Study Plus – Software Engineering Systems Report

Tempe High School

Contents

1. Identifying and defining

1.1. Define and analyse problem requirements

Problem context

High-school students sometimes fall trap to poor productivity, lack of management skills, and doom-scrolling issues with the advent of new technologies. To address these issues, I will deploy a simple but effective task reminder which stays consistent in layout, tracks progress that can be visualised, motivating users. Overall, these features will prevent the user from avoiding tasks and doom-scrolling.

Needs and opportunities

Need	Description
1. Task list	A simple to-do list which is interactable and customisable
2. Secure users	A profile where students can view other students or profiles stats
3. Progress tracker	A way to monitor complete tasks and study time, potentially with statistics or goal setting features

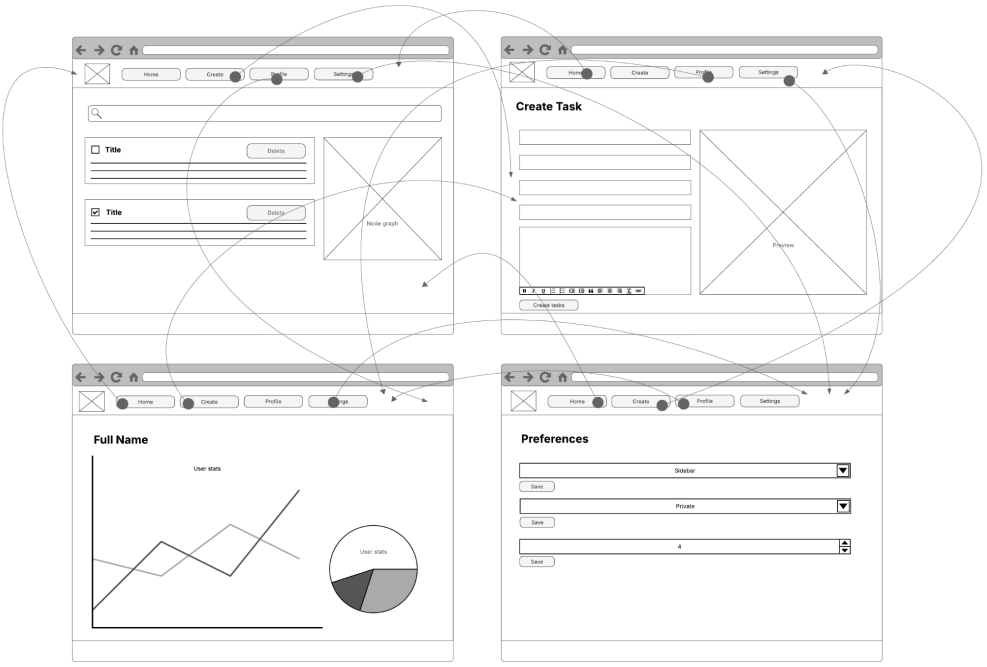
Boundaries

- ***Hardware:*** *The app will run smoothly on mobile devices, tablets, and PCs without any performance concerns.*
- ***Browser:*** *The app will run on popular browsers such as Google Chrome and Safari and work with outdated .*
- ***Operating systems:*** *The app should be cross-platform with operating systems such as iOS, Android and Windows. The app will not require cloud-based features.*
- ***Security concerns:*** *User data will be stored securely with necessary steps to encrypt logins and hash passwords.*

1.2. Tools to develop ideas and generate solutions

Application of appropriate software development tools

Idea	Client	Proposed Software Project
Productivity app	Me Dek Ethan	A customisable daily to-do list app with tags which Includes features such as alarms, AI driven categories, activity heatmap and tree. Additionally, sort tasks by difficulty.
Rhythm game	Me Ethan	A rhythm game where you click arrows to the beat of procedurally generated music.
Mouse tester	Me Dek Ethan	A hub for changing DPI, automatic sensitivity changer depending on game and mice specs.
Study music player	Me Dek Leo	A customisable calm music player which plays depending on mood/weather/day with AI.



