Technical Specifications

for

PC^2 Web Team Interface

Version 2.0

Prepared by:
Josie Isaacson
Andrew Combs
Ethan Holman
Danielle Frodge

EWU CS Department

Winter 2019/Spring 2019

Table of Contents

Summary

The System Internals Technical Specifications (“Technical Specifications”) describes the implementation and operation of the PC^2 Web Team Interface (“WTI”).

Background

Information regarding development for PC^2 WTI API and UI. This document breaks down what the developers had in mind when designing and explaining why such designs were used as opposed to others. This document is not set in stone and can be modified for future releases or versions. WTI is broken down into two fundamental projects: the API and the UI. The reason this feature was implemented was to allow an easy switch to a new UI if deemed necessary.

Software Requirements for End Users

These are the requirements for using the WTI:

- **Internet browser** -- we recommend using the following browsers:
    - Chrome version 72 or higher
    - Firefox version 67 or higher
    - Safari version 12.1 or higher
- **Screen resolution -** we recommend 1280x720 minimum

Development Platform

These are the software requirements for modifying or enhancing the source code.

UI Development

The UI project was developed using Angular 7.
In order to run the Angular development environment, you will need the following installed:

- NodeJS v10.8+
- NPM v6.9+
- Visual Studio Code was used as the IDE for development. While any editor that supports file system can be used, we recommend Visual Studio Code.

See package.json in the WTI-UI directory for further details on all packages necessary.
Further details for setting up the Angular development environment and installing dependencies can be found in the **readme.md** file located in the WTI-UI directory under the EWWebTeam project.

API Development

- Eclipse Java EE - version 8 or higher
- Jetty - version 9.4.14 - 9.4.15
- Swagger - version 1.5.21
- Jackson - version 2.9.8
- JavaSE - version 1.8
- Mockito-core - version 2.25.0
- Jersey - version 2.28

Architecture Overview

WTI is composed of an Angular front-end that communicates with a Java EE-based backend via REST calls and websocket messages. The front-end makes HTTP requests against the WTI-API. The API can send messages to the UI via websocket. For example, when a clarification has been judged, this event is communicated to the UI by sending a message across the websocket. The UI listens for this, and will make an HTTP call to get the updated clarification.

UI:

WTI-UI was generated with the Angular CLI and follows a standard directory structure. Here is a basic rundown of what modules exist and their purpose:

- **Core:** Contains service related code, primarily functionality that communicates with the API and authentication logic. Core is organized into the following subdirectories:
    - **auth:** contains auth guards, application authentication service, http interceptor to append team_id as header to all outgoing HTTP requests
    - **Abstract-services:** Main services that are abstracted and to be inherited. This was done to allow switching between a development mock and the actual backend API calls. (Allows for testing UI functionality without the need for a contest to be running)
    - **Models:** Data models
    - **Services**: Contains all app services (singleton's). Some of these extend services defined in abstract-services
- **Clarifications:** contains components for the /components route
- **Login:** contains login and logout components. This is where the websocket communication is established
- **Options:** contains components for the /options route

- **Runs:** contains components for the /runs route
- **Shared:** contains components that are used throughout the app (I.E., in multiple different modules)

API:

The API architecture is structured utilizing MVC pattern and follows it to the best of the developments ability. The packages are split based on the controllers made, the models, server creation, websocket communication, and tests.

- The controllers utilize inheritance with both controllers, team and contest, sharing similarities between them i.e. connection to PC^2.

Special Note Regarding Pinging the API: Each endpoint will need to have a special key to be able to ping an endpoint. If the key is not received then the API will throw an error. Reason for this was to store each user connection into a cache like system (please review caching) that has a key value pair of each user and each server connection to allow users to interact with the PC^2 system.

  - **TeamController**: Holds all resources regarding teams. Teams controller is broken down into what makes sense to what teams will have. For example, each team will have a set of runs (whether teams are getting their runs or posting to the server for a new run).
  - **ContestController**: Holds all resources regarding what the contest holds. This allows any UI to ping the server to request contest information. For example, receiving problem set will be received at this controller location.
  - **MainController:** Holds everything that is similar between controllers. For example, each team will have a connection to the API. Here the cache will store that information and the controllers will interact and utilize it. It shall be inherited for any controller that is made within WTI-API.
- **Websockets:** Websocket communication was created utilizing a mediator type pattern. Here the mediator controls the communication between all the sources connected to WTI-API.
- **Config**: Config file holds all information regarding server start up. WebServer file is the server itself. Here information regarding where application state goes, endpoint information is stored, and where Swagger location shows up can be located in this file. Also holds the file that utilizes obtaining users IP address that will be passed on to WTI-UI.
- **JettyConfig**: Here information regarding overridden mapping errors (or any other Swagger error or Jetty configuration needed will be located here). Errors named Jackson are overridden mapping errors used by Jersey when information mapped to a POJO is considered bad. **Why was this needed?** The information

here is needed due to the way Jersey mapping is used. Jersey does not handle mapping and utilizes Jackson to be able to map to POJOS. Without Jackson, Jersey will behind the scenes create an object and then later map with information to object, if the information is bad you will have an empty object and thus internal server errors. Jackson was overridden by wrapping Jersey in JacksonFeature and JacksonJaxbJsonProvider found in webserver.

