УДК 681.3: 658.56

Н.Н. Глибовец, Р.И. Заболотный, И.О. Завадский

Разработка системы учебного тестирования на базе G*oogle Docs API*

Описано использование *Google Docs* в виде серверной платформы для распределенных приложений на примере реализации системы учебного тестирования знаний в виде специализированного приложения, построенного на базе *Google Docs API.*

The use of the Google Docs as a server platform for the distributed applications at the example of a system of the educational testing of knowledges, implemented as a specialized application built on the basis of Google Docs API is described.

Описано апробацію *Google Docs* як серверної платформи для розподілених застосувань на прикладі реалізації системи навчального тестування знань у вигляді спеціалізованого застосування, побудованого на базі *Google Docs API.*

Введение. Современный процесс перехода от индустриального к информационному обществу требует существенных изменений во многих сферах деятельности человека. В первую очередь это касается реформирования образования: информация и теоретические знания становятся приоритетным стратегическим ресурсом постиндустриального общества. Основной информационный ресурс сейчас – Интернет.

Одним из преимуществ компьютеризации образования есть именно онлайн-документы. Исчезает необходимость переносить документ между разными компьютерами. Он доступен со всех компьютеров, имеющих доступ к Интернету. Исчезает необходимость думать о версионности документа. Онлайн-документы менее уязвимы и к потерям информации и документов в целом. Система контроля версий позволяет восстановить состояние документа на любую дату. Важен является и уровень защищенности онлайн-документов. Поскольку документ хранится только на сервере, появляется реальная возможность контролировать права доступа тех, кто взаимодействует с документом. Это дает возможность гибко организовывать взаимодействие учащихся с преподавателем. Появляется возможность вынести определенное количество работы за пределы урока. Работа с онлайн-документами позволяет эффективно проводить тестирование, опросы, самостоятельную или групповую работу учеников.

Работа с онлайн-документами, естественно, вызывает достаточно много трудностей [1]. В первую очередь – это необходимость стабильного и быстрого соединения с Интернет. Но эти проблемы носят временный характер.

Система тестирования *GTester*

Уточним постановку задачи. Используя *Google Docs Spreadsheets* [2–4] разработаем приложение, которое позволяло бы эффективно организовывать и проводить тестирование знаний учащихся в избранной предметной области.

Рассмотрим простой рабочий вариант функционала системы тестирования.

Работа приложения должна базироваться на анализе двух типов документов – регистрационного и тестового. В первом будет храниться вся необходимая регистрационная (административная) информация, во втором – все, что касается контекста теста. Необходимо также обеспечить возможность параллельной работы с документами, достаточную защищенность и удобство.

Регистрационный документ содержит следующие таблицы: *учителя*, *школы*, *группы*, *тесты*, *категории тестов*, *регионы.* С ним может работать администратор через веб-доступ, преподаватель и ученик с помощью приложения. Этот документ – ключевой в системе тестирования и должен быть единым для всех. Адрес регистрационного документа жестко прописывается в приложениях при настройке.

Тестовый документ содержит набор задач (вопросов) отдельного тестирования. В нем имеются таблицы: *вопросы*, *ответы*, *регистрационные данные*.

Администратор позволяет создать новый тест или добавить нового преподавателя. Первая функциональность реализуется добавлением в таблицу *Учителя* кода нового преподавателя, разрешения на пользование группы тестов и фиксацией разрешенных групп тестирования для

каждого набора тестов. Вторая – выделением соответствующего количества строк в таблице *Группы* заполнением их кодом учителя.

Создание нового теста происходит путем создания нового документа с выделением страницы *Вопрос*. Она заполняется, сохраняя определенный формат тестов. Затем запускается генерация служебных страниц *Ответы и Регистрационная.* Открывается доступ учетной записи преподавателя и ученика к этому документу. Новое тестирование регистрируется в документе *Регистрационный* в таблице *Тесты* в соответствующей категории тестов (в конец строки добавляется ссылка на созданный тестовый документ).

Преподаватель работает с *Приложением преподавателя*, пользуясь определенным аккаунтом преподавателя. Имеет право на редактирование *Регистрационного* документа и *Теста*.

