Backend requirements

Milestone 2 & 3: Backend

-Requirement analysis:

- °Service users will have the ability to create and log into an account. This is neccessary for proper user-content identification
- °Service users will also have the ability to retract their data, as well as alter that data, if desired
- °Service users will be able to navigate between archive functions, deckbuilder services as well as account services
- °Archive functions are diverse. This will ensure that the desired data can be easily found by service users
- °Deckbuilder services need to be restricted by creating and logging into a archive_user - account first. This level of User Access Control will ensure, that user content can only be changed by the same archive_user that created that particulaur content
- °Data is stored in 5 different sessions in order to clearly communicate user created content to the database (backend ONLY)
- °Service classes are needed for a streamlined communication with the database
- °Gateway classes are needed to POST and GET database information
- °Controller classes are needed to handle the logic circuit of the interface
- °The index.php file is the interface file, that routes the service users input parameters to the database

-Planning MS 2:

°user:

1. Creating a user requires an e-mail-, a username-, a password-, as well as a confirm-input that needs to be the exact same value as the password-input. The password input will be hashed, before it is posted in the database
2. Logging in a user will start a session, that will hold the users ID, in order to access deckbuilder-services (backend ONLY)
3. Logging out a user, will delete the user session and its contents (backend ONLY)
4. Updating a users information will not destroy the current session and its contents (backend ONLY)
5. Deleting a user, will result in automatically destroying the current session and all its contents as well as all database entries that use that user_id, including the cards_decklists-, decklists- and archiveuser-tables (backend ONLY)

°card_archive:

1. Six filter options are required in order to browse the card_archive in v.1.0.0, depending on the service users needs
2. No archive_user needs to be created or logged in to access the card_archives
3. The following filter options will be present in v.1.0.0:
   card_name LIKE „X“
   mana_value LIKE „X“
   cmc LIKE „X“
   card_type LIKE „X“
   super_type LIKE „X“
   sub_type LIKE „X“

4. A single card and all its database values can be displayed, by clicking a link that is created as a part of the search result

°deck_archive:

1. Three filter options are required in order to browse the deck_archive in v.1.0.0, depending on the service users needs
2. No archive_user needs to be created or logged in to access the deckbuilder_archives
3. The following filter options will be present in v.1.0.0:
   user_id LIKE „X“
   deck_name LIKE „X“
   format LIKE „X“
4. A deck and its contents can be displayed, by clicking a link that is created as a part of the search result. This link will use an INNER JOIN command to use the card_archive and its services to display the contents of the desired decklist in its entirety

-Planning MS 3:

°**deckbuilder**:

1. Archiveuser can create a decklist. The stored user-session data is read and used to fill the required user_id column in the database. Name as well as format will be chosen by the archive_user and will be input as a parameter
2. Archiveusers can switch between their created decklist, by selecting a deck_id of their own creation via input parameter. The Gateway class contains a function that checks the input deck_id, by accessing all created decks of the archive_user and comparing their entries to the selected deck_id
3. When selecting or creating a deck four sessions will be initiated. The first will store the deck_id of the selected deck. The second, third and fourth represent „containers“ for storing contents from and properly pushing these contents into the database. These are „main“, „side“ and „maybe“
4. The Gateway class responsible for communicating the user created lists to and from the database will include seperate functions for the „main“, „side“ and „maybe“ containers, to ensure no SQL-Injections into the database. Values for „sideboard“ and „maybeboard“ will either be set to „Yes“ or „No“ by the Gateway class. This is required for proper identification in the database
5. When selecting a deck_id, all previous sessions are emptied, destroyed and then reinititated. If database content for a deck_id is stored in the cards_decklist table, these contents will be used to fill the „main“, „side“ and „maybe“ sessions based on the UNIQUE constraint of their respective database entry (for example: SELECT * FROM cards_decklists WHERE deck_id = :deckID AND side_board = „No“ AND maybe_board = „No“; this will save/hold the respective data in the „main“ session)
6. Displaying the current deck contents, will use json_encode to display the contents of the four sessions seperately in this order: „deck“, „main“, „side“ and „maybe“
7. Cards can be added or removed from their respective containers via input parameters, only allowing „No“ and „Yes“ for either the ‚sideboard‘ or the ‚maybeboard’ parameters

8. Deleting a decklist, will also delete all database entries with the respective deck_id value
9. Updating deck contents will first delete all entries in the database table cards_decklists and then refill them with the current contents of the the „main“, „side“ and „maybe“ sessions

<A comprehensive list of all User Stories can be found in the Kanban Board as well as in the deckbuilder_archive_xampp_PHP_backend_US.pdf file>