Technical Decisions - WIP **2**
Potential changes - to be discussed 2
Framework library 2
Database Decision 2
2.1 Database type: Postgresql - Refactoring MS for db options - Cross-project issue 2
2.2 Existing JsonCustomType vs JSON type - Refactoring MS to json type? - Cross-project issue 3
2.3. Separate schema vs single schema - BIT schema injection on MS - Cross-projects issue 4
2.4. Numeric vs Float for date type - Refactoring MS for Numeric? - Cross-project issue 5
Background Enum and Usages 6
REST API Endpoints error codes conflict MS-BIT API - Cross-projects issue 6
Access token is needed for GET /users on MS API but for BIT API it's not needed - Cross-projects issue 7
Refactoring MS for Dictionary object to DefaultDict - Cross-projects issue 8
Refactoring MS for list empty check (if is not … or if len….) - Cross-projects issue 8
Refactoring MS for increment function (a = a + 1 => a += 1) - Cross-project issue 8
Refactoring MS for string format `%s` to `f-string` - Cross-project issue 8
Add BIT schema in MS code base - Cross-projects issue 9
Mocking api calls on both backend and frontend 9
Refactoring MS from tuple to dictionary or namedtuple - Cross-project-issue. 9
Delete /user functionality - Soft delete vs hard delete ?? 11
Enum logic - keep it on one place 12
AUTH_COOKIE user_id bug issue 12
Refactor POST/PUT /user/additional_info and /user/personal_background 13
References 14

Technical Decisions - WIP

Potential changes - to be discussed

1. Framework library

What?
Flask-RESTPlus is no longer maintained. Flask-RESTx is the extension of the Flask-RESTPlus that continues to maintain the project.

Why is this a problem?
Could have an impact on the Mentorship System that is currently using Flask-RESTPlus

Suggested solution?
Refactor to Flask-RESTx. MS will do this gradually (not part of GSoC), but BIT will start fresh so use Flask-RESTx from the beginning.

Decision: BIT is set using Flask-RESTx

2. Database Decision

2.1 Database type: Postgresql - Refactoring MS for db options - Cross-project issue

Reasons:

- MySQL:
  Pros:
  - Support JSON

  Cons:
  - No support for JSONB - means - no direct indexing ability. JSON_EXTRACT alternative support indexing but more complex
  - Multiple schemas are treated as multiple databases
- SQLite:

Pros:
    - In-memory database, no prior setup needed
    - Support JSON

Cons:
    - Not allowing multiple schema in one database
    - No support for JSONB - means no indexing ability. JSON1 is available as alternative

- Postgresql:

Pros:
    - Support JSON and JSONB - get the advantage of indexing ability (with sqlalchemy.dialect.postgresql)
    - Can have multiple schemas
    - Both AWS and Heroku have postgresql database support.
- Cons:
    - If we want to go with JSONB, We can only have one option of database type == postgresql,
    - No in-memory sqlite option
    - Slightly more complex in initial setup on the local machine. But can be overcome with step-by-step instructions

Decision: BIT is currently using only postgresql.

2.2 Existing JsonCustomType vs JSON type - Refactoring MS to json type? - Cross-project issue

- JsonCustomType:

Pros:
    - Already being used in MentorshipRelationModel
- Cons:
    - Flask-migrate version script must be changed because sqlalchemy doesn't recognise the type
    - Even after changing the flask-migrate version script, it'll still be inserted into the database as TEXT type (tested on postgresql)
- JSON type:

Pros:
    - SQLAlchemy JSON type is native type (support postgresql, MySQL >= v5.7, sqlite >= v3.9
    - JSON native type won't need json.dump/load?

    - JSON native can be extended to JSONB (on postgresql only - supports direct indexing)
    - JSON native can be extended to JSON1 (on sqlite v3.9 only - supports indirect indexing)
    - Indexing with MySQL on JSON can be done with JSON_EXTRACT - indirect indexing

- Cons:
    - If chosen, existing mentorship_relation related code must be changed for consistency
    - Different database types need language specific support to json with indexing ability (jsonb: postgres, json1:sqlite, or json_extract:mysql) -> increase complexity if we want to keep all 3 database types available
    - Json_extract, json1 == complicated, indirect, no difference than writing own JsonCustomType

Decision: BIT is using JSON type on the newly added BIT tables.

