myTECHCareers

ASP.NET API design notes

* 1. **CohortsController**.   
     Accessible to Staff only.
     1. GET - /list - Return all cohorts as a list (DTO)
     2. GET - /get - Return a cohort with a specific CohortID (DTO)
     3. PUT - /add - Add a cohort to the database (DTO, BLL)
     4. PUT - /modify - Modify cohort with a specific CohortID. Parameters passed in Body (DTO, BLL)
     5. DELETE - /delete - Remove a cohort with a specific CohortID [SuperAdmin only]
        1. Output: Status Code

* 1. **StudentsController**.   
     Accessible to Staff only.
     1. GET - /list - Return all students as a list
        1. No input parameters
        2. Output DTO: UserDTO (no virtual props)
     2. GET - /details - Return all students as a list with Cohorts and User objects
        1. No input parameters
        2. Output DTO: UserDetailDTO (with virtual props)
     3. GET - /get - Return specific user information based on StudentID
        1. Input: int id
        2. Output DTO: UserDTO
     4. PUT - /add - Add a student to the database (DTO, BLL)
        1. Inputs: string firstname, string lastname, string email, bool active, int CohortID, string BearTracksID
        2. BLL: firstname not null, lastname not null
        3. BLL: firstname proper case, lastname proper case, email valid email/and not null (regex)
        4. BLL: email conforms to the domain restriction as specified in appsettings.json
        5. BLL: email not present in database and the student record exists (will cater for presence of user and no presence of student record)
        6. BLL: CohortID must correspond to an active Cohort [SuperAdmin override]
        7. Output DTO: UserDetailDTO
     5. PUT - /active - Activate/deactivate the user (prevent from logging in) (DTO)
        1. Inputs: int id, bool state
        2. Output DTO: UserDTO
     6. PUT - /modify - Modify student record
        1. Inputs: Uses an object passed as body in the message with the following properties { int UserID, int CohortID, string Firstname, string Lastname, string Email, bool Active, string BearTracksID }. This is stored in an object StudentDetailDTO.
        2. BLL: checks are the same as for /add with exception of e-mail not present in database check. Checks if the new email is not present in the database and does not belong to current user.
        3. BLL: all parameters, even those that are not changed need to be passed back to the API. Potential for improvement by using only those parameters that are updated.
        4. Output DTO: StudentDetailDTO (input params)

* 1. **Attendance**Accessible to anyone for reading their own records. Write operations are Staff only.
     1. GET: /get - Return attendance records matching criteria
        1. Inputs: int studentID, int attendanceStateID, DateTime startDate, DateTime endDate
        2. BLL: All the inputs are optional, so if any are missing defaults are applied
           1. startDate = -7 days from DateTime.Today
           2. endDate = DateTime.Today
           3. studentID = currentStudentID or 0 (if staff) - returns all Student records
           4. If Student is logged in, StudentID in the request must match their StudentID
           5. Start date can't be after the end date
           6. AttendanceStateID specified in the request must be valid
        3. Output DTO: AttendanceDTO
     2. PUT: /add - Adds an attendance record
        1. Inputs: AttendanceDTO record in body of the request, consisting of int StudentID, int StaffID, int AttendanceStateID, DateTime Date and string Comment
        2. BLL: All the inputs are mandatory:
           1. StudentID must exist in the table of Students
           2. AttendanceStateID must exist in the table AttendanceStates
           3. StaffID specified in the request must exist
           4. StaffID provided must match the logged in user (user submitting the request)
           5. Duplicate records must not be made in the database
           6. New record must be no more than 3 days old [Super Admin override]
           7. Student must be active [Super Admin override]
        3. BLL: All parameters are passed back to the API without modification as AttendanceDTO
        4. Output DTO: AttendanceDTO
     3. PUT: /modify - Modify Attendance record
        1. Inputs: AttendanceModDTO record in the body of the request, consisting of int RecordID, int AttendanceStateID and string Comment
        2. BLL: Checks are simplified compared to Add:
           1. Specified RecordID must exist
           2. Specified AttendanceStateID must exist
           3. Comment field is not manipulated in any way.
        3. BLL: Returns the parameters back to API target method without modification as AttendanceModDTO
        4. Only the AttendanceStateID and Comment can be modified for each record
        5. Output DTO: AttendanceDTO
     4. DELETE: /delete - Delete Attendance record [Super Admin only]
        1. Inputs: RecordID
        2. No BLL. Record must exist prior to deletion.
        3. Returns the removed record.
        4. Output DTO: AttendanceDTO

