Kiss

User Manual
Version 0.1.0
May 26, 2018

by Blake McBride

This manual was typeset with the TEX typesetting system developed by Donald Knuth utilizing the Texinfo format.

Short Contents

Table of Contents

1 Introduction

The *KISS Framework* is an application development framework for developing web-based business applications. The main home for *Kiss* is https://gitlab.com/kissweb/Kiss

Kiss' focus is on simplicity and development speed. By being simple to develop in, development and support of the application can occur more rapidly. Simplicity is achieved by abstracting away as much common functionality as possible so that developer lines of code are maximally for the application solution rather than infrastructure and support of the framework. Throughout the framework, business-normal defaults have been employed in order to minimize commonly needed functionality.

Another goal of the *Kiss* framework is to be a complete web-based application development solution. *Kiss* isn't a browser solution alone, nor is it a back-end solution. *Kiss* includes solutions for both ends – although the two sides may largely be used independently.

Kiss attempts to create a consistent interface. This can greatly simply code even in simple cases. For example, in terms of an input text control, why would you disable/enable it with:

```
$('#id').prop('disable', false);
$('#id').prop('disable', true);
```

and then hide/show it with:

```
$('#id').hide();
$('#id').show();
```

Kiss provides a consistent interface. With *Kiss*, you would do:

```
$$('id').disable();
$$('id').enable();
$$('id').hide();
$$('id').show();
```

Kiss is designed to be simple to get started with, simple to learn, and simple to use. *Kiss* does this while supporting important technologies such as micro-services, front-end components, and SQL.

The term *single page application* has several, subtly different, meanings. One meaning is that the entire application code is bundled into a single file or HTTP GET request. In that sense, *Kiss* is not a single page application. This makes no sense for a business application that could have hundreds of screens.

Another meaning of the term *single page application* is that there is only a single `html` tag and all of the remaining pages are modifications of the original `html` tag contents. In this sense, *Kiss* is a *single page application*. *Kiss* applications lazy-load as needed. Browser cache is leveraged to minimize Internet traffic.

Kiss is used in a production environment and built by someone with more than 30 years experience as a framework designer and a business application software engineer. So *Kiss* is not a proof of concept.

Kiss was built as a solution to the challenges faced by the author when developing web-based business applications. As such, *Kiss* is more a solution for business application development than for the development of public facing company presentation web sites.

Another goal of *Kiss* is to keep the front-end and back-end as independent of each other as possible. To this end, communications between the front-end and back-end occur via REST services and JSON. This accomplished two things. First, it allows your organization to be best prepared for the ever-evolving software environment. Pieces can be changed and enhanced without causing massive re-writes of the entire system. The second advantage is that by pushing as much processing to the front-end as possible (rendering the display on the front-end), the system can better scale.

1.1 *Kiss* Highlights

Some highlights of the *Kiss* system include:

1.1.1 Back-end Highlights

1. Micro services - add, change, or delete a web service on a running system.
2. Each web-method is in a single file and are very simple. No configuration files or setup code needed.
3. Easy access to common SQL databases with support for nested queries without cursor interference.
4. All REST services are stateless. However, the system fully authenticates each request.
5. Changes to web services occur immediately, on a running system, without the need to reboot the application.
6. A growing class library to handle common business application needs.
7. Back-end framework is written in Java, and the system is portable to Linux or Windows servers.
8. Web services may be written in Groovy, Java or Common Lisp. Python, JavaScript, Ruby, and Scala are expected to follow soon.
9. User authentication

1.1.2 Front-end Highlights

1. Build your own HTML components thus encapsulating any amount of code into a simple, custom HTML tag.
2. Browser cache control. Never ask your users to clear their browser cache again.
3. All code written in JavaScript/HTML/CSS. No need for a complex build and debug process, nor any need to learn yet another language.
4. Growing list of included business oriented components designed to provide simple access to fully functional business components.
5. Straight forward means of designing your own components without a lot of hidden and unpredictable magic.
6. System is small and concise, rather then hundreds of megabytes other systems take up.
7. Consistent and simple API.