1.3. Implementation method

For this project, I have chosen **phased** implementation as the most suitable approach due to the function of my app being overall quite simple. A phased implementation allows in mitigating the risks involved with updating, as well as supporting incremental improvements. By deploying the features such as user profiles, issues can be identified and addressed without affecting the entire system. Additionally, this supports smoother user adoption and provides flexibility for adjustments based on given feedback at each stage.

Other approaches in implementations such as **parallel** and **direct** cut-over, may be overly complex and resource intensive for the project. For relatively simple apps like mine, without much focus towards data integrity compared to the likes of banking apps, this redundancy is unnecessary and inefficient. Direct cut-overs involve an immediate switch from the old system to the new one. While this method is faster it introduces significant risk due to day-one exploits and does not allow for gradual testing or user feedback.

Similarly, **pilot** implementation, while useful for large-scale systems, may limit broader user input during early phases and is less suited for this project’s streamlined functionality. Since my target market is school students, equal access to features should be imperative.

1.4. Financial feasibility

Students are to **conduct** a financial feasibility study, including producing an opening-day balance sheet, to assess whether their application is financially viable.

SWOT Analysis

Strengths	Weaknesses
• Simple navigation and intuitive design enhance usability. • Users can personalize study lists based on their preferences. • Mobile and web integration ensures accessibility across devices. • Node-network task labelling helps with prioritization. • Ensuring user data is stored safely builds trust.	• Early versions may lack advanced integrations like AI-assisted planning. • Keeping users engaged long-term could be challenging. • Early development stages may have glitches that need troubleshooting. • Managing performance as the user base grows may require adjustments. • Time and funding for development might restrict ambitious features. • Some users may struggle with customization options if too complex.
Opportunities	**Threats**
• Many students struggle with organization and look for digital solutions.	• Established study planners may attract more users.

• Partnering with educational platforms or schools can increase adoption. • Future updates could incorporate AI-driven study recommendations. • Adding shared study groups could drive engagement. • Schools are shifting toward digital learning, making apps like mine more relevant.	• Compliance with standards like GDPR could require extra precautions. • Apple, Google, or other app stores may impose policies that affect functionality. • Students may prefer simpler alternatives. • Security risks like data breaches could affect trust. • If offering premium features, balancing free and paid content is important.

Study	Go or No Go?	Assessment and evidence
Market feasibility	GO	There is a **growing demand** for **school productivity** tools. Apps like Notion, Todoist, and MyStudyLife have seen rapid adoption among students, with Notion reaching over 10 million users globally and trending on. Additionally, research shows students struggle with organisation and time management, creating a strong potential user base.
Cost of development	GO	Development costs are manageable due to access to **open-source frameworks** like **React** for cross-platform apps and **Firebase** for backend services

Study	Go or No Go?	Assessment and evidence
Cost of ownership	GO	Ongoing costs such as hosting, maintenance and updates are predictable and scalable with cloud infrastructure keeping operational costs low. Services like **Firebase, Heroku, or Vercel** offer free tiers or low-cost plans to support early growth.
Income potential	GO	The app can generate income through **freemium models**, subscriptions or **educational institution** partnerships in NSW.
Future expansion opportunities	GO	Students and schools have the **increasing need** for flexible, accessible, and affordable productivity tools. Possible disruptions like new competitors and **AI advancements** have been considered, however the community driven attitudes provide resilience.

Opening Day Balance Sheet

Study Plus Balance Sheet [AUD $]	2024	2025	2026
Assets			
Current assets:			
Cash	167,971	181,210	183,715
Accounts Receivable	5,100	5,904	6,567
Prepaid expenses	4,806	5,513	5,170
Inventory	7,805	9,601	9,825
Total current assets	185,682	202,228	205,277
Property & Equipment	45,500	42,350	40,145
Goodwill	3,580	3,460	3,910
Total Assets	**234,762**	**248,038**	**249,332**
Liabilities			
Current liabilities:			
Accounts Payable	3,902	4,800	4,912
Accrued expenses	1,320	1,541	1,662
Unearned revenue	1,540	1,560	1,853
Total current liabilities	6,762	7,901	8,427
Long-term debt	50,000	50,000	30,000
Other long-term liabilities	5,526	5,872	5,565
Total Liabilities	**62,288**	**63,773**	**43,992**
Shareholder's Equity			
Equity Capital	170,000	170,000	170,000
Retained Earnings	2,474	14,265	35,340
Shareholder's Equity	**172,474**	**184,265**	**205,340**
Total Liabilities & Shareholder's Equity	**234,762**	**248,038**	**249,332**
Check	*0.000*	*0.000*	*0.000*