2.3. Separate schema vs single schema - BIT schema injection on MS - Cross-projects issue

- Separate:
  Pros:
    - So BridgeInTech can be an independent addition to Mentorship-System
    - Less disruption? Not sure if this is the case.
  Cons:
    - It's impossible to keep the two totally separate because public (existing MS) schema will still need some relationship reference to bitschema on Foreign Keys regardless if they are kept as separate or not.
    - Only targeting postgresql database. Multiple schemas in one database is not possible in sqlite or mysql. Mysql will treat them as separate databases
    - Much complex initial setup because flask-migration files need to be adjusted.
- Single:
  Pros:
    - Simplifying structure and migration process
    - Easier to sync BIT development with MS development
    - Less complex to merge and migrate existing MS tables to new BIT tables because ne changes needed to auto-generated flask-migrate env.py
    - Can be applied to Mysql, sqlite and postgresql database

Cons:

- Challenge in merging existing MS to BIT related tables, but this shouldn't be a problem because auto-generated flask-migration script can be used directly to migrate database, no modification needed. The only drawback will be potential lost of existing data if we go with this option.

On both options (separate or single schema), the MS tables need to be modified to reflect relationships to BIT tables.

Decision: BIT is currently using 2 separate schemas: 'bistschema' for BIT and 'public' for MS.

2.4. Numeric vs Float for date type - Refactoring MS for Numeric? - Cross-project issue

- Numeric
  Pros:
  - Exact numeric data type
  - Better precision
  - Support '=' operator (we use this a lot in our testing) for comparison

  Cons:
  - Slower calculation than using Float
- Float
- Pros:
  - Approximate numeric data type
  - Faster calculation than using Numeric
- Cons:
  - Less precise
  - Should be avoided on '=' operator
  - When using with sqlalchemy + flask-migration, data stored in database vary in decimal digits (e.g sometimes 4 digits, sometimes 2 digits, inconsistent). See discussion on PR#668 Mentorship System backend, testing failed when datetime.utcnow() is kept as Float but passed when refactored to Numeric with precision (16,6).
  - Depends on the floating-point hardware on OS as well (windows powershell will behave differently to float data type) ~ potential caused on issue faced by Foong here

Decision: BIT is currently using Numeric for datetime.utcnow().

3. Background Enum and Usages

Array of Enum reference:
https://stackoverflow.com/questions/41258376/array-of-enum-in-postgres-with-sqlalchemy

Background Enum is used in the following tables (same keys, but different values):

- Personal_Background
  - Each user must choose one answer for each category. The default answer is set to DECLINED ~ 'Prefer not to say'.
  - On the Personal Background Form, the option NOT_APPLICABLE is disabled.
- Target_Candidate
  - A company can select more than one answer in a category and can have more than one category as their target candidate.
  - On the Create Program Form, the option DECLINED is disabled.
- Demographic_Data
  - A company must fill a Demographic Data Form by putting a percentage on each value per each background enum categories to get an overview of their employee demographic data.

Decision:

- **For simplicity, currently the Target_Candidate is set to a JSONB on BIT table not Array of Enum**
- **Demographic_Data is going to be put as an icebox. It is going to be revisited post-GSoC as part of future features.**

4. REST API Endpoints error codes conflict MS-BIT API - Cross-projects issue

What?

- Some of the error codes are misrepresented on MS API and need to be modified so BIT can represent them correctly

Why is this a problem?

- They need to be modified on MS side so might block the progress on BIT

Suggested solution?

- Option 1. Open small issues per API requests on mentorship-backend repository so community can take up the issues and work on it gradually
    - Pros:
        - Fix issues at the heart of the problem
        - Maintain **one source of truth** principle
    - Cons:
        - Delay progress on BIT side
- Option 2. Ignore MS error code and just represent them correctly on BIT response and move on.
    - Pros:
        - No delay on BIT side
    - Cons:
        - More than **one source of truth**
        - Inconsistent responses between the coupled MS-BIT system

Decision taken for now?

- Open small issues on MS side to be taken up by AnitaB community
- Isabel is to discuss with May how to handle if any similar future MS-BIT blockers/conflicts:
    - should it be treated as an **allowed deviation** from GSoC schedule or as **technical debt?**
    - should we have a **designated OS team** to be involved and work on this type of issue (**AnitaB's cross-projects issues**)?
        - Response: Not applicable - no need for designated OS team to work on BIT cross-project. If it's cross-projects, then anyone can work on it.

5. Access token is needed for GET /users on MS API but for BIT API it's not needed - Cross-projects issue

What?

- Currently MS API requires user to login to be able to get list of users by sending GET /users request

Why is this a problem?

- As part of BIT Required feature for MVP, non-login and public users should be able to view (get) list of users by going to **Members** page from the Navbar

Suggested solution?