1.1.3 Back-end Web Service Example

The following example depicts a complete back-end web service. The path to the file is its URL. The class name is the web method name.

The file is a text file, but compiled code gets executed. Authentication occurs before `main` is called.

Simply drop the file in place and the web service and method become immediately available on a running system. Changes to the service take effect immediately (no need to reboot the server app). There are no configuration files or other code that needs to be changed.

For example, the following file is located in the `my/web/service` directory.

```
class MyWebMethod {
    static void main(JSONObject injson, JSONObject outjson) {
        int num1 = injson.getInt("num1");
        int num2 = injson.getInt("num2");
        outjson.put("result", num1 + num2);
    }
}
```

1.1.4 Front-end Web Service Usage Example

The following front-end example utilizes the web service defined in the previous sub-section.

```
var data = {
    num1:  22,
    num2:  11
};
Server.call("my/web/service", "MyWebMethod", data).then(function (res) {
    if (res._Success) {
        var result = res.result;
        //...
    }
});
```

1.2 HTML component usage

To use a component add to HTML:

```
<my-component></my-component>
```

Add to JavaScript:

```
utils.useComponent('MyComponent');
```

The component can put any HTML in the component location, have any functionality, have its own modal windows, and use other components. The component can have custom and non-custom attributes (like style). Non-custom attributes do what you's expect them to do.

The system also supports tag-less components. This provides an easy way to package arbitrary blocks of code (that can have screens too).

1.3 System Maturity And Future

Largely, the Kiss system, in the form of running applications, has been used in production environments for a few years. The effort to tease away the generic parts and generalize them is recent (2018). Moving forward, as time permits, the priorities are as follows:

1. Improve documentation
2. More examples
3. Greater front-end functionality (modal windows, grid control, etc. - note that these can be done by any other known facility. They just don't come included with Kiss yet.)
4. Support more back-end languages (such as Python, JavaScript, Ruby, etc.)

1.4 Contact And Links

The *Kiss* main web site is at http://kissweb.org

Source code is at https://gitlab.com/kissweb/Kiss

Public discussion and support is available at
https://groups.google.com/forum/#!forum/kissweb

Issue tracking is at https://gitlab.com/kissweb/Kiss/issues

Commercial support is available. Contact us via email at kissweb.org@gmail.com

1.5 License

LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1.6 Acknowledgments

The Kiss design, code, documentation, and web site were written by Blake McBride. The author gratefully acknowledges and appreciates, among others, the following:

Apache Groovy located at http://groovy-lang.org

ABCL project located at https://common-lisp.net/project/armedbear

Dynamic Loader located at https://github.com/dvare/dynamic-loader

JSON-Java located at https://github.com/stleary/JSON-java (although I am using a modified version located at https://gitlab.com/blakemcbride/JSON-java)

C3P0 located at https://www.mchange.com/projects/c3p0

Melaine Sarbey (melswildart@gmail.com) for creating the Kiss logo.

2 System Setup

2.1 Pre-requisites

You should download and install the following pre-requisites.

1. Java JDK 8 from https://www.oracle.com/java/index.html
2. Gradle from https://gradle.org
3. An SQL database server (e.g. PostgreSQL, Microsoft SQL Server, MySQL, Oracle, SQLite)
4. IDE (e.g. IntelliJ, Netbeans, eclipse)
5. GIT source code control system

The system was built and tested with JDK 8, PostgreSQL, IntelliJ, and tomcat. Other environments such as IIS, Glassfish, eclipse, should work fine but may require some configuration.

Install the above per their instructions.

In addition to the Gradle build files, Kiss comes with IntelliJ project files that may be used out-of-the-box. Other IDE's can import the Gradle build files.