* 1. **AttendanceStates**Accessible to anyone for reading attendance states
     1. GET: /list - retrieves all the records from the attendance\_states table
     2. PUT: /add - adds an AttendanceState record into the database ("practice", "assignment", "milestone"…)
        1. Input: string description as query parameter
        2. BLL: Checks implemented:
           1. Description parameter must not be empty
           2. Description must be unique (no duplicates allowed in the table)
        3. Returns the new record with the result
        4. Output DTO: TaskTypeDTO
     3. PUT: /modify - modifies an existing TaskType record
        1. Input: TaskTypeDTO record in the body of the request, consisting of int TypeID and string Description
        2. BLL: Checks implemented:
           1. Specified TypeID must exist
           2. Description cannot be empty or left out
           3. Description must be unique (no repeat TaskTypes), excluding the one with the TypeID being modified
           4. Description is modified to start with a capital letter, with the rest of the letters in lowercase.
        3. TaskType must exist otherwise notfound error will be returned
        4. Returns the modified record
        5. Output DTO: TaskTypeDTO
     4. DELETE: /delete - deletes an existing TaskType record
        1. Inputs: TypeID
        2. No BLL: Record must exist prior to deletion
        3. Returns the removed record
        4. Output DTO: TaskTypeDTO

* 1. **TaskTypes**Accessible to anyone for reading task types.
     1. GET: /list - retrieves all the records from the TaskTypes table
     2. PUT: /add - adds a TaskType record into the database ("practice", "assignment", "milestone"…)
        1. Input: string description as query parameter
        2. BLL: Checks implemented:
           1. Description parameter must not be empty
           2. Description must be unique (no duplicates allowed in the table)
        3. Returns the new record with the result
        4. Output DTO: TaskTypeDTO
     3. PUT: /modify - modifies an existing TaskType record
        1. Input: TaskTypeDTO record in the body of the request, consisting of int TypeID and string Description
        2. BLL: Checks implemented:
           1. Specified TypeID must exist
           2. Description cannot be empty or left out
           3. Description must be unique (no repeat TaskTypes), excluding the one with the TypeID being modified
           4. Description is modified to start with a capital letter, with the rest of the letters in lowercase.
        3. TaskType must exist otherwise notfound error will be returned
        4. Returns the modified record
        5. Output DTO: TaskTypeDTO
     4. DELETE: /delete - deletes an existing TaskType record
        1. Inputs: TypeID
        2. No BLL: Record must exist prior to deletion
        3. Returns the removed record
        4. Output DTO: TaskTypeDTO

* 1. **Tasks**Table containing task definitions used for time tracking

a. GET: /get - retrieves all task records based on input parameters (or default values where applicable)

* + - 1. Inputs: int TaskID, int cohortID, int unitID, DateTime startDate, DateTime endDate
      2. BLL: All the inputs are optional, so if any are missing defaults are applied
         1. CohortID = current user's Cohort, or 0 if Staff / Super Admin - returns all records
         2. startDate = -14 days from DateTime.Today
         3. endDate = DateTime.Today + 7 days
         4. If Student is logged in, CohortID in the request must match their CohortID
         5. Start date can't be after the end date
         6. TypeID, CohortID, TaskID, UnitID specified in the request must be valid
         7. BLL returns a TaskDTO object back to the target
      3. Returns the record(s) based on criteria
      4. Output DTO: TaskDTO
    1. PUT: /add - adds a Task record into the database (specific information pertaining to the task)
       1. Input: TaskDTO record in the body of the request, consisting of int int UnitID, int CohortID, int TypeID, string Title, DateTime StartDate, DateTime EndDate, string DocURL
       2. BLL: Checks implemented via AddTaskBLL method:
          1. Specified UnitID, CohortID and TypeID must exist in the respective tables.
          2. Specified CohortID must be active.
          3. StartDate and EndDate must be in the future (Date based, excl time component for this determination
          4. Title must not be empty
          5. DocURL, if present must start with http:// or https://
       3. Returns the newly added record with the appropriate RecordID
       4. Output DTO: TaskDTO