- Option 1: Do work-around:
        - Set a generic user A

        - Write code (scheduler) so this user A requests an access token (using GET /login?) per life cycle set in MS (one week for access token and 4 weeks for refresh token)
        - Use this token on the background process of GET /users to intercept request by anyone accessing **Members** page

    - Pros:
        - Could work
    - Cons:
        - Add complexity and time delay
        - Add overhead (background scheduler)
        - Increase security risk??
- Option 2: Remove JWTAuthentication requirement from MS GET /users and only applied authentication on GET /users/{user_id} (when a user wants to view details of the other users.
    - Pros:
        - Maya is personally see no harm on doing so
    - Cons:
        - Must be handled on MS side for the community to take it up.
        - Might cause some delay on GSoC schedule (if no one wants to work on it or take their time while working on it - unless we have OS Team assigned to tackle this type of issue - **AnitaB's cross-projects issues**)

Decision taken for now?

- Not yet explored as it will come up when working on **Homepage** backend/frontend features

6. Refactoring MS for Dictionary object to DefaultDict - Cross-projects issue
7. Refactoring MS for list empty check (if is not … or if len….) - Cross-projects issue
8. Refactoring MS for increment function (a = a + 1 => a += 1) - Cross-project issue
9. Refactoring MS for string format `%s` to `f-string` - Cross-project issue

10. Add BIT schema in MS code base - Cross-projects issue

Update: For MS and BIT integration, MS no longer needs to have BIT schema in its code base. This was trialled on Maya's MS fork branch. The only few adjustments made on MS to accommodate BIT are:

- renaming `organization` field on UserModel to `current_organization` to avoid ambiguity to the Organization table of BIT schema.
- refactor the use of Float to Numeric with prescribed precision since the tests initially failed on these floating points (see discussion point 2.4)

11. Mocking api calls on both backend and frontend

Why does BIT need to mock its api calls for testing?
Because, for MS related data (user and mentorship relation), BIT needs to send http requests to MS REST API and not directly query-ing the database. If the development environment is `local` (both MS and BIT servers are run locally) this is not going to be an issue. The issue comes when pushing a commit to the main repository (to be merged with the develop branch). The tests on travis will fail since the actual call is not made during travis build.

12. Refactoring MS from tuple to dictionary or namedtuple - Cross-project-issue.

Why is refactoring needed?
Index referencing to a tuple object is not as straightforward to new developers or contributors coming to the program in comparison to namedtuple or a dictionary.

1st Problem: Cross-project issue which might become a blocker if not dealt with (or refactored) asap.

Temporary solution:
Using python decorator to convert MS http responses as soon as they are received by BIT while MS deciding/refactoring its direction moving forward.

2nd Problem: Deciding between NamedTuple vs Dictionary object options.
Discussions:
=> **NamedTuple:**
Pros:

- Similar structure to tuple so refactoring won't be too much different

Cons:

- Less adaptable to changes. It is tightly coupled to source (in this case, MS). If MS modifies the tuple elements (add/remove) then BIT code will not work

=> **Dictionary:**

Pros:

- Loosely coupled to source. If MS adds an element to the dictionary object, BIT can still get the original elements, it'll just miss the added elements. However, in the case of deletion of source elements, same case applies as namedtuple object (aka, will not work)

Cons:

- More difficult to work with when testing mocking api which is done in BIT when testing http requests related to MS API endpoints. This is because the responses http would be in a tuple-like object received from MS side but throughout the code, the object to be worked on is dictionary, if it is chosen as the preferred object.

Decision:
For now, **namedtuple** will be used on the BIT side with python decorator as it is easier to work on with mocking api calls.

Example:
> On validations/user.py password validation
=> Using tuple:

```
if not is_valid[0]:
    return is_valid[1]
```

=> using namedtuple:

```
if not is_valid.get("is_valid"):
    return is_valid.get("message")
```

> On HTTP response
=> Using tuple:

```
if response[1] != HTTPStatus.CREATED:
    return response
```

=> using namedtuple:

```
@http_response_namedtuple_converter
def http_response_checker(result):
    # TO DO: REMOVE ALL IF CONDITIONS ONCE ALL BIT-MS HTTP ERROR ISSUES ON MS ARE FIXED
    # if result.status_code == HTTPStatus.OK:
    #     result = http_ok_status_checker(result)
    # # if result.status_code == HTTPStatus.BAD_REQUEST:
    #     result = http_bad_request_status_checker(result)
    # if result.status_code == HTTPStatus.NOT_FOUND:
    #     result = http_not_found_status_checker(result)
    # if result.status_code == json.dumps(HTTPStatus.INTERNAL_SERVER_ERROR) and not AUTH_COOKIE:
    #     # if not AUTH_COOKIE:
    #     return messages.TOKEN_IS_INVALID, HTTPStatus.UNAUTHORIZED
    return result
```

In the BIT code example above, HTTP responses from MS are masked as temporary solutions to point 4 issue above while MS responses are gradually being refactored to match HTTP responses on BIT.

13. Delete /user functionality - Soft delete vs hard delete ??

What?