2.2 Download Kiss

Kiss is located at https://gitlab.com/kissweb/Kiss

It can be downloaded via the following command:

```
git clone https://gitlab.com/kissweb/Kiss.git
```

2.3 Setup And Configuration

Given that Kiss is for business applications, it authenticates its users. In order for this to work, there must be a database of valid users. This information is persisted in an SQL database. Therefore a database will need to be created. As shipped, the system comes configured as follows:

Database type	PostgreSQL
Host	localhost
Database	kiss
Database user	postgres
Database user password	postgres

This can be configured in the file `src/main/application/KissInit.groovy` Valid options for the Database type are as follows:

- PostgreSQL
- Microsoft SQL Server
- MySQL

- Oracle
- SQLite

Support for other databases is easy to add.

The remaining parameters should be self-explanatory. Use the format shown in the example.

Create the database (named "kiss" above or whatever name you configured). After that, you can run the script file `init.sql` to create the table and some initial data.

The default username is *kiss*, and the default password is *password*

3 Orientation

The entire source code comes with the system and is convenient when debugging, however, only a few areas in the system would normally be of concern when building an application as follows.

3.1 Back-end Application Files

`src/main/application/KissInit.groovy`
: This file is used to configure the system.

`src/main/application`
: All other files under this directory represent the application back-end. All the files are used and distributed in source form. The Kiss system compiles them at runtime but does not save the compiled form. Updates to files under this directory take affect immediately on a running system.

3.2 Front-end Application Files

Files under the `src/main/webapp` directory represent the front-end of the application.

All files under the `src/main/webapp/kiss` directory are part of the *Kiss* system and would normally not need to be touched.

`index.html` and `index.js` are also part of the *Kiss* system and aren't normally modified.

`login.html` and `login.js` represent the user login page and would be modified to suit your needs.

Other directories such as `page1` represent other user pages and would be the application specific screens you create. The included `page1` directory is only an example page.

3.3 Single Page Application

Kiss applications are single page applications in the sense that there is a single `<body>` tag and all other pages essentially get placed into that tag on a single page. However, *Kiss* is not a single page application in the sense that the entire application gets loaded with a single `GET` request. This doesn't make sense for a large business application in which many hundreds of pages may exist. *Kiss* lazy-loads pages as they are used, and except for browser cache, eliminates them once another page is loaded.

3.4 Controlling Browser Cache

The user's browser cache can be controlled from the file `src/main/webapp/index.html` In that file you will see two lines that look as follows:

```
var softwareVersion = "1";  // version of the entire system
var controlCache = false;   // normally true but use false during
                            // debugging
```

If `controlCache` is set to `true`, each time `softwareVersion` is incremented all users starting the application will be forced to load new code from the server and not use their

browser's cache. Once they download the new version, normal browser cache activity will occur.

3.5 Creating JavaDocs

JavaDocs for the *Kiss* system may be created by issuing the following command: `gradle javadoc` The JavaDoc files end up in the `build/docs/javadoc` directory.

3.6 Deploying A Kiss Application

The only file needed to deply the application is `Kiss.war` It can either be built by your IDE, or at can be build by typing `gradle war` at a command prompt. In the case of Gradle, `Kiss.war` ends up in the `build/libs` directory.

If using *tomcat*, `Kiss.war` should be placed in the `webapps` directory. When *tomcat* starts, it will see the file, unpack it, and run it. The application will be available at `[HOST]/Kiss`

Renaming `Kiss.war` to `ABC.war`, for example, will cause the application path to change to `[HOST]/ABC`

3.7 Database Access

Kiss comes with a powerful library for accessing SQL databases. Code for this is located under `org.kissweb.database` It is currently only documented via JavaDocs but examples and documentation will follow soon. It is currently being used in production environments. This API provides the following benefits:

- Automatic connection and statement pooling
- Vastly simpler API than bare JDBC
- Handling of parameterized arguments
- Auto generation of SQL for single record adds, edits, and deletions
- Auto handling for cases of cursor interference on nested queries
- Supports transactions out-of-the-box

3.8 Learning The System

In order to start getting a feel for how `Kiss` applications function, in terms of the back-end, look at files in the `src/main/application/page1` directory. With `Kiss` you can develop applications in several different languages. The `page1` example shows the same code in all of the suported languages.

In terms of the front-end, see the example files under `src/main/webapp/page1`

Method, Macro, Function And Variable Index

(Index is nonexistent)