PUT: /modify - modifies a Task record in the database

* + 1. Input: TaskDTO record in the body of the request, consisting of int TaskID, int UnitID, int CohortID, int TypeID, string Title, DateTime StartDate, DateTime EndDate, string DocURL
    2. BLL: Checks implemented via the ModifyTaskBLL method:
       1. Specified UnitID, CohortID and TypeID must exist in the respective tables.
       2. Specified CohortID must be active.
       3. Title must not be empty
       4. DocURL, if present must start with http:// or https://
    3. Task must exist otherwise notfound error will be returned
    4. Output DTO: TaskDTO
    5. DELETE: /delete - deletes a Task record in the database
    6. Input: TaskID
    7. No BLL: Record must exist prior to deletion
    8. Returns the removed record
    9. Output DTO: TaskDTO

* 1. **Timesheets**Table containing timesheet records for every student, per Task
     1. GET: /get - retrieves timesheet records based on input parameters (or default values where applicable)
        1. Inputs: int recordID, int studentID, int assignmentID, int cohortID, DateTime startDate, DateTime endDate
        2. BLL: Checks implemented via the GetTimesheetRecordBLL:
           1. If specified, recordID, studentID, unitID, typeID, assignmentID and cohortID must be valid
           2. If user is not a member of Staff or Super Admin, then studentID must match the current user (default behavior)
           3. If specified cohortID must be valid. If user is not staff/Super Admin it is ignored
           4. startDate must be earlier than endDate. If not specified they default to current date
        3. Returns List of TimesheetDTO objects conforming to criteria
        4. Output DTO: TimesheetDTO
     2. PUT: /add - Create a timesheet record
        1. Input: TimesheetDTO record in the body of the request, consisting of int RecordID (ignored for add operation), int StudentID, int AssignmentID, DateTime Date, decimal TimeAllocation
        2. BLL: Checks implemented via AddTimesheetRecordBLL method:
           1. Specified StudentID and AssignmentID must exist
           2. The date of the record can only be up to 7 days in the past
           3. The date cannot be in the future
           4. TimeAllocation has to be a positive number, smaller than 18
           5. TimeAllocation has to be rounded to nearest 15 min
           6. StudentID from the authentication has to match the StudentID in the request record
           7. Date and AssignmentID allocations must be unique for a StudentID
        3. Returns the created record
        4. Output DTO: TimesheetDTO
     3. PUT: /modify - Modify a timesheet record
        1. Input: TimesheetDTO record in the body of the request, consisting of int RecordID, int StudentID, int AssignmentID, DateTime Date, decimal TimeAllocation
        2. BLL: Checks implemented via ModifyTimesheetRecordBLL method:
           1. StudentID of the logged in user must match the StudentID in the record
           2. AssignmentID must exist
           3. Date cannot be more than 7 days from present (date of the record isn't changed regardless)
           4. TimeAllocation must be positive and smaller than 18
           5. TimeAllocation has to be rounded to nearest 15 min

* 1. **AuthController**Manages confirmation of authentication records between client and server
     1. GET: /login - receives JWT token from the frontend. If authenticated, returns back user parameters
        1. Input: Request Header, Authentication, Bearer JWT token
        2. BLL: none
        3. Returns an anonymous object containing user's full name, e-mail address, user rights and cohortID.
        4. If authentication fails, returns 401 unauthorized response.