Перед началом любой деятельности, преподаватель должен зарегистрироваться. Для этого он запускает преподавательскую подпрограмму, выбирает пункт *Регистрация* и вводит: *Код*, *Логин*, *Пароль*, *ФИО*, *e-mail*. Применение в *Регистрационном* документе находит строку с указанным кодом в таблице *Учителя*. Если регистрационные данные преподавателя здесь еще не введены, сохраняет новую информацию. В противном случае – выдает ошибку регистрации. При успешной регистрации допускаются дальнейшие действия с системой.

Перед каждым сеансом работы с преподавательским приложением преподаватель обязан войти в систему, запустив преподавательскую подпрограмму. При успешном входе преподаватель может осуществлять такие операции в системе: регистрацию новой группы, выбор конкретной группы тестирования, назначение группе тестирования, просмотр результатов тестирования (может добавить комментарий к отдельному ответу ученика), открытие результатов к просмотру ученикам.

Ученик работает только с ученической подпрограммой, с помощью которой он может выполнять такие действия, как регистрация в системе, прохождение тестирования, определение временных ограничений на тестирование и анализ результатов тестирования.

Архитектура приложения

Сначала выберем язык программирования. Поскольку *Google Docs API* реализован на многих языках, выбор довольно велик. Например, связка *PHP*/*JavaScript* дает возможность пользователям работать с системой без установки на своем компьютере дополнительного программного обеспечения, прямо в браузере. Но *PHP* и *JavaScript* не являются простыми языками для разработки удобного и интуитивного интерфейса. В данном случае больше подошли бы такие языки, как *.NET C#* или *Java*. Учитывая создание только прототипа системы, можно воспользоваться языком программирования, удобным для разработчиков, а окончательную версию системы можно адаптировать для работы и в браузере. Как основной язык разработки был выбран *C#*.

Остановимся на структуре системы тестирования. Понятно, что желательно было бы отдельное приложение для учащихся, которые должны проходить тестирование. Отдельная подпрограмма нужна и преподавателю, который будет создавать задачи. Поскольку часть функций в них должна быть общая (как минимум – они должны работать с одними и теми же данными в *Google Docs*), следует иметь и общую часть. Приходим к варианту с такими проектами: *GTester*.*Student*, *GTester*.*Teacher*, *GTester.Common.*

Далее опишем организацию иерархии классов для построения тестовых заданий (вопросов). Следует учесть такие факторы:

- каждый тип вопросов должен имеет специфику, требующую отдельной логики методов (оценки, создания, рандомизирования) и свойств (в разных тестах разные входные данные);
- приложение должно работать со всеми типами тестов одинаково, не учитывая особенностей типа вопросов;
- каждый тест должен быть создан по входным данным, которые разбираются одним алгоритмом;

