# 3K04-PacemakerProject DCM (Group 10)

**Graham Power, Hamza Ashraf**

**Oct 29, 2020**

# CONTENTS:

CHAPTER

ONE

# FLASKAPP

## 1.1 flaskapp package

### 1.1.1 Subpackages

#### flaskapp.config package

#### Submodules

#### flaskapp.config.config_manager module

#### Config Manager

A collection of configuration setup functions, to initialize the config variables and setup application logging.

`flaskapp.config.config_manager.`**`init_config`**`(`*cfg_files=[]*`)`

init_config Initializes the application configuration given a list of configuration files

Will read all configuration files, creating a application varaible matching the value in the files. File locations in the list must be specified relative to the ~/3K04-Pacemaker/DCM/flaskapp/config directory. The order of the files in the list matters, as any repeat variables will have their previous values overriden. This allows for a default application configuration to be specified and later overriden by an end user.

**Parameters** **`cfg_files`** (*`list, optional`*) – A list of config file locations, defaults to []

**Returns** A handler for the applications configuration manager

**Return type** `configparser.ConfigParser`

`flaskapp.config.config_manager.`**`init_logging`**`(`*config*`)`

init_logging Initializes the application logging

Will create two logging handlers, a file and stream hendler. This ensures all log output is sent to both the console at runtime as well as a dated log file located in the /logs directory. Both handlers are created as defined in the Logging section of the application configuration files.

**Parameters** **`config`** (`configparser.ConfigParser`) – The handler for the application configuration

**Returns** A handler for the applications logging manager

**Return type** `logging.Logger`

### flaskapp.config.decorators module

### Decorators Library

A collection of decorators used by the main app, created to avoid the unnecessary reproduction of code.

`flaskapp.config.decorators.`**`login_required`**`(`*f*`)`

login_required Ensures that a user is logged in before granting access to user restricted pages

Will check if a user is logged in and only allow a redirect to the user restricted page if there is a user logged in. If no user is logged in it will chenge the redirect to the home page and flash a login required messege, letting the user know they must login before accessing that endpoint. NOTE: This function should never be called as a function, only used as a decorator above functions who require its functionality to be implemented on entry. Always use a decorator (i.e. @login_required) to invoke this function.

**Parameters** **`f`** (*`function`*) – The function being decorated. This will be automatically filled when using the @ tag

**Returns** The wrap function, whose contents are the input function, modified to add the functionality of this decorator

**Return type** function

`flaskapp.config.decorators.`**`logout_required`**`(`*f*`)`

logout_required Ensures that a user is logged out before granting access to non user restricted pages

Will check if a user is logged out and only allow a redirect to the non user restricted page if there is no user logged in. If a user is logged in it will simply log them out automatically before redirecting to the requested endpoint. No logout action on the part of the user is necessary. NOTE: This function should never be called as a function, only used as a decorator above functions who require its functionality to be implemented on entry. Always use a decorator (i.e. @logout_required) to invoke this function.

**Parameters** **`f`** (*`function`*) – The function being decorated. This will be automatically filled when using the @ tag

**Returns** The wrap function, whose contents are the input function, modified to add the functionality of this decorator

**Return type** function

### Module contents

### flaskapp.data package

### Submodules

### flaskapp.data.database module

### Database Library

A collection of functions capable of interacting with a sqlite3 single file databse. NOTE: Users are returned as lists, whose entries are in the following order

[ _userid, username, password, LowerRateLimit, UpperRateLimit, AtrialAmplitude, AtrialPulseWidth, AtrialRefractoryPeriod, VentricularAmplitude, VentricularPulseWidth, VentricularRefractoryPeriod ] `flaskapp.data.database.`**`find_user`**(*cursor*, *username=None*, *password=None*)

find_user Given search parameters will find all matching users in the database

Given one or more of the optional search parameters will return a list of all users matching that search criteria. Accepted search parameters are username and password. If neither optional parameters are given, the function will return None

**Parameters**

- **`cursor`** (`sqlite3.Cursor`) – The cursor handler for the database to search for the user in
- **`username`** (*`str, optional`*) – The username of the user to search for, defaults to None
- **`password`** (*`str, optional`*) – The password of the user to search for, defaults to None

**Returns** A list of tuples containing all users matching the search query

**Return type** list

`flaskapp.data.database.`**`get_rows`**(*cursor*)

get_rows Returns the number of rows (.i.e users) in the database

**Parameters** **`cursor`** (`sqlite3.Cursor`) – The cursor handler for the database

**Returns** A list of tuples containing all items matching the search query

**Return type** list

`flaskapp.data.database.`**`get_user`**(*cursor*, *id*)

get_user Reuturns a complete users information given their unique ID

Return type is a list of users. Since the search is done by unique ID, this list is garunteed to be either of length one, if a user with matching unique ID is found, or zero, if no user with matching unique ID is found.

