AGH University of Science and Technology

Faculty of Physics and Applied Computer Science

Engineering thesis

Michal Stefaniuk

major: **applied computer science**

Web application developement using WebToolkit C++ on the example of a banking service

Supervisor: **Dr Grzegorz Gach**

Cracow, June 2021

Oświadczenie studenta

Uprzedzony(-a) o odpowiedzialnosci karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz. U. z 2018 r. poz. 1191 z pozn. zm.): ,,Kto przywlaszcza sobie autorstwo albo wprowadza w blad co do autorstwa calosci lub czesci cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolnosci albo pozbawienia wolnosci do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu tworcy cudzy utwor w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwor, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a takze uprzedzony(-a) o odpowiedzialnosci dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018r. Prawo o szkolnictwie wyzszym i nauce (Dz. U. z 2018 r. poz. 1668 z pozn. zm.) ,,Student podlega odpowiedzialnosci dyscyplinarnej za naruszenie przepisow obowiazujacych w uczelni oraz za czyn uchybiajacy godnosci studenta.", oswiadczam, ze niniejsza prace dyplomowa wykonalem(-am) osobiscie i samodzielnie i nie korzystalem(-am) ze zrodel innych niz wymienione w pracy.

Jednoczesnie Uczelnia informuje, ze zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysluguje pierwszenstwo w opublikowaniu pracy dyplomowej studenta. Jezeli Uczelnia nie opublikowala pracy dyplomowej w terminie 6 miesiecy od dnia jej obrony, autor moze ja opublikowac, chyba ze praca jest czescia utworu zbiorowego. Ponadto Uczelnia jako podmiot, o ktorym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyzszym i nauce (Dz. U. z 2018 r. poz. 1668 z po 'zn. zm.), moze korzystac bez wynagrodzenia i bez koniecznosci uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiazkow zwiazanych z odbywaniem studiow, udostepniac utwor ministrowi wlasciwemu do spraw szkolnictwa wyzszego i nauki oraz korzystac z utworow znajdujacych sie w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister wlasciwy do spraw szkolnictwa wyzszego i nauki moze korzystac z prac dyplomowych znajdujacych sie w prowadzonych przez niego bazach danych w zakresie niezbednym do zapewnienia prawidlowego utrzymania i rozwoju tych baz oraz wspolpracujacych z nimi systemow informatycznych.

Contents

Introduction

Software engineering is a very wide area of engineering which particularly concerns developing and maintaining programming products. A development process itself is a major challenge to all people involved, starting with engineers creating the code, continuing with product owners who are managing teams and coordinating the work flow and ending-up with managers who are setting the direction of the whole process.

The main motivation to create this thesis was simply to present a development process that includes basic and nowadays necessary tools which are significantly helpful in such process. Simultaneously creating a C++ web application was an equally important factor. To carry out the development and present all tools it was decided that an example C++ web application will be created with a GUI library in modern C++ called **Web Toolkit**.

Nowadays in a software engineering world there is a trend to migrate desktop applications to the internet, which has also impacted a lot GUI desktop libraries in decreasing their utilities. The secondary reason behind choosing C++ to create a web application, which is quite unusual, was to show the tremendous possibilities that this language still provides and to exhibit the capabilities of open source libraries shared among developers and engineers. The bottom line is - next chapters are a review of possibilities that C++-based web library provides.

1 Projects assumptions

1.1 Application's blue-print

The application is a single-page service run on a local server with an **http://localhost:8080/** address. After starting, logging panel appears, giving the user an opportunity to type their credentials and log-in.

The idea of the service is to provide basic functionalities of a banking service like for instance transferring money or checking their balance to the user. These required implementation of secure back-end layer with a database containing information about the users and transactions and a front-end layer which is transparent to the user.

Regarding the credentials a user will see different features after logging into the service. There might be many accounts created and held up in the database of the service but there are only two rights of access.

USER Transferring money and seeing current account's balance.

ADMIN Seeing details about every account in the service as well as having access to the logs from the application.

After user decides to quit the service, a possibility of logging out is provided and whilst doing that the application returns to the logging page where session is refreshed and database is updated. If the user was an administrator of the service, an access to the console logs that are gathering and sniffing network traffic like HTTP methods for RESTful APIs is also granted.