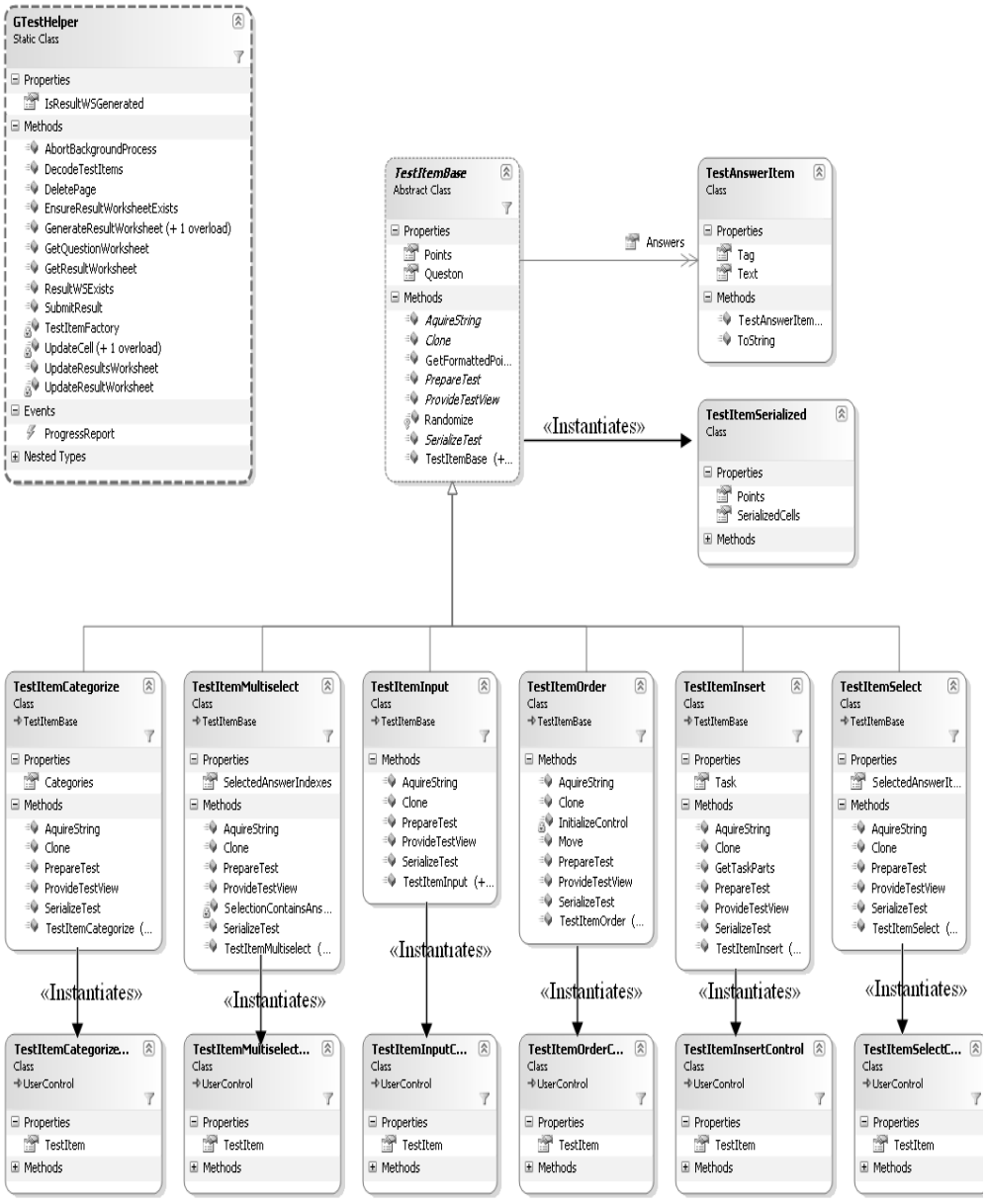
• сериализация результатов тестов должна быть также выполнена по одному принципу, независимо от типа тестового задания;

• должна быть обеспечена возможность создания копии задачи, не зная ее типа.

Для удовлетворения всех этих требований была разработана иерархия классов (рисунок).

Рассмотрим, как именно такое построение обеспечивает выполнение перечисленных требований одновременно.

Поскольку все типы вопросов происходят от одного базового абстрактного класса *TestItemBase*, со всеми вопросами можно работать, не задумываясь об особенностях типа (формы) задания. Для удобства и полноты реализации необходимых действий этот абстрактный класс содержит набор виртуальных и абстрактных методов, которые должны быть переопределены в специфических классах. Так решается проблема обработки специфических свойств и логики каждого отдельного типа вопросов. Это позволяет применять один интерфейс для обработки всех типов заданий теста.



Иерархия классов

А как же создать начальные тесты? Для этого обратимся к коду статического класса *GTestHelper*, осуществляющему общие операции с *Google Docs API* и сетью. В его функции входит и сериализация, и десериализация отдельных тестов из таблиц *Google Spreadsheets*.

Google Docs API предоставляет возможность читать содержание ячейки таблицы. Поэтому функция создания теста читает ячейки, анализирует их содержимое и выполняет необходимые действия:

```
string question = "";
double points = 0;
TestItemBase item=null;
foreach (CellEntry cEntry in cFeed.Entries)
{
   // вычисляем относительный адрес ячейки
  int relativeRow = (int)cEntry.Row % 10;
  int testId = (int)cEntry.Row / 10;

  if (testId == blockedTestId)
```

```
        continue;

  if (cEntry.Column == 1 && relativeRow == 1)
{   // это текст вопроса
    question = cEntry.Value;
    continue;
}

  if (cEntry.Column == 2 && relativeRow == 1)
{   // это количество набранных баллов
    if (!double.TryParse(cEntry.Value, out
        points))
       blockedTestId = testId;
    continue;
}

  if (cEntry.Column == 3 && relativeRow == 1)
{    // это тип теста
    item = TestItemFactory(cEntry.Value);
     //создаем соответствующий тип
      if (item != null)
      {
        item.Queston = question;
        item.Points = points;
        result.Add(item);
      }
      else blockedTestId = testId;

      continue;
      }
      if (item != null)
       item.AquireString(relativeRow,(int)
          Entry.Column, cEntry.Value);
}
```

