1. Проектно-технологическая часть
   1. Проектирование начального и тестового наполнения базы данных.

Так как в качестве источника данных выступает документ Microsoft Office Excel, необходимо наполнить таблицы данными вручную.

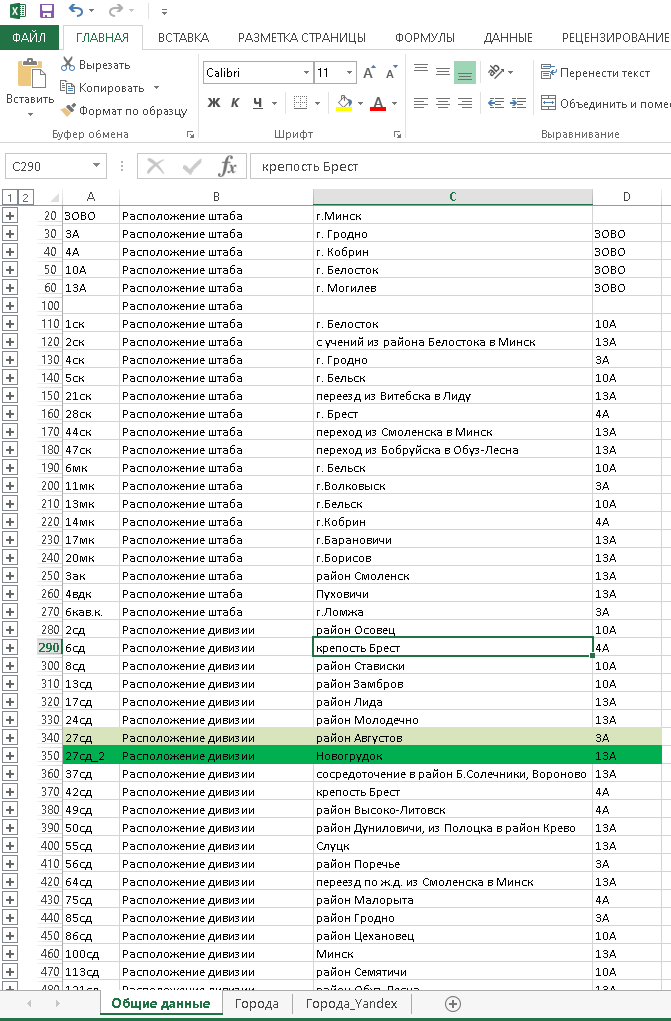


Рисунок – тестовое наполнение данными

В качестве модели данных выступает сущность «Подразделение» с полями:

- Расположение штаба

- Название подразделения

- Командующий

- Состав

- Численность

- Удостоверяющий документ

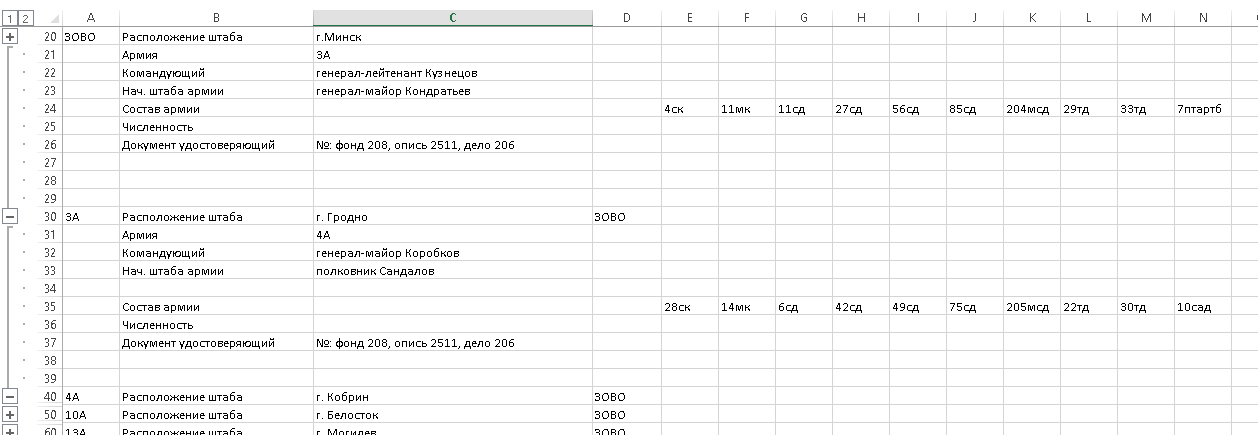


Рисунок – соответствие модели

Для задания местоположения подразделения на карте, необходимо иметь координаты городов. Для этого была введена сущность «Местоположение» с атрибутами:

- Название города;

- Координата X на карте;

- Координата Y на карте.

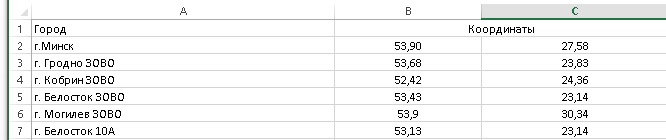


Рисунок –наполнение данными

Для определения координат города был использован сервис yandex-map-constructor-location-tools.