1.2 Programming environment

Developing any kind of programs usually requires specific environmental variables therefore a programmer's task is to choose a specific set of tools which will be the most handy during the process.

Regarding the specifics of the project, the chosen set of tools looks like following:

IDE Microsoft Visual Studio Community 2019,

LIBRARY Web Toolkit 4.3.1,

LIBRARY Boost C++ lib [latest version],

TOOL CMake VERSION 2.4,

TOOL Git BASH.

The application was developed under Windows 10 64-bit operating system, but it could have been developed under any other platform that is supported by WebToolkit library.

IDE - Integrated Development Environment

IDE is a program or a set of programs merged into one that usually gathers tools, libraries, debuggers, run time scripts and any other stuff necessary for the developer to write the code. The purpose of IDE is to allow easily and swiftly create code, but at the same time test it, compile it and run it in one place.

The advantage of this solution is that it gives the developer an opportunity to set up and scale his development environment adjusting it to the project needs. Usually IDE, as well as modern text editors, also support plugins which are nice addition to the program, like for instance syntax highlighters, semantics hints, code analyzers or refactoring scripts.

The chosen IDE for this project is **Microsoft Visual Studio Community 2019**. Web Toolkit library is supported on various platforms including Linux distributions (even less popular ones like ArchLinux, Slackware or Opensuse), Windows or other operating systems like Android, Raspberry Pi or even OS X. The consequence of choosing MSV IDE was using Windows platform and prebuilt Windows binaries of the WT library.

Microsoft Visual Studio is and IDE produced by Microsoft Company and it allows creating cross-platform software with graphical user interface. It basically supports every programming language but the basic package contains support for

- Microsoft Visual C#
- Microsoft Visual C++
- Microsoft Visual Basic
- Microsoft Visual J#
- Microsoft Visual Web Developer ASP.NET
- Microsoft Visual F#

MSVC also provides a lot of built-in features. A lot of them occurred to be significantly useful in the process of developing this application. A few most important ones are

- Debugger, Linker and Compiler
- Projects and Build Systems
- Writing and refactoring C++ code
- Code analysis overview
- Unit tests support
- Universal Windows Apps like command line appliactions

Boost C++ library

Boost is a collection of C++ libraries that enhances capabilities of C++ code development, which is also licensed by **Boost Software License**. For the project Boost is necessary to build Web Toolkit library as it's implementation uses Boost functionalities.

The most important features provided by Boost are

- Algorithms
- Concurrent programming
- Complex containers
- Correctness validating and enhanced unit testing
- Additional data structures (like bimap, fusion, tuple etc)
- High level programming and functional objects
- Parsers and graphs
- Meta-programming with templates

Boost library as a collection is not used in this project explicitly, as the application was developed in Microsoft Visual Studio environment with pre-built Windows binaries. However, for automation server, the project is built under linux machine where WebToolkit needs to be built manually, therefore boost is required. More details will be unveiled in "Automation server" chapter.

CMake

CMake is a cross platform tool that provides automatic management of compiler that builds the code of an application. It's role is to create a configuration for project files of popular programming environments, which then are used in a process of compilation. The main advantage of using CMake is it's independence of the compiler and the platform. However CMake as a standalone program creates files with rules for compilation dedicated to another program like IDE and it forms a unified building environment. CMake stands for *Cross-platform Make*.

The most important features provided by CMake are

- Platform independence
- Cross compilation
- Out-of-source building
- Building projects with complex catalog structure
- Unit testing support
- Detecting dependencies and outer libraries

This project required including CMake tool because of various dependencies and complex building due to including Web Toolkit and Boost.

To be able to explore favors that CMake offers one must create **CMakeLists.txt** file placed in the main catalog of the project. The core of this file is a simple scripting language that describes rules and defines variables telling the compiler how to link files and what should be the outcome of the compilation process.