Как свидетельствует этот фрагмент текста, создание экземпляров вопросов никак не привязано к обработке конкретного типа вопроса. Вся работа заключается в считывании данных из таблицы и их передачи экземпляру задания, если он уже создан. Если же нет – узнать, какой тип задания требуется создать и передать определенное значение типа функции создания.

Именно создание вопросов происходит в функции *TestItemFactory*, которая, как видно из названия, реализует паттерн *Factory Method*.

```
Private static TestItemBase TestItemFac
    tory(string typeSpecification)
        {
            if (string.IsNullOrEmpty(type
               Specification))
                return null;

            if(typeSpecification.ToLower().
             Equals(ORDERTypeDescriptor))
                return new TestItemOrder();

            if (typeSpecification.ToLower().
             Equals(INPUTTypeDescriptor))
                return new TestItemInput();

            if (typeSpecification.ToLower().
               Equals(INSERTTypeDescriptor))
                 return new TestItemInsert();

            if (typeSpecification.ToLower().
               Equals(SELECTTypeDescriptor))
                 return new TestItemSelect();

            if (typeSpecification.ToLower().
           Equals(MULTISELECTTypeDescriptor))
                 return new TestItemMulti
                   select();

            if (typeSpecification.ToLower().
           Equals(CATEGORIZETypeDescriptor))
                 return new TestItemCate
                   gorize();

          return null;
        }
```

Здесь, в зависимости от значения строчной константы, и решается, какой именно тип вопроса создать. Стоит обратить внимание и на вызов метода *AquireString*, с помощью которого, фактически, и строится каждый вопрос. Этот метод – реализация паттерна *Builder*. Ему передаются все ячейки, расположенные в пределах отдельного вопроса. Реализация этого метода специфическая для каждого определенного типа вопроса. Только он знает, какие ячейки что должны содержать.

Аналогично работает и операция сериализации – сохранение данных в ячейки таблицы *Google Docs Spreadsheets*.

```
  For (int i = 0; i < testResults.Count;
       ++i)
     {
    TestItemSerialized serialized = test
       Results[i].SerializeTest(Etalon[i]);

     if (serialized == null)
         continue;

     resCustom = new ListEntry.Custom();
     resCustom.LocalName = answerListEntry.
               Elements[index++].LocalName;
     resCustom.Value = serialized.Points.
               ToString(usCultureInfo);
     result += serialized.Points;
     resEntry.Elements.Add(resCustom);

     for (int answerIndex = 0; answerIndex <
serialized.SerializedCells.Count;++answerIndex)
     {
      resCustom = new ListEntry.Custom();
      resCustom.LocalName = answerListEntry.
                Elements[index++].LocalName;
```

```
        resCustom.Value = serialized.Serialized
                  Cells[answerIndex];
        resEntry.Elements.Add(resCustom);
      }
  }
```

Главным здесь будет вызов *SerializeTest*, с помощью которого каждый тест создает объект, в котором описывается, как именно следует сохранять текст в таблице.

После того, как разобран процесс создания и сериализации тестовых вопросов, интересно рассмотреть их отображения на форму вопроса, поскольку у каждого теста должно быть свое собственное представление в *UI*.

Для отображения любого вопроса в основной форме ученической программы используется следующая функция:

```
Private void ShowTestItems(List<Test
                        ItemBase> tests)
  {
    foreach (TestItemBase testItem in tests)
    {
         TabPage page = new TabPage();
         Control ctrl = testItem.Provide
                          TestView();
         ctrl.Dock = DockStyle.Fill;

         page.Controls.Add(ctrl);
         this.tcTesting.TabPages.Add(page);
            }
          this.tcTesting.SelectedIndex = 0;
   }
```

Главная форма также не зависит от конкретных типов вопросов теста. Это возможно благодаря реализации *Abstract Factory* в методе *ProvideTestView*.