**Parameters**

- **`cursor`** (`sqlite3.Cursor`) – The cursor handler for the database the user can be found in
- **`id`** (*`int`*) – The unique ID of the user to search for

**Returns** A list of tuples containing all items matching the search query

**Return type** list

`flaskapp.data.database.`**`init_db`**(*file*)

init_db Initializes a database located at a given file location

The file location should be specified relative to the ~/3K04-Pacemaker/DCM/flaskapp/data directory. The file should also have a supported sqlite3 extension (.db .db3 .sdb .s3db .sqlite .sqlite3) and if the file does not already exist it will be created and populated with a new databases.

**Parameters** **`file`** (*`str`*) – The relative file location of the single file sqlite3 database

**Returns** A tuple containing the databases connection handler and cursor (sqlite3.Connection, sqlite3.Cursor)

**Return type** tuple

`flaskapp.data.database.`**`insert_user`**(*conn*, *cursor*, *username*, *password*)

insert_user Given a username and password of a new user, will insert the user into the database

This function will create a new entry in the database of a user with the given username and password. Only the users username and password are initialized upon user creation, the pacemaker parameters will default to None, forcing the user to manually enter their parameters. NOTE: This function does no check for conflicting users in the database before inserting a new user. It is up to the user of this function to check for conflicts (if they wish to do so) before calling this function.

**Parameters**

- **`conn`** (`sqlite3.Connection`) – The connection handler for the database to insert a new user into
- **`cursor`** (`sqlite3.Cursor`) – The cursor handler for the database to insert a new user into
- **`username`** (*`str`*) – The username for the new user
- **`password`** (*`str`*) – The password for the new user

`flaskapp.data.database.`**`update_pacemaker_parameters`**(*conn*, *cursor*, *id*, *values*)

update_pacemaker_parameters Given a list of pacemaker parameters, updates the database values

When given handler to the database and the unique ID of the user being affected, will update the users pacemaker parameters to match the input list. NOTE: No complete check is done to ensure the validity of the input, it is up to the method user to ensure the lists correctness.

**Parameters**

- **`conn`** (`sqlite3.Connection`) – The connection handler for the database whos contents to change
- **`cursor`** (`sqlite3.Cursor`) – The cursor handler for the database whos contents to change
- **`id`** (*`int`*) – The unique ID of the user whos parameters should be changed
- **`values`** (*`list`*) – A list of pacemaker parameters, whos order matches the databases contents

## flaskapp.data.user module

### User Class

The class to represent a user of this application. Capable of initializing and accessing the database specified in the application configuration, as well as loggin in and out, creating an account, and modifying the pacemaker parameters of the currently logged in user.

**`class`** `flaskapp.data.user.`**`User`**(*config*)

Bases: `object`

This is a class representation of a simple flask app user

The class to represent a user of this application. Capable of initializing and accessing the database specified in the application configuration, as well as loggin in and out, creating an account, and modifying the pacemaker parameters of the currently logged in user.

**Parameters** **`config`** (class:*configparser.ConfigParser*) – A handle to the `configparser.ConfigParser` config object initialized by the main application on startup

**`create_account`**(*username*, *password*)

create_account Creates a new user account

Checks the database to ensure no user with the same username exists, and that the maximum allowable local-agents has not been exceeded (defined in the application.ini). If no conflicts exist, a new user is created then both inserted into the database and logged in.

**Parameters**

- **`username`** (*`str`*) – The username of the new user being created
- **`password`** (*`str`*) – The password of the new user being created

**Returns** True if the account creation was successful, False otherwise

**Return type** bool

**`get_pacemaker_parameters`**()

get_pacemaker_parameters Returns a dictionary of this users pacemaker parameters

**Returns** A dictionary of this users pacemaker parameters

**Return type** dict

**`get_username`**()

get_username Returns this users username

**Returns** This users username

**Return type** str

**`is_loggedin`**()

is_loggedin Checks if this user is logged in

**Returns** True if the user is logged in, False otherwise

**Return type** bool

**`login`**(*username*, *password*)

login Attempts to log a user in, given their username and password

Searches the database to check if the user with matching username and password exists. No conflict management (i.e. ensuring only one user matches that username) is necessary since it is handled on account creation. If a matching user is found the `data.user` is updated with that users information and the user is logged in.