2. Research and planning

2.1. Project management

Software development approach

For this project, I have chosen the **agile** approach due to emphasis on collaboration (clients) and flexibility. An agile approach divides the work into short cycles called sprints, encouraging frequent reassessments and adaptations. This approach is highly adaptive to changing requirements

A **waterfall** approach is well suited for fixed requirements, however the approach is inflexible due to a step-by-step approach where each phase must be completed. This is not ideal for projects where requirements evolve during development, and can lead to issues when communicating with the client. Overall, my small-scale project requires flexibility.

A **WAgile** approach can be useful when tackling a medium-scale system, however this approach may be too slow compared to Agile and Waterfall and can be confusing if roles or phases are not clearly defined.

Scheduling and task allocation

Students **develop** a Gantt Chart that details the tasks required to be completed, person or people assigned to each task, timeline that does not exceed the project due date, resources required. In addition, students **identify** any collaborative tools used. For example Repl.it, GitHub and so on.

2.2. Quality assurance

Quality criteria

Quality criteria	Explanation
The to-do list should effectively support organization and productivity.	Users must be able to add, edit, categorize, and prioritize tasks easily. Reminders and deadlines help users stay on track, enhancing study effectiveness.
User statistics should dynamically reflect study patterns	By visually representing study intensity, users can identify trends in their habits. Customizable time frames can improve clarity and engagement.
User data must be securely stored and handled responsibly.	Encryption, secure logins, and privacy policies ensure compliance with data protection laws. Users should have control over their information to maintain trust.

Compliance and legislative requirements

Students **explain** compliance and legislative requirements their projects need to meet and how they plan to mitigate them where possible. For example, projects that deal with sensitive personal data being publicly available may fall under the Australian NSW Privacy and Personal Information Act (1998) and/or Federal Privacy Act (1988). Alternatively, international standards on information security management such as ISO/IEC 27001 may also be applicable.

Compliance or legislative issue	Methods for mitigation
ISO/IEC 27001	Adopting basic principles from the standard such as risk assessment to identify vulnerabilities and implementing strong cybersecurity policies, including encryption and access control. I will also maintain thorough documentation of security protocols and audit records demonstrating compliance while enabling ongoing improvements.
Children's Online Privacy Protection	Including parental consent, limit data collection by collecting only necessary data, anonymise data where possible, and implement stronger data protection. Additionally, important data is encrypted and inactive accounts are automatically deleted.
Federal Privacy Act (1988)	Protecting personal information including names, emails, student progress etc. Store only necessary data, obtain user content, provide privacy data and allow users to request deletion of their data.

2.3. Systems modelling

Students are to **develop** the given tables and diagrams. Students should consult the Software Engineering Course Specifications guide should they require further detail, examples or information. Each subsection below should be completed with Section 1.1. in mind.

Data dictionaries and data types

Students take the needs identified in Section 1.1. of this Systems Report. For each need, students **identify** the variables required, data types, format for display, and so on.

