BECS with Flask

Documentation

Submitted by: Michael Afonin, 310514997

**Overview**

The following project is a fairly basic implementation of BECS – Blood Establishment Computer Software. The program runs in a server-client configuration, where the server is a Linux-based machine and the client is any web browser, desktop or mobile.

The software, referred in this document as BWF (For BECS with Flask), contains the basic functionality of a medical computer software in accordance with HIPAA, Part 11 and main BECS standards:

1. Blood type correlation – BWF supports requests based on the patient’s blood type, returning the correct blood type to be used in the transfusion process. The decision is made via a calculation of the most common blood types and the amounts held in the coolers or freezers.
2. Blood storage – aiming to preserve as much blood as possible, BWF will check daily at 2am the blood packs stored in the coolers and will transport to the freezers the older (30 days and above) blood packs. The cost and procedures for freezing a blood pack are omitted.
3. Prioritization of pack withdrawal – BWF gives the option to request blood packs based on the severity of a given incident. A normal priority, as in planned operations or known issues with a given patient, will produce a frozen blood pack, as the procedure is less time-dependent. High priority, dubbed “emergency” in BWF, will warrant the quickest response and thus the main source of blood packs is the coolers. Trauma and ER rooms, as well as mass casualty incidents (MCIs) will provide an emergency withdrawal with frozen packs as the secondary source of blood packs.
4. Data logging and export (Part 11) – BWF logs each and every action of the user, from the login attempt up until he/she logs out of the system. These actions include, but are not limited to, logging in and out of the system, web page redirections, blood withdrawal, blood submission, mass withdrawal for MCI, personal details updates and log exports. The data can be exported by an administrator-level user in the personal details page.
5. User privilege levels (HIPAA) – in order to distinguish users with different levels of action privileges, three levels of data access are provided:

* Student – can view data but can’t perform any actions with blood packs.
* User – can view perform any operation with the blood packs.
* Administrator – can perform any action with blood packs. Additionally, he is the only one who is able to raise and lower privileges, delete users and export log files.

A user signing up to the system will be delegated a student-level privileges, and an administrator will have to update them later.

1. Secure password storage and transfer – BWF implements a simple encryption of the passwords on the relevant pages. The encryption is done using MD5, a simple encryption that is to be changed to a more secure one. The password encryption is done on the user side, so the data is sent to the server in an encrypted form. No decryption is done on any stage of the work.
2. Time-limited standby for users – since BWF contains sensitive medical information, the users have a limited time span of 24 hours between any two actions. For every action the user makes, his last action time stamp is checked, and in case of a 24+ hour time span between two consecutive interactions BWF will automatically log the user out of the system and request a log-in procedure.

**Development stages**

1. Basic BECS:

Principal work on BWF started by adding the most basic functionality – the general user had the ability to submit, withdraw and mass-withdraw blood packs, as well as view the current blood bank inventory. The withdrawal logic, based on blood type amounts and distributions, was introduced in this stage. A basic confirmation page was implemented to notify the user about successful or failed action. No user privieges were implemented at this point. Another basic feature was considering all the blood packs as frozen, so no time stamps were used and the packs were implemented as counters for add/withdraw functions.

The client side for BWF had a small number of pages with basic design – mobile web browsers treated the page as a full desktop page.

The server side implemented a REST API structure based on Flask framework. Page redirection was performed via one of the API endpoints in the server.

A basic Docker container was used to run the server.

1. Part 11 and HIPAA:

The first major update was adding basic implementation of the HIPAA standards. The server at this point expanded significantly, adding support for more complex web page redirections and better messages for the user. At this point the user could sign up and log into the system, and password validation was implemented. Basic validation of users as logged in or logged out was used, without time stamps for each action.

The second major update was implementing privilege separation of actions for each of the three user levels – “student”, “user” and “admin”, along with privilege-level validation for each user. A time stamp for one’s last action, along with the 24-hour validation, was implemented. Additional checks for existing users, logged out users and unprivileged actions were also introduced in this update. As a next step in development, users could update their details in their profile page, along with the options reserved for the administrators.

The third and final major update on this stage was adding Part 11 support. The users were now logged in the log file, which was produced for every day separately. Log file lines contained the time, ID and action description for each user in the system and the actions he performed. A log export page was added, containing a field for the requested date. Additional changes were made to the responsiveness of the web pages, which could now work well on mobile devices as well as desktop browsers.

1. Extra features (final part):

In the final part of the project, a few major change were introduced:

* The blood packs no longer were “only” frozen, and the ssubmission was made to the cooled packs. They were divided into frozen and cooled, and a timed transfer of blood packs was implemented. The operations were parallelized using multi-threading.
* Blood pack withdrawal was prioritized and a logic of withdrawal based on emergency levels was implemented.
* Users’ passwords were protected in transfer by encryption on the client side.
* Better, clearer messages were shown all around the system, along with the existing log files on the log export page.
* An option to populate the blood bank with randomized initial data, or in the real case – with empty coolers and pack counts, was introduced.
* Several cosmetic changes were made to the code, including general cleanup, simpler functions, clearer logs and comments where needed.

Some of the changes are not listed as they were insignificant in this update’s scope.

1. Future development:

As the project operates on a somewhat basic level, and no overrides are implemented for BWF’s choices of blood manipulation, some changes may be introduced in the future:

* More granular control over blood withdrawal (donor’s and patient’s sex, pregnancy, blood condition etc.) and overrides of the predetermined choices based on the user’s considerations.
* Additional parameters for the blood packs storage, in addition to the sole existing blood type input.
* Mass export of log files by time intervals, as in the present state only a single file can be exported at once.
* Security checks for users’ inputs to minimize risks of cybersecurity threats.
* Stronger encryption of passwords and better implementation of user identification (i.e. using cookies and implementing better-timed intervals between actions).
* Migration to an external SQL/noSQL database for blood pack logging.

**First Setup**

BWF can be run on first setup in two configurations:

1. Empty database’ to be populated with data gradually.
2. Database with initial amounts of data (mock setup).

By default, for the software to run in test conditions, the random mock data is created and BWF is initialized with somewhat full databases. In order to run the software in a real environment, a modification to the Docker file is required:

In Dockerfile, the last line: CMD [ "api/requests.py", "-f" ]

Should be replaced with: CMD [ "api/requests.py"]

This will change the environment so that the database filler function is not run on first setup.

The software was built, tested and run on a desktop Ubuntu 20.04, and thus the support for this system is guaranteed. Other systems, especially based on Arch and Red Hat, were not used and the software can’t be guaranteed to run on them in the correct way.

The next steps to perform are:

1. Open the terminal in the project’s root folder (it should contain the Dockerfile and the “api” directory).
2. Check if Docker is installed on the system: sudo apt install docker.io
3. Run the “docker build” command to build the container: docker build -t becs:latest .
4. Run the “docker run” command to start the server: docker run -p 5000:5000 becs:latest

All input from the internal processes and the server actions will be printed on the terminal screen.

To access the system please enter in your browser of choice:

http://localhost:5000

The main existing user, who is an administrator by default, has the following details:

* id: 123456789
* password: 1234

To stop the server, please use Ctrl+C twice - a second or two apart - since the multi-threading is still somewhat buggy.