TECHCareers capstone project proposal document - Student Record Management System

## Problem statement

Effective time management is an essential skill for software developers. To entrench this behavior, TECHCareers developers are encouraged to keep an accurate record of the time spent on studying, practice problems and assignments.

TECHCareers staff team has developed a spreadsheet system which has proven difficult to manage considering the large number of students, assignments and number of cohorts enrolled in the program.

An opportunity exists to use the timesheet data to gain insights into teaching effectiveness and students’ understanding of the material.

## Proposed solution

It is proposed that the existing spreadsheet system be replaced with a combination of a database backend and a web-based frontend:

* ASP.NET Web API and Microsoft SQL Server (Express) will be used as the backend solution. Internally, Entity Framework will be used to create the DB-Object model, which makes the solution relatively easy to adapt to alternative database solutions should it be necessary.
* Frontend solution will be entirely based on the React library, with Bootstrap CSS library used for application theming. These have been chosen due to popularity, standardisation, and codebase maturity. Frontend solution will be designed as mobile-first, with adaptation for tablet and desktop users.

Key features of this approach are:

* Separating the API from the frontend allows for future design/front end functionality changes without impacting the backend business logic or compromising data integrity.
* For example, this creates an opportunity for creation of platform-native clients. From code maintenance perspective, this approach also has distinct advantages due to the separation of concerns and ability to scale up the app should the number of users require it.
* React/Single Page Applications allow for creating responsive end-user experiences and minimize network traffic requirements. This improves user experience for mobile platforms.
* There is a need to authenticate each backend request, as the frontend and backend are independent from each other functionally, which increases the processing time for each request. However, this is necessary to achieve the

Details of the proposal are discussed in Section 3 of this document.

## Scope of the project

Minimum viable product: Service that accurately and reliably captures student timesheets, that is easy to maintain or expand in the future.

### User management

The application is envisaged to have three categories of users:

* Students
* Staff
* Super Admin (typically 1 person with additional rights over the Staff members)

The application uses two layers of data protection. The first layer of protection is achieved through the authorization layer, intended to prevent unauthorised access to the system by external parties. This portion of the system is exclusively reliant on use of Google OpenID/OAuth2 authentication tokens. The decision not to integrate internal password management has been based on:

* Google authentication is regularly audited and updated to ensure that the solution is secure. This also means that the application will require minimum to no updates to authentication code base to maintain security levels.
* Significant reduction in code complexity by removing the need for password management and associated code (password recovery, email verification, hashing, password salt, etc.) within the app.
* Reduces the effort needed by staff to create and maintain student records. All students and staff have University of Alberta e-mail accounts, which authenticate through Google.

The second layer of protection is achieved through category/role-based authorization. Students for example, would only have access rights to edit/create their own data within boundaries of the database schema. Staff members on the other hand would have ability to create, delete and edit user records, create reports, etc. (more detail is provided in respective sections).

User data is stored in 4 separate tables:

* Users table:
  + User ID – unique ID specific to each user
  + First name
  + Last name
  + Email address – subject to being a valid email address and unique to the table
  + Active flag, to allow the user to be deactivated while remaining on database.
* Staff table:
  + Staff ID – unique ID for each staff member
  + UserID – Foreign Key reference to the UserID field in the Users table
  + SuperUser – bool (bit) value indicating if the user has Super Admin rights
* Cohorts table, defining each class of students:
  + Cohort ID – unique ID specific to the cohort
  + Name – descriptive name for the cohort
  + Start date – date cohort starts
  + End date – date cohort ends (graduates)
* Students table, intended to bridge cohort and user information:
  + Student ID – unique ID specific to each student
  + User ID – Foreign Key reference to User ID field from the Users table
  + Cohort ID – Foreign Key reference to the Cohort ID from the Cohorts table
  + BearTracks ID/ Student Number – contains registration information specific to UofA.