Let's have an example project[1] which structure looks like below

```
.
|-- CMakeLists.txt
|-- build
|-- include
|   \-- Student.h
\-- src
    |-- Student.cpp
    \-- mainapp.cpp
```

Listing 1: Example CMake project structure

The content of ***.cpp** and ***.h** files is irrelevant here, at this point it could be any code. In the main directory we have a simple **CMakeLists.txt** which is used to build the project. The code of our CMakeLists script:

```
cmake_minimum_required(VERSION 2.8.9)
project(directory_test)

#Bring the headers, such as Student.h into the project
include_directories(include)

#Can manually add the sources using the set command as follows:
#set(SOURCES src/mainapp.cpp src/Student.cpp)

#However, the file(GLOB...) allows for wildcard additions:
file(GLOB SOURCES "src/*.cpp")

add_executable(testStudent ${SOURCES})
```

Listing 2: CMakeLists.txt of example project

The most important things that above CMake will perform (in order from the top) are

- Include directories to let the CMake know where headers files with definitions are
- Sources are also set, but the line is commented out as each file needs to be manually added in place
- file() command to add source files to project's GLOB SOURCES
- add_executable() which uses SOURCES variable in order to build executable program

Now to invoke building using CMake:

```
1  $ mkdir build && cd build
2  $ cmake ..
3  $ make
4  $ ./testStudent
```

Commands above will create build directory. From there cmake will create **Makefile** containing references to all sources and headers, which ultimately will create an executable that is ready to be run. This as a very basic example of how **CMake** works. In next chapters there will be more details about **CMake** and what is the main concept behind basing on the one used in project.

Others

Git BASH tool and *Web Toolkit* library are described thoroughly in next chapters.

2 WebToolkit C++ library

2.1 Library overview

Web Toolkit is an open-source C++ library that allows developers to create SAP web applications without writing single line of JavaScript code. The main concept behind [2]SAP (Single Page Applications) is they do not render HTML files, but instead they use asynchronous JavaScript (AJAX) to reload contents of the page in real time without refreshing the whole page. To some extent it might be a bit misleading, as with loading different content, the URL of the page changes as well, like in ordinary multi-page service.

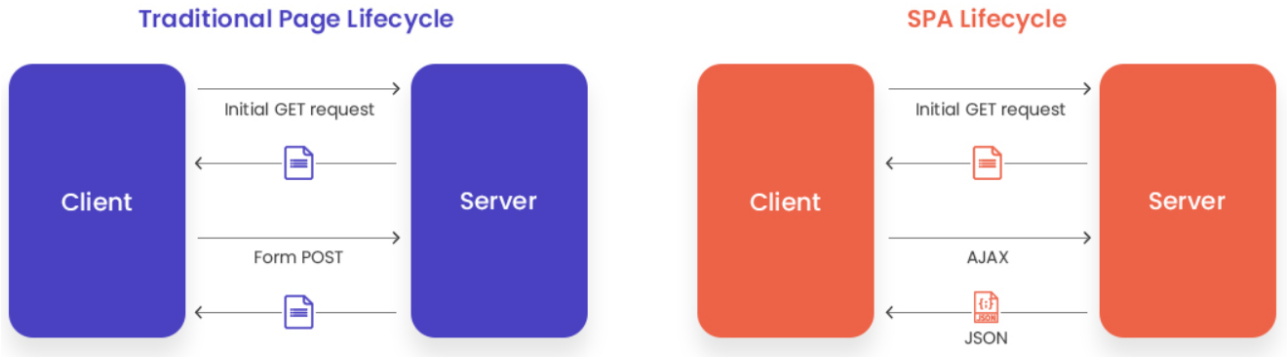

Figure 1: Traditional Page VS SPA[2].

As a SPA the application works on widgets which are the core of WebToolkit library. A developer should create a main widget, which is not visible to the user. Instead it works as a widget container that will hold other template widgets that will be pluggin in or plugged out depending on what content will be loaded on the page.