Need							
1. Task list							
Variable	**Data type**	**Format for display**	**Size in bytes**	**Size for display**	**Description**	**Example**	**Validation**
status	int	N	4	N/A	Check task status: completed, ongoing or overdue.	1	Created by 'form', then updated with

Need							
							completed_task
label	text	XX..	12	12	Label for task contents, intended for subject attribution	Software	12 characters max.
body	text	XX..	128	128	Summary of task.	Revise for **History** exam with **Atomi** ***"boom"***	128 characters max.
title	text	XX..	48	48	Subject content of task.	History revision	48 characters max.
due_date	datetime	YYYY-MM-DD HH:MM:SS	4	17	Recorded time of final due date.	2025-01-23 21:22	Created by user field 'due_date'
user_id	int	N	4	N/A	A primary key, tied to their credentials and other information.	23	Handled by SQLite, integer

Need							
							primary key auto increment
todos_id	int	N	4	N/A	A primary key, tied to their credentials and other information.	23	Handled by SQLite, integer primary key auto increment

Need							
2. Secure Users							
Variable	**Data type**	**Format for display**	**Size in bytes**	**Size for display**	**Description**	**Example**	**Validation**

Need							
user_id	int	N	4	N/A	A primary key, tied to their credentials and other information.	23	Handled by SQLite, integer primary key auto increment
privacy	text	XX..	12	12	Configurable privacy settings: private or public.	private	From a drop-down list