Этот метод имеет конкретную реализацию для каждого типа заданий теста. Каждый конкретный класс знает о виде задачи в основной форме. Приведем пример реализации этого метода для одного из типов тестовых вопросов:

```
Public override System.Windows.Forms.Con
       trol ProvideTestView()
        {

            if (this.control == null)
                this.control = new Test
             ItemMultiselectControl(this);

            return control;
        }
```

Метод возвращает системный класс *Control*, в котором хранится специфичность класса, ответственного за вид этого типа задания.

Последний пункт, который следует рассмотреть в рамках архитектуры классов заданий – это создание копии вопроса, не зная, какой именно тип тестового задания рассматривается. Для этого применен абстрактный метод *Clone* в базовом классе *вопрос*

```
Public abstract TestItemBase Clone();
```

Последнее можно организовать с помощью копировального конструктора. Если в каждом классе *вопрос* будет реализован конструктор копирования, метод *Clone* будет очень простым:

```
public override TestItemBase Clone()
{
    return new TestItemMultiselect(this);
}
```

Таким образом, при необходимости можно получить копию задания, вызвав метод *Clone* от существующего задания, не заботясь об определении типа (формы) задания.

Структура тестов в тестовом документе

Тестовые задания хранятся в таблице с названием *Вопросы*. Для каждой задачи выделяется 10 строк таблицы, т.е. первое задание начинается с первой строки, второе – с одиннадцатой, третье – с 21-й строки. В дальнейшем вся нумерация строк и столбцов будет считаться в пределах одной задачи.

Первая строка каждого задания фиксированная. Она состоит из таких пунктов:

- вопрос (первая строка, первый столбец (*A*1));
- количество баллов за успешное выполнение задания (*B*1);
- тип теста (*C*1).

Все последующие столбцы первой строки игнорируются и могут содержать любую необходимую разработчику информацию – комментарии, дополнительные вычисления или не содержать ничего.

Следующие девять строк специфические для каждого типа задания, а потому будут описаны в соответствующем разделе.

Количество баллов как за вопрос, так и за любую его часть (если это возможно) может быть выражено действительным числом.

Вопросы могут быть следующих типов:

Таблица 1

Тип задания и описание	Код типа
Вписать ответ – пользователь должен вписать ответ на вопрос теста.	*INPUT*
Заполнить пробелы – пользователь заполняет пробелы в тексте вопроса по тем же принципам что и предыдущий вопрос.	*INSERT*
Выбрать один правильный ответ – пользователь должен выбрать только один правильный ответ из предоставленных вариантов.	*SELECT*
Выбрать правильные ответы – пользователь должен выбрать правильные ответы из множества предоставленных; правильный набор может содержать как одну, так и более ответов.	*MULTI SELECT*
Распределить по категориям – пользователь должен распределить понятия по имеющимся категориям.	*CATEGORIZE*
Упорядочить понятие – пользователь должен упорядочить предоставленные понятия по определенному правилу, описанному в задании.	*ORDER*

Сравнение с другими системами тестирования

Если сравнивать приложение *GTester* с аналогичными разработками известных компаний, то оно имеет ряд существенных недостатков. Сравним разработанное авторами приложение с двумя известными системами тестирования: *Zoho Challenge* [5] и *Google Forms*.

Очевиден основной недостаток *GTester*: это применение написано на *.NET Framework*, поэтому для работы требуется *.NET Framework* и запуск с локального компьютера. Причина этого недостатка то, что представлен только прототип системы. При дальнейшем развитии системы этого недостатка можно избежать с минимальными изменениями в архитектуре приложения, перейдя, например на *ASP.NET*.

С другой стороны, *GTester* имеет несколько преимуществ перед конкурентами. Во-первых, в нем реализовано значительно больше типов тестовых заданий, и добавить новый тип не составит значительной проблемы благодаря специфике выбранной архитектуры. Во-вторых, если сравнивать с *Zoho Challenge*, то возможности, предоставляемые *GTester*, даже лучше, чем дает бесплатная версия *Zoho Challenge*. Если сравнивать с *Google Forms*, то *GTester* предоставляет более удобные возможности работы преподавателя с тестами и их результатами. Но описанный прототип *GTester* пока не имеет никаких специализированных приложений для упрощения создания новых тестов, тогда как *Zoho Challenge* и *Google Forms* предоставляют специализированный сервис по созданию новых вопросов и тестов.