At one point, a user might want to remove themself (their account) from BridgeInTech, or an Admin could also do this for archiving purposes. A decision need to be made as to whether or not the user's record will be kept in the record but any reference to it will be removed (soft delete) or whether the record is to be removed totally from the system with no way to retrieve it back (hard delete). Both come with pros and cons.

Why is this a problem?

Could pose as legal risk, unnecessary overhead, and increase complexity that would delay development progress

Hard delete:

Pros:

- No legal risk would come from keeping user's data without their consent

Cons:

- More complex. Must take extra care when removing the relationships between user's record to the other tables

Soft delete:

Pros:

- Quicker, less complex since the record still there only the connection to other tables are removed
- Provide option for user to retrieve their old records if they ever regret their decision to remove/delete their account

Cons:

- If the user is not informed properly that `delete` does not mean removing their record from the system they might bring legal action in the future since they don't give their consent
- The `deleted` user's record must be protected as much as any existing user's account therefore BIT still liable on any security breach

14. Enum logic - keep it on one place

What?
Need to keep all enum logic in one place, either on backend or frontend, but not both.

Why?
- Keeping it in one place helps make the project workflow cleaner

Important points:
- PostgreSQL keeps enum as Key instead of their Value, but users view them in their values. Therefore, conversion logic is needed to convert from enum chosen as user input to key (how it is recorded in the database).
- Enums are listed as DB Types and created along with the initial DB creation.

Backend vs Frontend?

Backend:
Pros:
- Where most of the logics are
- Lighter on client-side == faster/better in performance (UX - user experience)

Cons:
- Adds to backend overhead

Frontend:
Pros:
- Less overhead to backend

Cons:
- Might affect performance because heavier processing on frontend
- Frontend best to be kept away from business logic. So, strictly for viewing. The lesser business logic is kept on the frontend, the lesser risk from cross-site-scripting (XSS) attack.

Decision:
Keep enum logic on backend

15. AUTH_COOKIE user_id bug issue

What?

As per reported on issue #94 backend, AUTH_COOKIE user_id from the previous logged in user persists and can be used by the next logged in user to get previous logged in user information.

Why is this a problem?

1. Users should only be allowed to get their own personal details, additional information or personal background from GET /user/xxx api endpoints. In BIT, Users authentication is handled through MS REST API where JWT token is created, therefore, there's **no automatic mapping user_id to jwt token**.
2. Currently BIT is keeping this token to user_id mapping through AUTH_COOKIE user_id where the user_id is retrieved using GET /user/personal_details after the user logged in. The problem is, if the next user login, the existing user AUTH_COOKIE user_id needs to be removed otherwise **the next login user will have access to this previous user id**.
3. However, removing existing AUTH_COOKIE user_id each time on login will only solve the issue when the development is done in a local environment because only one developer has access to the application. This will not work when the development is done on a remote server where more than one developer is accessing the application at a time. Same issue will occur if this is carried to production.

Potential solution:
On initial thought, BIT could create its own JWT token and use it strictly when making calls to BIT REST API. BIT would still need to keep MS JWT token for the http requests related to MS REST API.
This option is to be discussed with mentors and further explored.

Update: Currently the project is implementing the approach of saving a new User object in AUTH_COOKIE every time a new user send POST /login request. However, this is not ideal for multiple users environment when the application is deployed to a remote server. Read more about the issue on Spike issue #109 backend and medium blog post

16. Refactor POST/PUT /user/additional_info and /user/personal_background

What?
The Frontend UI at the moment is implementing one

References

- Celko, J. (2015). *Numeric Data Type - an overview | ScienceDirect Topics*. [online] www.sciencedirect.com. Available at: https://www.sciencedirect.com/topics/computer-science/numeric-data-type [Accessed 7 May 2020].
- Iman (2017). *Difference between numeric, float and decimal in SQL Server*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/1056323/difference-between-numeric-float-and-decimal-in-sql-server [Accessed 7 May 2020].
- SQLAlchemy (2020). *MySQL — SQLAlchemy 1.3 Documentation*. [online] docs.sqlalchemy.org. Available at: https://docs.sqlalchemy.org/en/13/dialects/mysql.html#sqlalchemy.dialects.mysql.JSON [Accessed 7 May 2020].
- SQLite (n.d.). *The JSON1 Extension*. [online] www.sqlite.org. Available at: https://www.sqlite.org/json1.html [Accessed 7 May 2020].
- Stack Overflow. (2017). *mysql - Differences between Database and Schema using different databases?* [online] Available at: https://stackoverflow.com/questions/1869522/differences-between-database-and-schema-using-different-databases [Accessed 7 May 2020].
- Stack Overflow. (2015). *Does SQLite support multiple schematas within the same database?* [online] Available at: https://stackoverflow.com/questions/33960762/does-sqlite-support-multiple-schematas-within-the-same-database [Accessed 7 May 2020].