layout	text	XX..	12	12	Configurable layout settings: sidebar or topbar.	private	From a drop-down list
last_activity	datetime	YYYY-MM-DD HH:MM:SS	4	N/A	Recorded time user last logged in.	2025-01-24 22:42:50	Created by 'datetime.now()'
email	string	X@X.X	255	255	Identification provided by the user and their way to log in.		

Need							
password	string	XXXXXXXN	8	8	A layer of authentication provided by the user.	helloworld1	At Least 8 characters including: 1 letter, 1 number, under 255 characters
firstname	string	XX..	16	16	First name of team member.	John	Only letters, 16 max
lastname	string	XX..	16	16	Last name of team member.	Doe	Only letters, 16 max

Need
3. Progress tracker

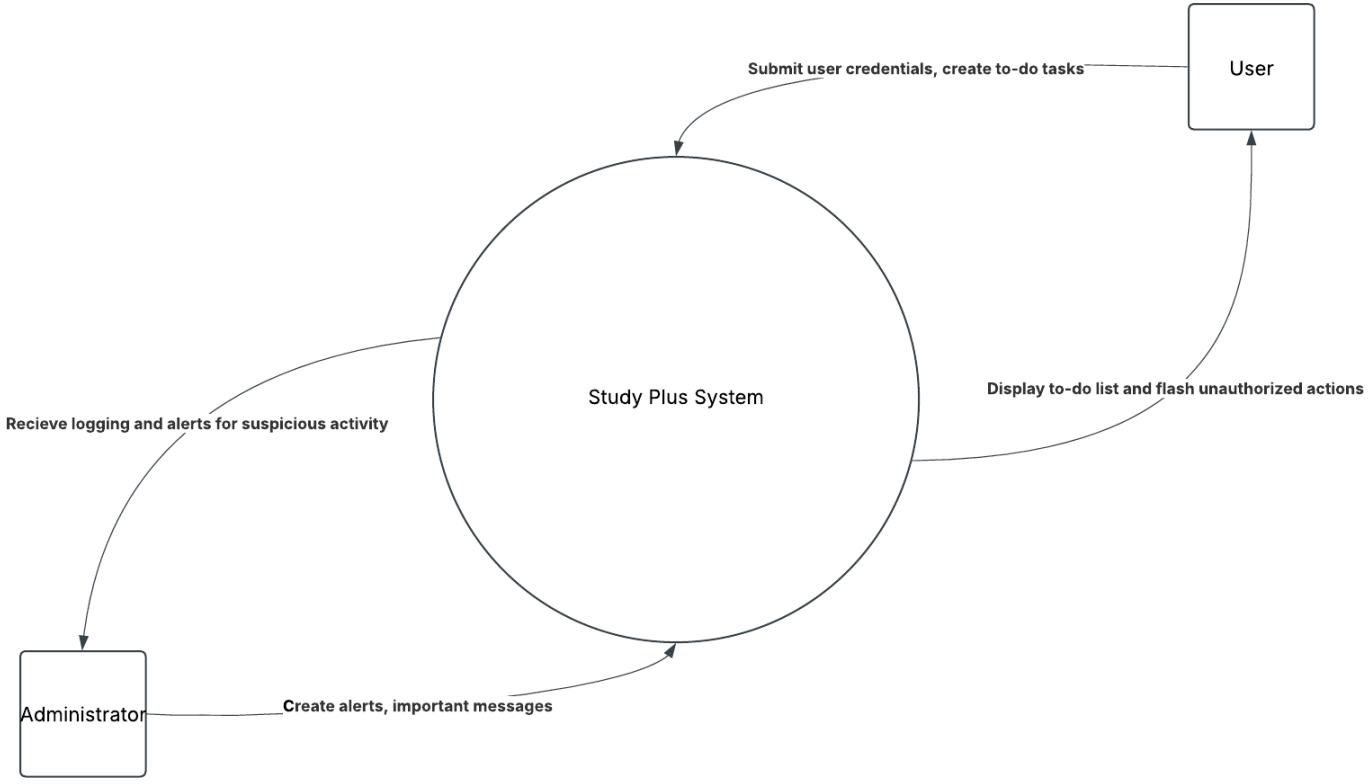
Need							
Variable	**Data type**	**Format for display**	**Size in bytes**	**Size for display**	**Description**	**Example**	**Validation**
completed_task	int	N..	4	4	Completed tasks associated with current user_id.	2	Handled by SQLlite
ongoing_task	int	N..	4	4	Ongoing tasks associated with the user.	2	Handled by SQLite
overdue_task	int	N..	4	4	Overdue tasks associated with the user.	2	Handled by SQLite
completed_at	datetime	YYYY-MM-DD HH:MM:SS	4	17	Recorded time of user completion.	2025-01-24 22:42:51	Created by 'datetime.now()'
user_id	int	N	4	N/A	A primary key, tied to their credentials and other information.	23	Handled by SQLite, integer primary key