Рисунок – начальная страница сервиса

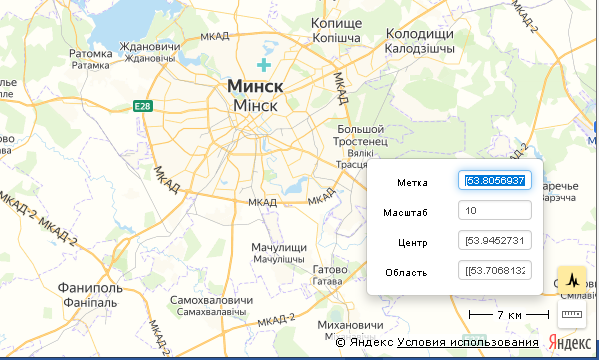


Рисунок – пример определения координат

* 1. Тестирование и отладка рабочей программы.

До запуска приложения в производство, когда оно станет доступнопользователям, важно убедиться, что данное приложение функционирует, как и должно, что в нем нет ошибок. Для проверки приложения мы можем использовать различные схемы и механизмы тестирования. Одним из таких механизмов являются юнит-тесты.

Юнит-тесты позволяют быстро и автоматически протестировать отдельные компоненты приложения независимо от остальной его части. Не всегда юнит-тесты могут покрыть весь код приложения, но тем не менее они позволяют существенно уменьшить количество ошибок уже на этапе разработки.

Для написания юнит-тестов используется фреймворк NUnit.

**NUnit**: портированный фреймворк для платформы .NET

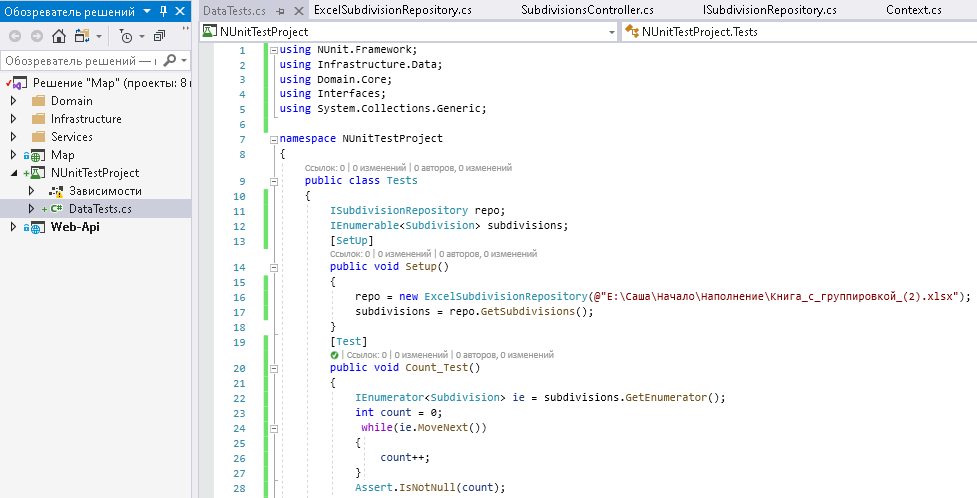


Рисунок – созданный проект для тестов

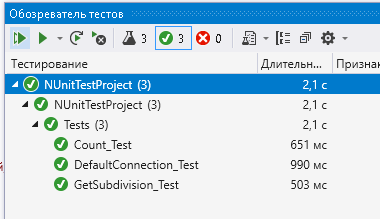


Рисунок – успешное прохождение тестов

К выявленным требованиям к хранилищу данных можно отнести следующие:

1. Наличие существующего хранилища данных;
2. Наличие существующего непустого хранилища данных;

3) Обязательность возвращения объекта согласно выявленной модели.

Отдельно стоит сказать о концепции TDD или разработка через тестирование. TDD представляет процесс применения юнит-тестов, при котором сначала пишутся тесты, а потом уже программный код, достаточный для выполнения этих тестов.

Использование TDD позволяет снизить количество потенциальных багов в приложении. Создавая тесты перед написанием кода, мы тем самым описываем способ поведения будущих компонентов, не связывая себя при этом с конкретной реализацией этих тестируемых компонентов (тем более что реализация на момент создания теста еще не существует). Таким образом, тесты помогают оформить и описать API будущих компонентов.

Порядок написания кода при TDD довольно прост:

* 1. Пишем юнит-тест
  2. Запускаем его и видим, что он завершился неудачей (программный код ведь еще не написан)
  3. Пишем некоторое количество кода, достаточное для запуска теста
  4. Снова запускаем тест и видим его результаты

Этот цикл повторяется снова и снова, пока не будет закончена работа над программным кодом. Так как большинство фреймворков юнит-тестирования помечают неудавшиеся тесты с красного цвета (например, выводится текст красного цвета), а удачный тест отмечается зеленым цветом (опять же выводится текст зеленого цвета), то данный цикл часто называют красным/зеленым циклом.