Required functionality is as follows:

* Ability to create and edit User/Student records by staff users (table Users and Students). Adding data to both tables is done through a unified interface (i.e. a single form).
* Ability to create and edit cohort records by staff users (table Cohorts)
* Ability to create, delete, and edit staff records by Super Admin (table Staff)
* Ability to assign individual staff members to Super Admin status.
* No complete Student record deletion is allowed except by Super Admin.
* All account management actions are to be appropriately logged.
* Adding students/users in bulk should be possible through the Web interface in CSV format

Additional scope:

* Ability to bulk add users using a CSV file

Authorization workflow will be as follows:

* No passwords will be stored in the DB and an external authentication provider (Google OpenID Connect) will be used.
  + Authentication is initiated through a React Google Auth library.
  + A valid authentication token is received from Google.
  + A call is made to the application’s API to authenticate the user internally. In this step, user’s email address together with the Google authentication token are passed to the API.
* Backed authentication provider uses a .NET Google Auth library to validate the authentication token.
* Valid authentication check will comprise of:
  + Received authentication token being valid
  + Account e-mail will correspond to an e-mail address in the Users table
  + User will have an Active bit set to true in the Users table
* On login and successful receipt of an auth token from Google, a session token is assigned to the user which is valid for as long as the google auth token is valid (same rules as for Google services).
* User is automatically logged out after 1h. Google auth token remains valid for 24h or until the user logs out of Google services (between 1h – 24h user will be prompted to authenticate by Google, but no password will be required). Backend API will regenerate the internal session token, however.
* Internal authentication will be managed via the JWT (JSON Web Token) Auth Provider.
* Subsequent interaction between the frontend app and the API occurs using the preauthorized JWT token.
* Implementation guides:
  + <https://developers.google.com/identity/protocols/oauth2/openid-connect>
  + ASP.Net Core 3 and React, Carl Rippon, Pakt Publishing 2019 (ISBN: 9781789950229)

### Timesheets and tasks

One of the difficulties with maintaining the existing spreadsheet based timekeeping system is that the changes in assignment positions/descriptions would corrupt time calculations and other formulas in the spreadsheet. The proposed solution avoids this scenario by providing a series of data consistency checks at each layer of the solution – frontend, API, and database itself.

In addition to typical activities such as classroom instruction, self-study, and rest, most of the students’ time is devoted to solving exercises, assignments, group assignments, etc. To make the time capturing process as intuitive as possible, a list of tasks will be offered to students. The list offered to the students would follow the appropriate time context, i.e. they would only see assignments which are due within a short time window (7 days) of the selected date.

Another purpose behind systematically naming and collecting task data is that it allows for simpler analytics and comparison between tasks, students, and cohorts. Additional criteria such as the assignment topic allow for further grouping of reported data.

The following business logic is foreseen:

* Only staff and Super Admin can create or modify task records.
* Task deletion is only allowed if there are no timesheet records associated with the specific task.
* To simplify task creation, an option will exist to copy all parameters from a task created earlier (with edits as appropriate). This creates an opportunity to compare cohorts for the same task (based on title, as each cohort gets a new unique Task ID assigned to them)
* A new task cannot be created more than 3 days after its intended start date.
* Students will have the ability to browse/see all the tasks assigned to their cohort up to and including the last task for which the start date has passed (i.e. they will not be able to see tasks with the start date set in the future).
* Tasks are associated with learning units, referenced by UnitID. This can be as granular as deemed necessary. As Unit ID will be used for the purpose of reporting and data analysis, its use should be consistent. Each task must have an unit associated with it.
* Tasks also have an associated type (practice, assignment, milestone, etc.), defined in the TaskType table. This also includes recurring tasks such as classroom instruction, online instruction, etc.
* Staff have rights to add new and edit existing units (titles/metadata). Deletion is restricted to cases where the topic was not previously associated with an task to maintain database consistency.
* List of Units is maintained in a separate table in the database.

The Units table has the following structure:

* UnitID – unique to each unit
* Description

The TaskType table has the following structure:

* TypeID – unique to each type of task
* Description

The Tasks table has the following structure:

* TaskID – unique to each task
* UnitID – foreign key reference to the Unit ID in Units table
* CohortID – foreign key reference to Cohort ID in the Cohorts table
* TypeID – foreign key reference to the TypeID in TaskType table
* Title
* StartDate (incl time)
* EndDate (incl time)
* DocURL - Google Docs URL

The intent is to incorporate assignment grading reports into the system at a later stage.

### Timesheets

From students’ point of view, timesheets are the main point of interaction with the system. To make this interaction as simple as possible, the following feature set is envisaged:

* In mobile view, the student only sees a daily view of the timesheet, with the current date selected by default. Controls are available that easily let the student select an alternate date from a dropdown and/or change the date one day backward/forward as required.
* In tablet view and desktop view, the student sees a weekly timesheet by default, with corresponding view controls.
* Student picks an activity from a predefined dropdown list and assigns hours to specific categories. Mandatory activities are always shown on screen.
* Zero and non-zero values are accepted. Figures are rounded to the nearest 15min (0.25 of an hour). All figures must be numeric.
* All date entries are validated.
* The interface provides a daily total. Tablet and desktop editions also include weekly/monthly totals.
* Students will be able to export their timesheet data into CSV format.
* Students will only be able to edit timesheet records up to 1 week from date accessed.

