5. Брайан Ларсон. Разработка бизнес-аналитики в Microsoft SQL Server 2005. – М. Питер, 2008 – 674 с.
6. Брайан Ларсон. Microsoft SQL Server 2005 Reporting Services. Традиционные и интерактивные отчеты. Создание, редактирование, управление. – М. NT Press, 2008 – 600 с.

УДК 004

Разработка мобильных приложений для задач информационной поддержки в off-line режиме

Бабаков И.В., Пономарев А.А.
Томский политехнический университет, 634050, Россия, Томск, ул. Ленина 30
E-mail: InstanT.977@gmail.com

This article is concerned with the problem of information support of the medical facility staff (in particular, of the military hospital staff). This problem is concerned with inability of data registration without access to the network. The author pay attention to the topicality of this problem, covers possible ways to implement such a system and presents effective one.

Key words
Medical information system, a treatment facility, a mobile application offline, data synchronization, Web SQL, HTML5.
Ключевые слова
Медицинская информационная система, лечебно-профилактическое учреждение, мобильное приложение, офлайн, синхронизация данных, Web SQL, HTML5.

Введение

В настоящее время в России, как и во всем мире, идет бурное внедрение информационных технологий во все сферы человеческой деятельности, в том числе и в медицинской отрасли. Хорошо известно, что в силу своих профессиональных обязанностей медицинским работникам требуется посещать пациентов и вести свою профессиональную деятельность за пределами ЛПУ, но, как правило, МИС не способны полноценно функционировать при отсутствии подключения к сети (*offline*).

Ни один из имеющихся на рынке продуктов не позволяет выполнять работу с МИС на планшетных устройствах в режиме *offline*. Одной из современных технологий, позволяющих реализовать такую задачу является HTML5.

От теории к практике - Web Sql

Сструктурная схема рассматриваемого приложения приведена ниже. В HTML5 есть много новых возможностей, которые позволяют web разработчикам создавать мощные и насыщенные приложения. Для решения нашей задачи наибольший интерес представляют способы хранения данных на клиенте в *offline* режиме. К этим таким возможностям относятся Web storage и Web SQL database [1].

Первым шагом работы с Web SQL является инициализация пространства имен в html5.

```
var html5rocks = {};
html5rocks.webdb = {};
```

База данных *html5* поддерживает как асинхронный, так и транзакционный режимы работы [4]. В транзакционном режиме также имеется два типа транзакций:

- Чтение\Запись (*transaction*()), при обращении к базе данных, не происходит

блокирования записей;

- Только чтение (*readTransaction*()), при обращении к базе данных, происходит блокирование записей для запрета внесения изменений в данные.

Чтобы получить доступ к базе данных необходимо открыть соединение, то есть указать имя сервера, версию, а также текстовое описание и предполагаемый размер базы данных в битах.

```
html5rocks.webdb.db = null; // Инициализируем базу
html5rocks.webdb.open = function() { // Открываем соединение
var dbSize = 5 * 1024 * 1024; // 5MB
html5rocks.webdb.db = openDatabase("Todo", "1", "Todo manager", dbSize);}
html5rocks.webdb.onError = function(tx, e) {
alert("Произошла ошибка: " + e.message);} // обработчик ошибки onError
html5rocks.webdb.onSuccess = function(tx, r) {
html5rocks.webdb.getAllTodoItems(loadTodoItems);} // обработчик ошибки onSuccess
```

Теперь перейдём к созданию таблицы. Создать таблицу можно с помощью выполнения SQL команды *create table* [2]. Определим функцию, которая создаст таблицу с именем *todo* в случае её отсутствия в базе данных. Пусть таблица будет иметь три столбца:

- *ID* – инкрементное, уникальное поле;
- *todo* – текстовое поле;
- *added_on* – поле, содержащее время добавления записи в базу данных.

```
html5rocks.webdb.createTable = function() {
var db = html5rocks.webdb.db; //Инициализируем объект базы данных
db.transaction(function(tx) {/// транзакционный запрос на создание таблицы
tx.executeSql("CREATE TABLE IF NOT EXISTS " +
"todo(ID INTEGER PRIMARY KEY ASC, todo TEXT, added_on DATETIME)", []);});}
```

Теперь, когда у нас имеется созданная таблица, наполним её данными, используя SQL команду *insert*.

В Web SQL существует команда *executeSql*, которая позволяет передавать в запросы в качестве параметров значения переменных, аналогично *String.Format* в C#, а также функции, которые выполняться в случае успешного запроса (*OnSuccess*), либо в случае ошибки (*onError*).

```
html5rocks.webdb.addTodo = function(todoText) {
var db = html5rocks.webdb.db;
 db.transaction(function(tx){
var addedOn = new Date();//Инициализируем новый объект даты
tx.executeSql("INSERT INTO todo(todo, added_on) VALUES (?,?)",//Выполняем запрос
[todoText, addedOn],
html5rocks.webdb.onSuccess,
html5rocks.webdb.onError);});}
```

Имея данные в таблице, попробуем извлечь их, используя SQL команду SELECT [5]. Объявим функцию, получающую все данные из таблицы *todo*.

```
html5rocks.webdb.getAllTodoItems = function(renderFunc) {
  var db = html5rocks.webdb.db;
  db.transaction(function(tx) {
    tx.executeSql("SELECT * FROM todo", [], renderFunc,
      html5rocks.webdb.onError);});}
```

Обратим внимание на то, что все обращения к базе данных мы производим асинхронно, то есть к примеру, функция *executeSql* – не возвращает никаких значений, она лишь выполняет одну из функций передаваемых ей в качестве параметров в случае ошибки, либо успеха.

