map<UserPoco, UserDto>(userPoco);  // Assert:  Assert.Equal(userPoco.UserId, mappedDto.UserId);  Assert.Equal(userPoco.UserName, mappedDto.UserName);  Assert.Equal(userPoco.Email, mappedDto.Email);  Assert.Equal(userPoco.FirstName, mappedDto.FirstName);  Assert.Equal(userPoco.LastName, mappedDto.LastName);  Assert.Equal(userPoco.Password, mappedDto.Password);  } |

### Testas 4

1. Testas patikrins ar teisingai veikia balsuotojų mapper’is.
2. Paimti balsuotojo dto ir pažiūrėti ar jis bus teisingai sumappintas į poco.

|  |  |
| --- | --- |
| Duomenys | Rezultatai |
| VoterPocoToDtoDataCollection – poco ir tikimosi gauti dto duomenų rinkinys. | Mapperis teisingai sumappin‘o duomenis iš dto į poco. |

Kodas

|  |
| --- |
| VoterPocoToDtoDataCollection.cs |
| using Events.Repository.Pocos;  using Events.TestHelpers.Data;  namespace Events.MappersTests.TestData  {  public class VoterPocoToDtoDataCollection : ClassDataBase<Voter>  {  public VoterPocoToDtoDataCollection() :  base(new[]  {  Create(1, 2),  Create(2, 3),  Create(3, 1),  })  {  }  private static Voter Create(int voterId, int eventId)  {  return new Voter  {  VoterId = voterId,  EventId = eventId  };  }  }  } |

|  |
| --- |
| Testas |
| [Theory, ClassData(typeof(VoterPocoToDtoDataCollection))]  public void ItChecksIf\_VoterIsMappedToDtoCorrectly(VoterPoco voterPoco)  {  // Prepare:  var iocContainer = DependencyInjectionHelper.CreateAndConfigureContainer();  var mapper = iocContainer.GetService<IMapper>();  // Act:  var mappedDto = mapper.Map<VoterPoco, VoterDto>(voterPoco);  // Assert:  Assert.Equal(voterPoco.UserId, mappedDto.UserId);  Assert.Equal(voterPoco.EventId, mappedDto.EventId);  } |

# Generuoti testai

Testų generavimui pasirinkau Unit Test Boilerplate Generator plėtinį Visual Studio. Šis įskiepis automatiškai sukuria testus nurodytoms klasėms.

## UserCases sugeneruotas testas

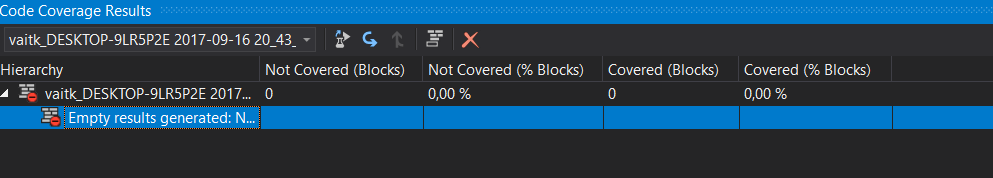
|  |
| --- |
| Kodas |
| using AutoMapper;  using Events.Api.Cases.Users;  using Events.Domain.Factories.User;  using Events.Repository.User;  using Microsoft.VisualStudio.TestTools.UnitTesting;  using Moq;  namespace Events.AutoGeneratedTests.Users  {  [TestClass]  public class UserCasesTests  {  private MockRepository mockRepository;  private Mock<IUserFactory> mockUserFactory;  private Mock<IUserRepository> mockUserRepository;  private Mock<IMapper> mockMapper;  [TestInitialize]  public void TestInitialize()  {  this.mockRepository = new MockRepository(MockBehavior.Strict);  this.mockUserFactory = this.mockRepository.Create<IUserFactory>();  this.mockUserRepository = this.mockRepository.Create<IUserRepository>();  this.mockMapper = this.mockRepository.Create<IMapper>();  }  [TestCleanup]  public void TestCleanup()  {  this.mockRepository.VerifyAll();  }  [TestMethod]  public void TestMethod1()  {  UserCases userCases = this.CreateUserCases();  }  private UserCases CreateUserCases()  {  return new UserCases(  this.mockUserFactory.Object,  this.mockUserRepository.Object,  this.mockMapper.Object);  }  }  } |

## VoterCases sugeneruotas kodas

|  |
| --- |
| Kodas |
| using Events.Api.Cases.Voter;  using Events.Domain.Factories.Voter;  using Events.Repository.Voter;  using Microsoft.VisualStudio.TestTools.UnitTesting;  using Moq;  namespace Events.AutoGeneratedTests.Voter  {  [TestClass]  public class VoterCasesTests  {  private MockRepository mockRepository;  private Mock<IVoterRepository> mockVoterRepository;  private Mock<IVoterFactory> mockVoterFactory;  [TestInitialize]  public void TestInitialize()  {  this.mockRepository = new MockRepository(MockBehavior.Strict);  this.mockVoterRepository = this.mockRepository.Create<IVoterRepository>();  this.mockVoterFactory = this.mockRepository.Create<IVoterFactory>();  }  [TestCleanup]  public void TestCleanup()  {  this.mockRepository.VerifyAll();  }  [TestMethod]  public void TestMethod1()  {  VoterCases voterCases = this.CreateVoterCases();  }  private VoterCases CreateVoterCases()  {  return new VoterCases(  this.mockVoterRepository.Object,  this.mockVoterFactory.Object);  }  }  } |

# Testų dengimas

Deja, bet nepavyko gauti testų kodo dengimo. Kadangi, .net core pakankamai nauja technologija, tai ji neturi paruoštų įrankių gauti kodo dengimo informacijai. Naujausioje VS17 versijoje atsirado code coverage funkcija, bet ji neveikia su xUnit testais (pav 1.), todėl negalėjau gauti rezultatų.



1 pav. Neveikiantis code coverage

# Išvados

Galbūt būtų įmanoma sugeneruoti 100% kodą dengiančius testus su Unit Test Boilerplate Generator, tačiau šių testų sukurtas šablonas nelabai padeda. Kadangi pasirinkau xUnit karkasą, o jam testų generatoriaus neradau, tai šio įskiepio sugeneruoti testai išvis netiktų. Be to pagrindiniai šio projekto testai turėtų būti integraciniai, nes jie padeda ištestuoti visus programos sluoksnius ir bendra jos veikimą. Taigi, rašant testus ranka prireiktų daug laiko, o generavimas šiuo atveju neišeitis, todėl 100% dengiantys testai yra per didelė investicija. Taip pat man kol kas sunku prašyti testus, kuris dengtų didelę dalį kodo, nes kodo bazė nuolatos kinta.