**Parameters**

- **`username`** (*`str`*) – The username of the user trying to login
- **`password`** (*`str`*) – The password of the user trying to login

**`logout`**()

logout Logs out the currently logged in user

**`num_parameters = 8`**

**`parameters = {'Atrial Amplitude': None, 'Atrial Pulse Width': None, 'Atrial Refracto`**

**`update_all_pacemaker_parameters`**(*values*)

update_all_pacemaker_parameters Updates all pacemaker parameters

Given a dictionary of pacemaker parameters, in the same order as the values of the pacemaker parameters dictionary, will update every value of the dictionary. NOTE: No complete check is done to ensure the validity of the input, it is up to the method user to ensure the dictionaries correctness.

**Parameters** **`values`** (*`dict`*) – A dictionary of the updated pacemaker parameters

**Returns** True if the pacemaker parameters were updated sucessfully, False otherwise

**Return type** bool

**update_pacemaker_parameter**(*key*, *value*)

update_pacemaker_parameter Updates a single pacemaker parameter

Given a valid key (one already contained in the pacemaker parameters dictionary), will update the value of that key with the passes in value.

**Parameters**

- **key** (*str*) – A key already contained in the pacemaker parameters dictionary
- **value** (*int*) – An updated value for the associated key

**Returns** True if the parameter was sucessfully updated, False otherwise

**Return type** bool

#### Module contents

#### flaskapp.tests package

#### Submodules

#### flaskapp.tests.test module

**class** flaskapp.tests.test.**FlaskTestCase**(*methodName='runTest'*)

Bases: `unittest.case.TestCase`

**test_login**()

**test_login_correct**()

**test_login_incorrect**()

**test_login_loads**()

**test_logout_works**()

**test_user_blocked**()

#### Module contents

## 1.1.2 Submodules

## 1.1.3 flaskapp.app module

### Main Application

The main implementation of the DCM flaskapp. Handles rendering off all the endpoints as well as communication between the frontend and backend.

flaskapp.app.**home**()

home The route to the homepage of the flask application

Renders the homepage of the flask app and manages post requests. Supported post request are ‘login’ and ‘account creation’ requests, which when completed successfully will redirect the user to their user specific page. Any failed post request will result in an error message flashed to the screen.

**Returns** The render template used by the homepage, in this case the ‘index.html’ template

**Return type** `flask.render_tmeplate`

`flaskapp.app.`**`logout`**`()`

logout The route to the logout page of the flask application

This page acts as an intermediary between a user page that requires login and the homepage of this application. This page logs the user out and immediately redirects to the homepage, flashing a logout message after the redirect.

**Returns** A redirect to the homepage of the application

**Return type** `flask.redirect`

`flaskapp.app.`**`open_browser`**`()`

open_browser Opens the flask app automatically in the web brower.

So the user doesn't have to memorize the url and port the application is being hosted on.

`flaskapp.app.`**`user_connect`**`()`

user_connect The route to the user connect page of the flask application

Renders the user connect page of the flask app. This page allows the user to view the status of the pacemaker in different pacing modes. Pacing modes can be changed via a selection box at the top of the page. NOTE: This page will correctly change between different pacing mode states, though these states and their corresponding rendered templates are black as no serial communication with the pacemaker has been implemented. As a result changing modes will have little visible effect on the rendered content of the application.

**Returns** The render template used by the user connect page, in this case the 'user_connect.html' template

**Return type** `flask.render_tmeplate`

`flaskapp.app.`**`user_page`**`()`

user_page The route to the user specific page of the flask application

Renders the user specific page of the flask app. This page acts as an intermediary between the 'login' state and any other states which require login, such as the user parameters and pacing mode specific states.

**Returns** The render template used by the user specific page, in this case the 'user.html' template

**Return type** `flask.render_tmeplate`

`flaskapp.app.`**`user_parameters`**`()`

user_parameters The route to the user parameters page of the flask application

Renders the user parameters page of the flask app. This page allows the user to view and modify their pacemaker parameters through a submittable form. Users can modify some or all parameters and submit the changes through a post request. Changes will be made immediately in the flask app and database, and the user parameters page updated with these changed values.

**Returns** The render template used by the user parameters page, in this case the 'user_parameters.html' template

**Return type** `flask.render_tmeplate`

### 1.1.4 Module contents

# CHAPTER TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

# INDEX