Need							
							auto increment

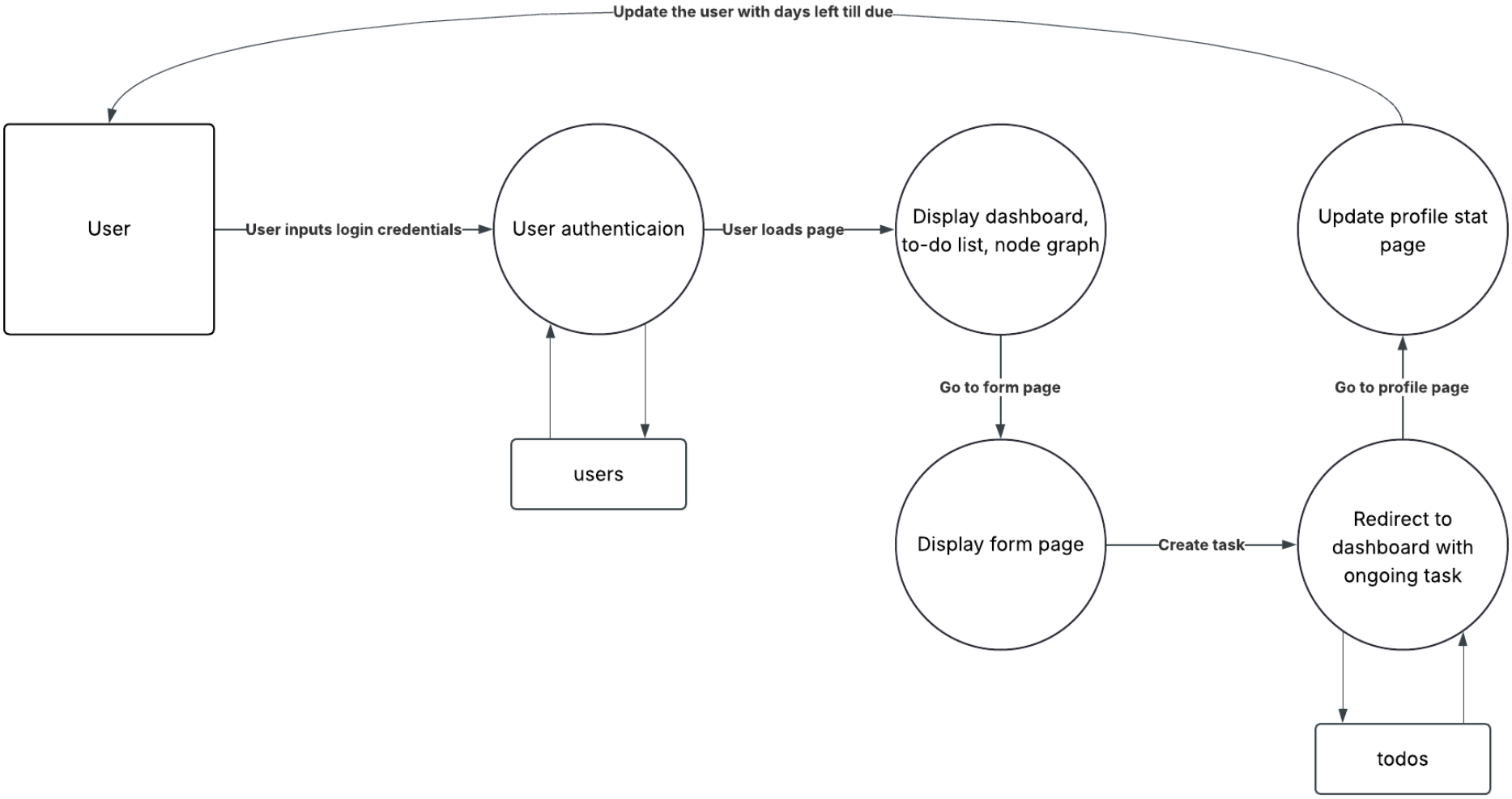
Data flow diagrams

Students **develop** data flow diagrams (DFDs) at Level 0 and Level 1. These diagrams should explicitly include the variables from the data dictionaries previously identified as well as the needs identified in Section 1.1.