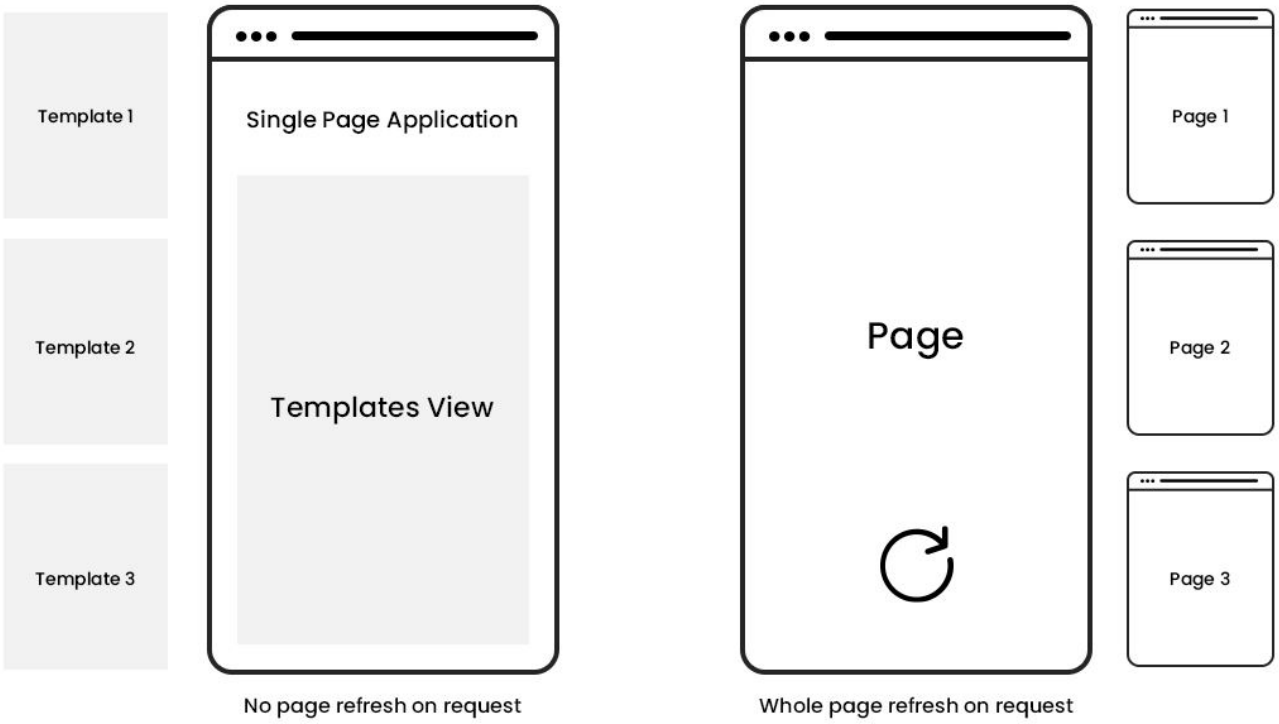

Figure 2: Views in Traditional Page and SPA[2].

There are a lot of modern services that use SPA technology like facebook, airbnb, twitter, paypal, gmail or even netflix. The biggest benefits of this web application model are quick loading time, good caching abilities, improved user experiance or rapid front-end development.

Beyond usual frameworks dedicated for SPA services are

- Angular.js
- React.js
- Backbone.js
- Ember.js

It is no secret that preferable programming technology stack for such development would be JavaScript, TypeScript, HTML, CSS and some back-end in Java, Python or PHP. With that being said, WebToolkit library seems to be very unorthodox tool for this task. The whole development process starts and ends with C++ code, which makes it interesting and unusual.

2.2 WebToolkit Features

WebToolkit library has a lot to offer. Now of course - creating web application in low-level language comes with a prize. It's biggest disadvantage is that while programming we have no control of HTML code. There is also a necessity of re-compiling the code with every change introduced where there is no such need when using regular tools. However - compilation time is quite fast and the library implementation is very robust.

Figure 3: WebToolkit logo and features[3].

Core Library

What is really important is that core library is fully open source and still under development. It's implementation allows to integrate it with 3rd party JavaScript libraries, which is a nice addition for developers that create services with more advanced front-end layer. What is more, WT is fully compatible with HTML5 and HTML4 browsers. Thus, as a hybrid single page framework it supports browser history navigation. The bottom line is, WT is a high performance library which is something you would expect from a C++ library. It is optimized when it comes to asynchronous I/O, multi-threading and throughout all rendering.

Event handling is completed with C++11 lambdas or bound object methods, which are a part of C++11 signal/slot API for responding to various events. Server-initiated updates are also available through WebSockets and automatic fallaback to AJAX. The handling itself is also completed with efficient synchronization of browser using session, which incrementally renders updates of application states. The concept of session is thoroughly described in implementation chapter.

Core library also provides 2D and 3D painting support to users and developers. Through WT API it is possible to generate simple graphics objects like HTML5 canvas, inline SVG or inline VML. Browser native vectors are also leveraged by rendering common image formats like PNG, GIF or even PDF. WT broads the variety of graphics support by also supporting hardware-accelerated 3D painting API like for instance server-side OpenGL.