- **Services**: Services are a way for WTI-API to have communication from the PC^2 contest and WTI-API. Each service has hook methods that have the specified one overridden. If other ones are needed it is simple to add code to allow the websocket carry that information off to where it needs to go.

Interfaces

WTI-API utilizes services to have communication between PC^2 and WTI-UI. Upon logging in, WTI-API creates a new service implementing PC^2's interfaces and subscribes them to PC^2. WTI services utilizes PC^2 hooks to transfer the data from PC^2 to the UI with WTI-API websockets.

Dependencies

Node.js - (see development section) - Used to build UI project and download Angular framework.
Apache Ant - (see development section) - Used to build and move project together.

Shortcomings

- **Test problems not fully tested:** Problems arose when testing the submit test for PC^2. Implementation of test was not fully implemented on PC^2 side so errors arose when fully able to test. Users may select a test and submit it and PC^2 is supposed to throw a mock PC^2 test. However, PC^2 instead was throwing an error when testing the endpoint that would end up crashing the system.
- **Allow saved drafts of user clarifications:** When first designing PC^2 WTI-UI we designed it to allow for users to be able to save their clarifications when canceling or navigating away. When coming to implement this feature, the complexity became too overwhelming and it was decided to be considered an extra feature that was not implemented. In the future, to implement this feature developer will need to store users

sessions to be able to grab that information later on (please see session storing section of shortcomings) and then store that object back when users select clarifications.

- **Allow storing user session:** When designing WTI-UI user sessions were not implemented to begin with, however, was talked about. The designs were overlooked when it was implemented and therefore became forgotten. Users who refresh the page lose session state. Later in development it was implemented. However, when we were developing it weird bugs started to arise and it was later removed to avoid the crashing of program. Users will have one session and will need to re-login if refreshing page or exiting out of the window.
- **Change user passwords:** When developing and designing WTI, documents will show a feature of changing password. This was not implemented on PC^2 end and by the time we started to work on it we realized it was too late to be able to get it implemented. WTI-API and WTI-UI both implemented hollow endpoints for future developers to work on.
- **Build.xml will not work in Eclipse:** When working on WTI all development was processed with command line arguments and was not at all tested to work with Eclipse. When doing final user agreement testing, build continued to fail with client. Initial thought as to why this is is because Eclipse runs its own environment and does not at all have any access to your environment variables. Through Google searching we discovered to fix the underlying issue is to add node to your Eclipse environment setup that will then have access to node.
- **Build.xml will only work for Windows:** When developing build.xml most development was done on Windows machines. The change needed is when building with Ant to check the OS system and to see what it is running. Then switch to that build system when running the “ant build” command.

Error Handling

WTI-API was first developed as a dynamic web project (different from a regular Java project). The reason this came to be was a change in requirements when developing. Changes could not be made since development was too far down the road. Errors may occur when developing within Eclipse:

- **Random loss of dependencies or files missing when running application or updating it**. If this occurs navigate to work bar and select **Project** -> **Clean…** -> **Select WTI-API** -> **Clean** in popup window.
- **Error on project:** Project seems to follow an error that looks to be tied to a docxlet. This is regarding the buildpath when the project is shown in Eclipse. As research showed this error seems to have to be with the docxlet is missing and is required by Eclipse. To fix the problem a new dependency needs to be placed. This is avoided due to one library needed to fix graphical problem or library that is not associated to project. Project still works without dependency and was opted out to avoid it.

- **Error from PC^2 test runs:** When users utilize WTI there is an issue when submitting a test run. Somehow during submitting the test run PC^2 will attempt to access the get all test runs from PC^2 API. However, it will fail from getting access and throw a random null pointer exception. This will then kill the endpoint since PC^2 WTI API will continue to attempt to access that run for mockedruns and instead continuously throw an error. After typing this and thinking a bit this would make no sense. There seems to be an error within the code for arraylist that could also be throwing an error. (a misunderstanding of arraylist api).
- **Error with logs:** WTI on first initial start throws an error when trying to find logs. The reason why is because WTI is looking for log directory which did not get made. To fix this, in the logging file on first startup create the logs folder and it will then be placed into logs.

Regeneration

To build project it will be utilizing ant build: build.xml that resides inside of WTI-API:

1. Navigate to **WTI-API**
2. **Optional:** Open up WebTeamInterface.ini and fill in desired port, desired API name, etc. (see **WebTeamInterface.ini)**
3. Run **ant build**
4. Once project is built: Navigate to **build -> lib**
5. Run **java -jar -Djdk.crypto.KeyAgreement.legacyKDF=true WebTeamInterface-1.0.jar**
6. Application will be running on specified port; open up desired browser and navigate to it to be able to see a running application.

Installation

Refer to the PC^2 Web Team Interface Installation Guide.