Timesheet table structure is as follows:

* RecordID (unique to each record)
* StudentID (foreign key reference to the StudentID in the Students table)
* Date
* TaskID (foreign key reference to TaskID in Tasks table)
* TimeAllocation (hours)

Reports derived from the timesheet data will be discussed in Section 3.5 below.

### Attendance register

Attendance register is normally taken at the beginning of each teaching session (or as necessary). Creation and editing functionality is available to staff/admin users. Students can view reports on their attendance.

The attendance register interface will:

* Allow the staff member to tick off attendance for each student in any of the active cohorts (one cohort and student at a time).
* Staff member selects a cohort from a dropdown list, after which an appropriate interface for all the students is automatically generated (no need to pick individual students)
* In case of absence, the staff member will be able to select from a predefined list of reasons why a particular student was not in attendance.
* Free text field is also available to capture any other relevant details for the record (this is not visible to students)
* Existing attendance records can be edited by staff members (to be confirmed if this is desired behavior)

Attendance states in the predefined list are stored in a separate table, with the following columns:

* AttendanceStateID – Unique ID corresponding to each state
* Description – prepopulated

The attendance table contains the following records:

* RecordID – Unique ID corresponding to a single attendance record
* StudentID – Foreign key reference to StudentID column in the Students table
* AttendanceStateID – Foreign key reference to the AttendanceStateID in Attendance table
* StaffID – Foreign key reference to the UserID belonging to the staff member creating the attendance record
* Comment
* Date and time

### Reports

Reports, aside from the timesheet functionality are seen as the core of the entire system. Timesheet reports are important in correlating individual student progress and performance to the effort put into solving various assignments. It can also provide insight into the level of understanding/difficulty for a particular topic for a particular student or cohort.

Reports are defined by category (i.e. specific assignment or type of activity undertaken by the student, such as class learning, self-study, etc.), unit (i.e. C# OOP, Entity Framework, etc.) or type (practice, weekend assignment, milestone, …).

The proposed built-in reports are:

* Total amount of time spent per assignment/category for all students in a cohort. (i.e. “240 man-hours spent on C# OOP Practice 1; 300 man-hours spent on class lesson on EF”)
  + Comparison with earlier cohorts for same type of assignment (i.e. Cohort 1 spent 240 man-hours vs. 300 man-hours for Cohort 2 for the same practice)
* Total amount of time spent per assignment, per student in a cohort (i.e. a graph/table of hours spent per assignment per selected student).
* Time-series plot for the individual student, for all assignments (and/or categories) or the entire cohort
* Any additional reports after client review.

Administrative reports:

* Student’s time and attendance report, comprising of information from the Attendance register record.
* Cohort record, containing a list of all students with full names, email addresses, BearTracks/Student IDs, etc.

Features:

* All reports are presented in tabular/graphical format as appropriate.
* Ability to download raw data from the database in CSV format

### Student notices

The app should provide simple means of notice-board level communication with the students.

* Staff members would be able to set notices that would be displayed on student login
* Markdown editor will be provided to cater for simple, yet reasonably powerful formatting requirements.
* Student notices will be stored in a table Notices, which has the following structure:
  + NoticeID – unique ID for each notice
  + StaffID – Foreign key referencing the staff UserID from Users table (notice issuer)
  + Cohort – Foreign key referencing the CohortID from Cohorts table
  + ValidFrom – date from which the message should appear to students
  + Markdown– notice contents in markdown format, for easy editing
  + HTML – parsed HTML copy of the notice
* Staff members can modify messages if required.
* A function should exist to notify the students/staff via e-mail that the new notice has been issued (external mailto: link for the demonstration)
* Notice should be directly accessible via URL
* (Additional Scope) Students can be asked to acknowledge that they have read the message. This would require an additional table to track confirmations based on UserID and MessageID

### Randomizer

A simple feature that allows the staff member to randomize a list of students in a cohort for the purposes of morning stand-up or other activities. Function is not available to students.

* Results are presented in a list/tabular format
* Students that have participated in the activity can easily be struck off the list in any order, as required by the staff member.