Security

As a completed web-development library, WebToolkit provides built-in security layer to the applications which contains:

- kernel-level memory protection in isolated sessions
- TLS/SSL support
- Cross-Site Scripting (XSS) prevention

- Request Forgery (CSRF) prevention
- Application logic attack prevention
- DoS mitigation
- **Authentication module** (including support for OAuth 2.0 and OpenID Connect)

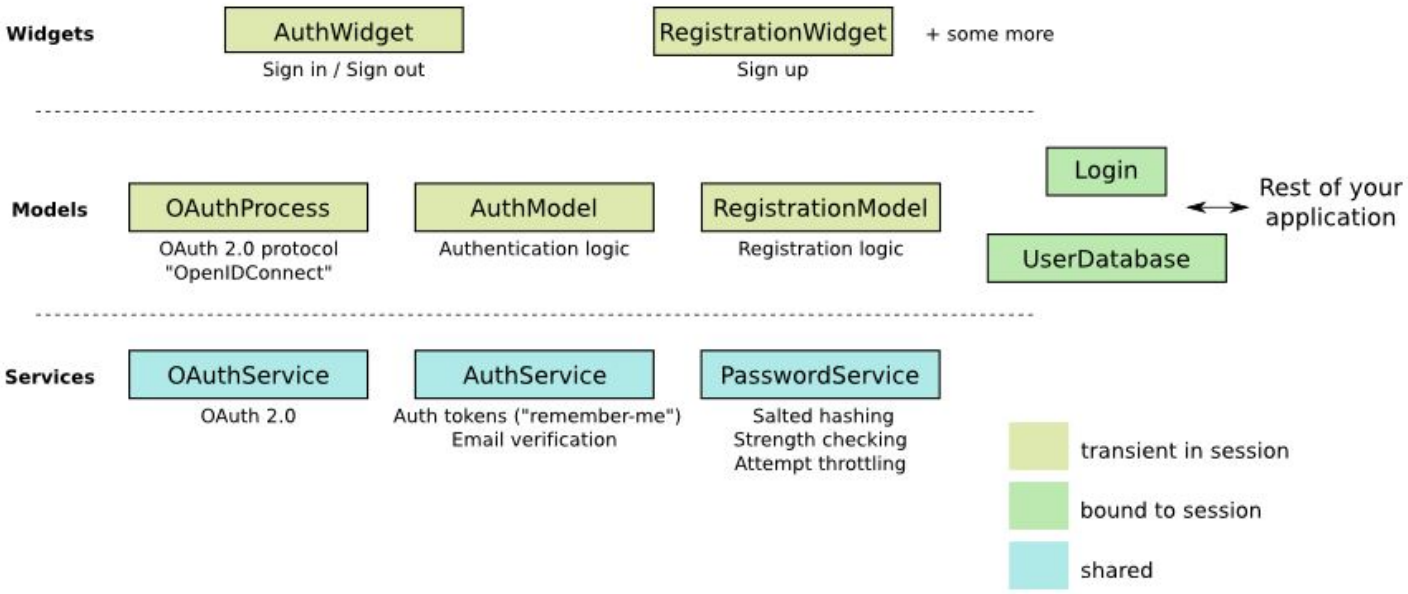

Figure 4: Main classes of the authentication module[4].

The authentication module itself occurred to be significantly important and helpful in Web Banking Application. Some of the classes which implement basic user features are presented above. What it means is that secure and robust service functionalities like signing in/out, registration or authentication are already provided.

Database support

As any other modern library, WT supports databases with a sub-library called Wt::Dbo, which by implementing ORM (Object-Relational Mapper) becomes a convenient way to communicate C++ application with SQL databases. What is more, Wt::Dbo as a self-contained sub-library is independent from Wt itself, which means it could be used for standalone back-end service. Other basic Wt::Dbo features are:

- Support for
  - Sqlite3
  - Firebird
  - MariaDB
  - MySQL
  - SQL Server
  - PostgreSQL
- Native SQL to query objects and fields

- Session with first-level cache
- DB transactions
- CRUD operations
- Many-to-One and Many-to-Many relations
- Just C++ without XMLs or Macros

Deployment

Extremely important thing in any application is process of Deployment. WT through various abstract interfaces connects with outer world. The basic one is built-in httpd but among others:

- Built-in **HTTPD**
  - Simple and high performance with multithreading and asynchronous I/O based on C++ asio library
  - Supports WebSockets and HTTP(s) with chunking and compression
  - Available for many platforms like Linux, Windows and any UNIX like systems
- **FASTCGI**
  - Integrated with common web servers like apache
  - Supported for Linux and UNIX
- **ISAPI**
  - Integrated with Microsoft IIS server
  - Completed with asynchronous API
  - Supported for Windows

2.3 Hello World Application

Lets go through simple Hello World applications just to get in touch with WebToolkit library environment.

```
#include <Wt/WBreak.h>
#include <Wt/WContainerWidget.h>
#include <Wt/WLineEdit.h>
#include <Wt/WPushButton.h>
#include <Wt/WText.h>

class HelloApplication : public Wt::WApplication
{
public:
    HelloApplication(const Wt::WEnvironment& env);

private:
    Wt::WLineEdit *nameEdit_;
    Wt::WText *greeting_;
};

HelloApplication::HelloApplication(const Wt::WEnvironment& env)
    : Wt::WApplication(env)
{
    setTitle("Hello world");

    root()->addWidget(std::make_unique<Wt::WText>("Your name, please? "));
    nameEdit_ = root()->addWidget(std::make_unique<Wt::WLineEdit>());

    Wt::WPushButton *button =
        root()->addWidget(std::make_unique<Wt::WPushButton>("Greet me."));

    root()->addWidget(std::make_unique<Wt::WBreak>());
    greeting_ = root()->addWidget(std::make_unique<Wt::WText>());

    auto greet = [this]{
      greeting_->setText("Hello there, " + nameEdit_->text());
    };
    button->clicked().connect(greet);
}

int main(int argc, char **argv)
{
    return Wt::WRun(argc, argv, [](const Wt::WEnvironment& env) {
      return std::make_unique<HelloApplication>(env);
    });
}
```

Listing 3: A complete "Hello world" application [5]

Now when it comes to building, it could be done locally by setting up a localhost HTTP server. Having IDE set-up it is done automatically. Otherwise while operating on UNIX-like systems:

```
1 $ g++ -std=c++14 -o hello hello.cc -lwthttp -lwt
2 $ ./hello --docroot . --http-address 0.0.0.0 --http-port 9090
```

Listing 4: Building "Hello world" application

The final result is:

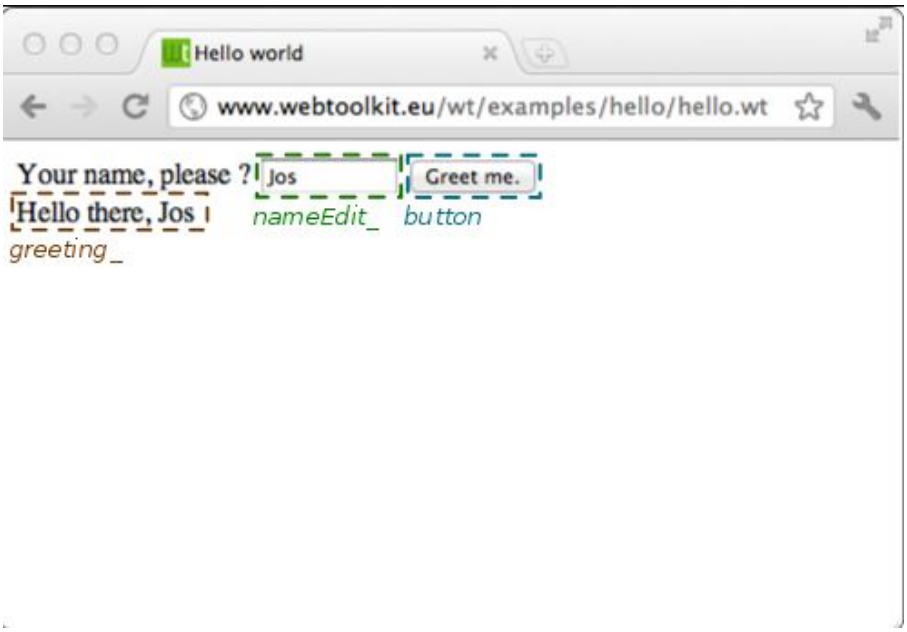

Figure 5: Hello Application on WT.