В данном случае, если запрос выполнится успешно, выполним функцию для отображения полученных данных *loadTodoItems*.

```
function loadTodoItems(tx, rs) {
var rowOutput = "";
var todoItems = document.getElementById("todoItems");
for (var i=0; i < rs.rows.length; i++) {
    rowOutput += renderTodo(rs.rows.item(i));}
     todoItems.innerHTML = rowOutput;}
    function renderTodo(row) {
     return "<li>" + row.todo +
          " [<a href='javascript:void(0);' onclick=\'html5rocks.webdb.deleteTodo(" +
          row.ID +");\'>Delete</a>]</li>";}
```

В функции renderTodo, которая будет возвращать нам html код, который мы будем отображать в DOM элементе *todoItems* в функции *loadTodoItems*. Сразу при создании html кода, создадим кнопку, которая будет удалять данную запись из базы данных, то есть обычную кнопку, по нажатию на которую будет вызываться соответствующая функция «deleteToDo».

Инициализируем функцию для удаления записей из таблицы используя SQL команду «DELETE».

```
html5rocks.webdb.deleteTodo = function(id) {
    var db = html5rocks.webdb.db;
    db.transaction(function(tx){
       tx.executeSql("DELETE FROM todo WHERE ID=?", [id],
          html5rocks.webdb.onSuccess,
          html5rocks.webdb.onError);
       });}
```

Теперь воспользуемся созданными функциями. Когда страница загрузилась, откроем соединение к базе данных и создадим таблицу, после чего отобразить все элементы данной таблицы. Функция *init* будет вызываться после загрузки страницы, поэтому она помещена в тэг «*body*» на событие *onload*.

```
    function init() {
     html5rocks.webdb.open();
     html5rocks.webdb.createTable();
     html5rocks.webdb.getAllTodoItems(loadTodoItems);}
    </script>
    <body onload="init();">
```

Также для добавления в базу данных записей из пользовательского интерфейса можно воспользоваться функцией «*html5rocks.webdb.addTodo*»

```
    function addTodo() {
     var todo = document.getElementById("todo");
     html5rocks.webdb.addTodo(todo.value);
     todo.value = "";}
```

Выводы

Таким образом, HTML5 обладает такими технологиями как: *Application Cache, WebStorage* и *WebSql*, которые в совокупности позволяют в полной мере решить поставленную задачу. На примере, стало ясно, что база данных, поддерживаемая HTML5 – «Web SQL» является полноценной базой данных, с возможность транзакций, а также асинхронным режимом работы.

Список литературы:

1. Хабрахабр // HTML5. Работа с WebSql базой. 2013. URL: http://habrahabr.ru/post/84654/ (дата обращения: 23.12.2013).
2. Хабрахабр // «Переезжаем в офлайн: Web Storage, Application Cache и WebSql. 2013. URL: http://habrahabr.ru/post/117123/ (дата обращения: 21.12.2013).

УДК 004

Microsoft открывает исходный код. Проект ROSLYN

Балашова О.В., Черкашин А.Ю.
Томский политехнический университет, 634050, Россия, г. Томск, пр. Ленина 30
E-mail: an0648766@mail.ru

The article is devoted to Microsoft open-source projects, in particular Roslyn. Roslyn project is an open-source .NET Compiler Platform which exposes a set of Compiler APIs and Workspaces APIs. It is described project itself, its advantages and the main compiler platform concept. The description is supported by short example of syntax tree implementation.

Key words: *open-source, Roslyn, compiler, .NET, syntax tree.*
Ключевые слова: *открытый исходный код, компилятор, синтаксическое дерево.*

Современный IT-мир очень быстро меняется, и в этом быстро меняющемся мире программные продукты с открытым исходным кодом обладают рядом однозначных преимуществ.

Коммерческое ПО закрыто, его разработчики хранят тайны своих решений и не раскрывают ни внутренней архитектуры, ни форматов представления данных, ни интерфейсов. Если клиент понимает, что возможности существующего решения его не устраивают, ему, как правило, предлагается купить новую версию. В результате возникает непонимание и конфликты с клиентами. А это ведет к тому, что клиент стремится уйти к другому поставщику.

Продукты с открытым исходным кодом, напротив, практически всегда построены по модульному принципу, то есть любое решение собирается из отдельных составляющих. При этом документированы и открыты и сами компоненты решения, и интерфейсы между ними.

Решения на основе продуктов с открытым исходным кодом оказываются более гибкими и лучше приспособлены к современному быстро меняющемуся миру. [1]

Долгое время исходные коды программных продуктов компании Microsoft были в закрытом доступе. В марте 2014 года Microsoft впервые запустила ряд проектов с открытыми исходными кодами, среди которых наибольшую значимость представляет проект Roslyn.