Основное преимущество *GTester* – ориентированность на украинских пользователей. Учителя школ Украины уже теперь используют наш прототип, размещенный на *http*://*zavadsky.at*.*ua*/*index*/*onlajnova_navchalna_sistema*/0–4. Его интерфейс полностью украинизирован, тогда как *Google Forms* украинизирован не полностью, а в *Zoho Challenge* отсутствует украинская локализация.

Следует отметить развернутую и специализированную систему управления тестированием *GTester*: система разработана специально для управления и администрирования большого количества тестов и при взаимодействии многих преподавателей, тогда как *Google Forms* и *Zoho Challenge* скорее ориентированы на одного преподавателя, который сам администрирует, создает тесты, оценивает учеников.

Результаты сравнения приведены в табл. 2.

Таблица 2

Система тестирования	Требует установки	Создание тестов	Локализация	Типы тестов	Система управления
Zoho Challenge	–	Реализовано	Отсутствует	5	+
Google Forms	–	Реализовано	Частичная	7	–
GTester	+	Не реализовано	Имеется	8	+

Заключение. В работе описана реализация системы тестирования в виде отдельного приложения, построенного на базе *Google Docs API*. Этим подтверждена эффективность использования *Google Docs* как серверной платформы для распределенных приложений.

Очевидно, что основным усовершенствованием *GTester* будет переход к полностью браузерному приложению. Если *GTester* не будет зависеть от внешних приложений, возможная сфера его использования существенно расширится. Этого можно достичь несколькими способами.

Простейший из них – превращение применения к виду *ASP.NET*-приложения. По сути, внутри основной логики применения ничего не изменится. Все будет работать, как и прежде. Но придется полностью переделать *UI*-приложения, что может в специфических местах оказаться довольно сложным. Другим вариантом решения этой проблемы может быть переход на *Java*, что повлечет частичное переписывание приложения. Оно ограничится заменой служебных слов и решения конкретных проблем перехода на другой язык. В этом случае *UI* переписывать все равно придется, но тогда применение станет независимым от операционной системы и архитектуры компьютера, на котором оно будет выполняться, что тоже может быть важно.

Еще одним вариантом перевода применения в браузерный режим есть переход на *PHP* + *JavaScript*, что влечет за собой полное переписывание применения в другие принципы программирования и существенно усложняет программирование отдельных частей приложения. В результате можно достичь уровня автономности, присущей *Google Docs*: приложение может работать почти без проблем в любом браузере и не нуждаться в дополнительных инструментариях на клиентском компьютере.

Другое слабое место в системе – ее администрирование. Система управления *GTester* достаточно гибкая и многофункциональная, но она не имеет собственного специализированного *UI*, а потому во многом будет слишком сложной.

Существенным пунктом дальнейших усовершенствований будет функционал преподавательского приложения. Бесспорно, в процессе пользования системой возникнет много пожеланий со стороны пользователей.

Поскольку *GTester* – это приложение для тестирования, ключевую роль в нем должны выполнять сами тесты. Благодаря прозрачной и четкой архитектуре системы тестирование (добавить новый тип или изменить существующие) проходит достаточно просто. Конечно, возможно еще больше упростить этот процесс, позволив подключать к применению плагины и конфигурировать функционал из внешних файлов при необходимости.

1. *Глибовец Н.Н.*, *Заболотный Р.И.*, *Завадский И.А.* Анализ систем онлайн документооборота // Науч. тр. ЧГУ им. Петра Могилы: Николаев, серия: Компьютерные технологии, 2011, Изд. 130, Т. 143 – С. 121–130.
2. *Google* Code: Protocol Reference. – http://code.google.com/apis/gdata/docs/2.0/reference.html (проверено 31.05.2010).
3. *Google* Code: Google Data Protocol. – http://code.google.com/apis/gdata/ (проверено 31.05.2010).
4. *Google* Code: Developer's Guide. – http://code.google.com/apis/documents/docs/developers_guide.html (проверено 31.05.2010).
5. *Zoho.* – http://www.zoho.com/ (проверено 31.05.2010).

E-mail: *glib@ukma.kiev.ua*

●