The above code presents basic concepts of the library workflow.

class HelloApplication A class that represents application and extends Wt::WApplication. Contains ctor that takes Wt::WEnvironment and private fields that represents input field and output greeting string wrapped in Wt::WText,

LINE 20: Sets title of appliaction which will be visible in web browser's card,

LINE 22-29: This part of the code handles adding widgets to the root which represents a container for the widgets. This is an implementation of SPA concept where we create widgets and plug them in or out, for instance by addWidget() function,

LINE 31: Lambda expression that returns WText string which called upon clicking the button,

LINE 34: Calling lambda expression from line 31 when button is clicked,

MAIN: Creating and running application.

2.4 Widgets gallery

WT also offers a widgets gallery, which is nothing else but a bootstrap where most common widgets are already implemented and ready to be used. Among them there are layouts, forms widgets, navigation widgets, trees widgets, tables widgets, graphics widgets, charts widgets and media widgets. Basically whilst going through development process one should keep in mind this feature as it may save a lot of time and a lot of coding.

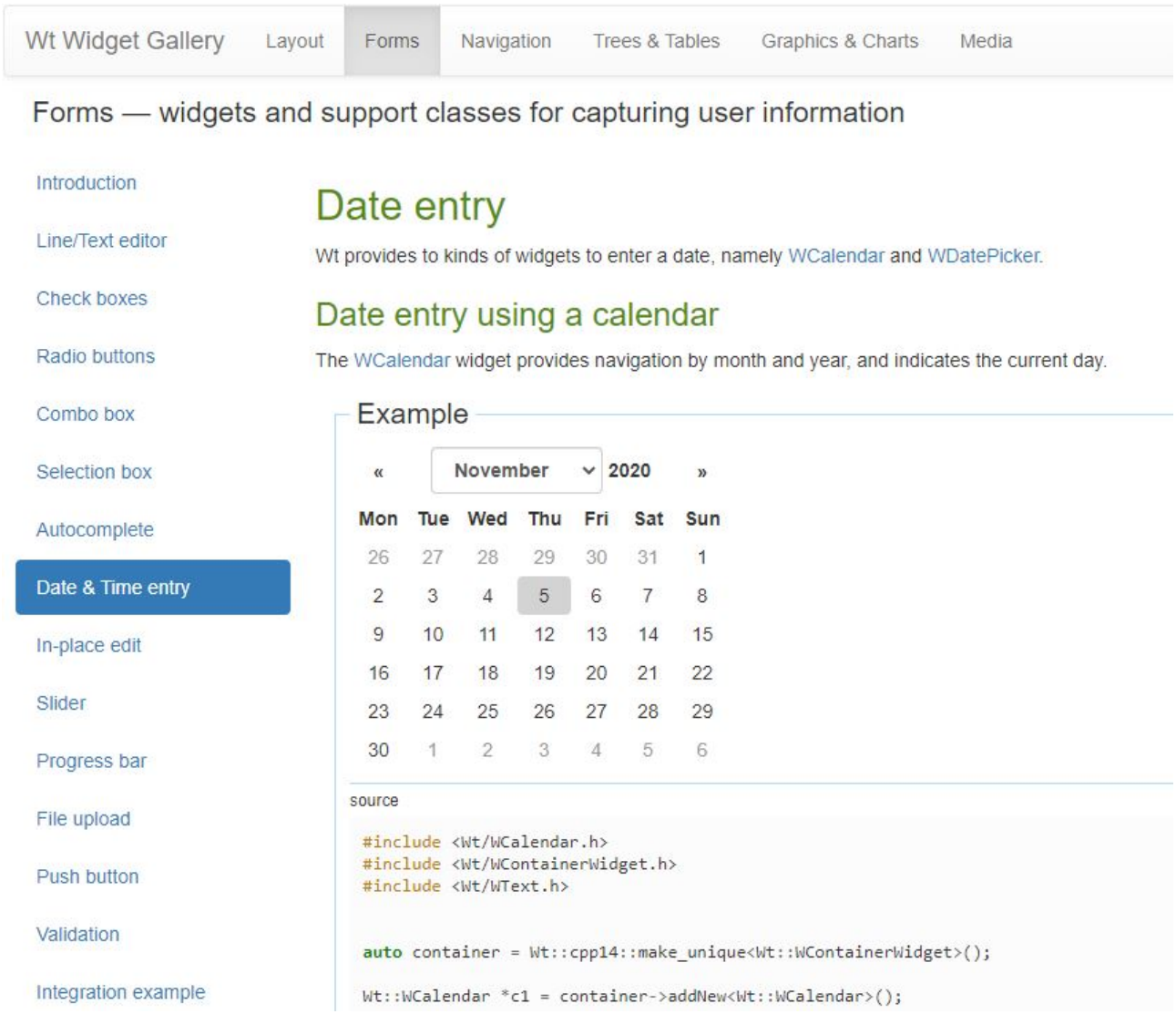

Figure 6: Wt Widget Gallery. [6]

3 DevOps layer

3.1 Distributed version-control system

3.1.1 GitHub

3.2 Proprietary issue tracking

3.2.1 JIRA Software

3.3 Other Atlassian tools

3.4 Containers

3.4.1 Docker

3.4.2 Containers vs OS-level virtualization

3.5 Automation server

3.5.1 GitHub CMake workflows

Example CMake file used in project

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.4)
Project(ConsoleApplication1)

# If Visual Studio IDE
IF(MSVC_IDE)
# Copy user file
  FILE(COPY ${CMAKE_CURRENT_SOURCE_DIR}/${PROJECT_NAME}.vcxproj.user DESTINATION ${CMAKE
ENDIF(MSVC_IDE)

# If Eclipse IDE
IF(${CMAKE_EXTRA_GENERATOR} MATCHES ".*Eclipse.*")
IF(${CMAKE_BUILD_TYPE} STREQUAL "Debug")
# Copy debug user file
    FILE(COPY ${CMAKE_CURRENT_SOURCE_DIR}/${PROJECT_NAME}-debug.exe.launch DESTINATION $
ENDIF()
IF(${CMAKE_BUILD_TYPE} STREQUAL "Release")
# Copy release user file
    FILE(COPY ${CMAKE_CURRENT_SOURCE_DIR}/${PROJECT_NAME}-release.exe.launch DESTINATION
ENDIF()
ENDIF()

# Copy resources to build tree
# FILE(COPY ${CMAKE_CURRENT_SOURCE_DIR}/resources DESTINATION ${CMAKE_CURRENT_BINARY_DIR

SET(ConsoleApplication1_SRC src/Main.cpp)

# If Visual Studio IDE
IF(MSVC_IDE)
SET(ConsoleApplication1_SRC ${ConsoleApplication1_SRC} src/Main.cpp)
ENDIF(MSVC_IDE)
```

```
ADD_EXECUTABLE(ConsoleApplication1 ${ConsoleApplication1_SRC})

ADD_SUBDIRECTORY("wt-4.3.1/" "Wt 4.3.1 msvs2019 x64/lib/")
# Set Wt include and library paths
INCLUDE_DIRECTORIES("Wt 4.3.1 msvs2019 x64/include/")

INCLUDE_DIRECTORIES("include")
FILE(GLOB SOURCES "src/*.cpp")
ADD_EXECUTABLE(ConsoleApplication1 ${SOURCES})
TARGET_LINK_DIRECTORIES(ConsoleApplication1 PUBLIC "Wt 4.3.1 msvs2019 x64/lib/")

TARGET_LINK_LIBRARIES(ConsoleApplication1
  debug wtd optimized wt
  debug wthttpd optimized wthttp
  debug wtdbod optimized wtdbo
  debug wtdbosqlite3d optimized wtdbosqlite3
  )
```

[1] Introduction to CMake, *http://derekmolloy.ie/hello-world-introductions-to-cmake*.

[2] Single Page Applications, *https://www.excellentwebworld.com/what-is-a-single-page-application*.

[3] WT Logo and Features, *https://www.webtoolkit.eu/wt/features*.

[4] Introduction to Wt::Auth, *https://www.webtoolkit.eu/wt/doc/tutorial/auth.html*.

[5] Wt Hello World Application, *https://www.webtoolkit.eu/wt/doc/tutorial/wt.html*.

[6] Wt Widgets Gallery, *https://www.webtoolkit.eu/widgets/forms/*.

Additional materials

- LaTeX wikibooks
  https://en.wikibooks.org/wiki/LaTeX