Level 0

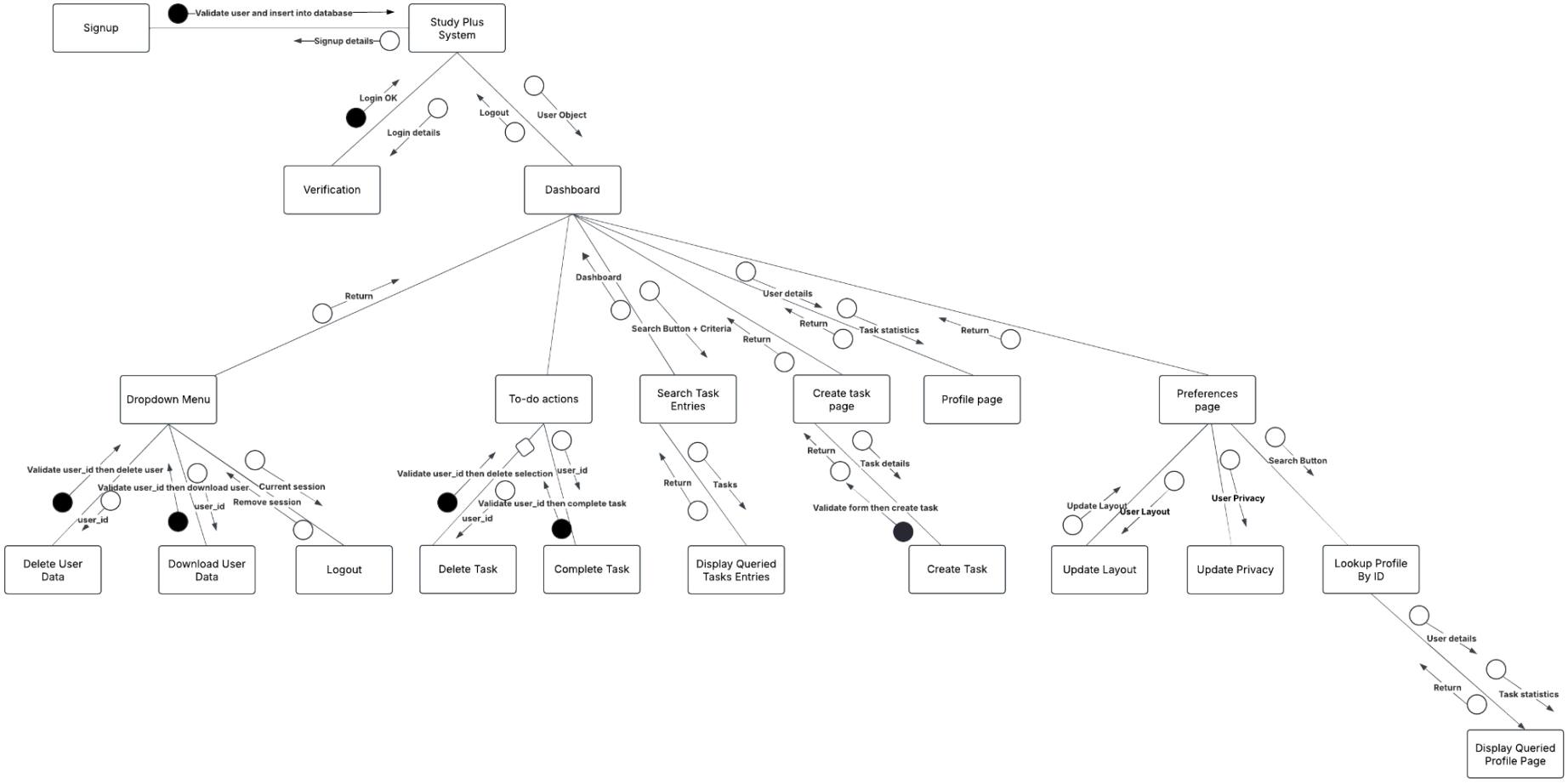


Level 1



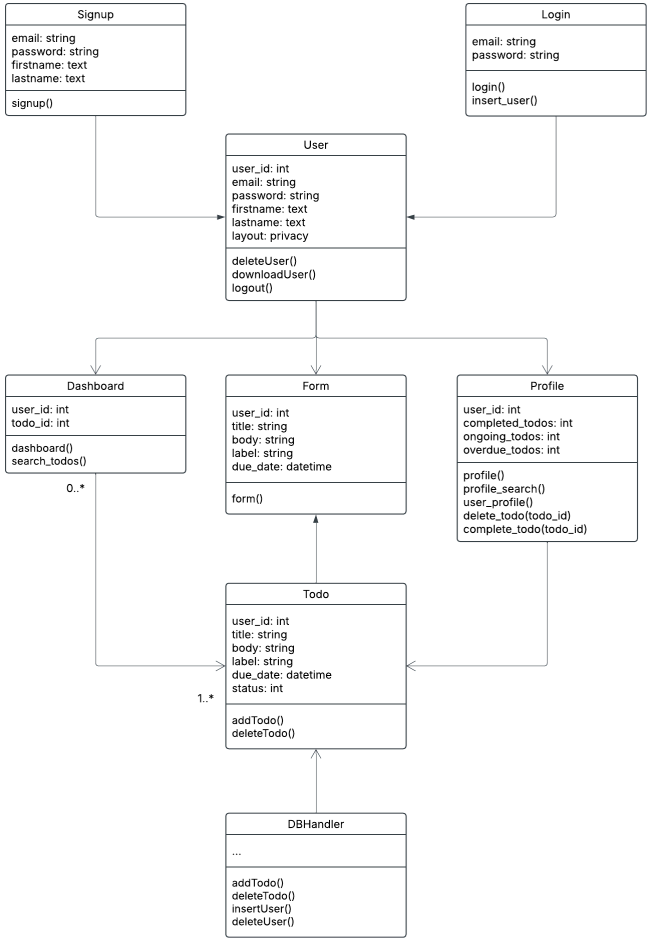
Structure charts

Students **develop** structure charts demonstrating how the procedures, modules or components of the final solution are interconnected.



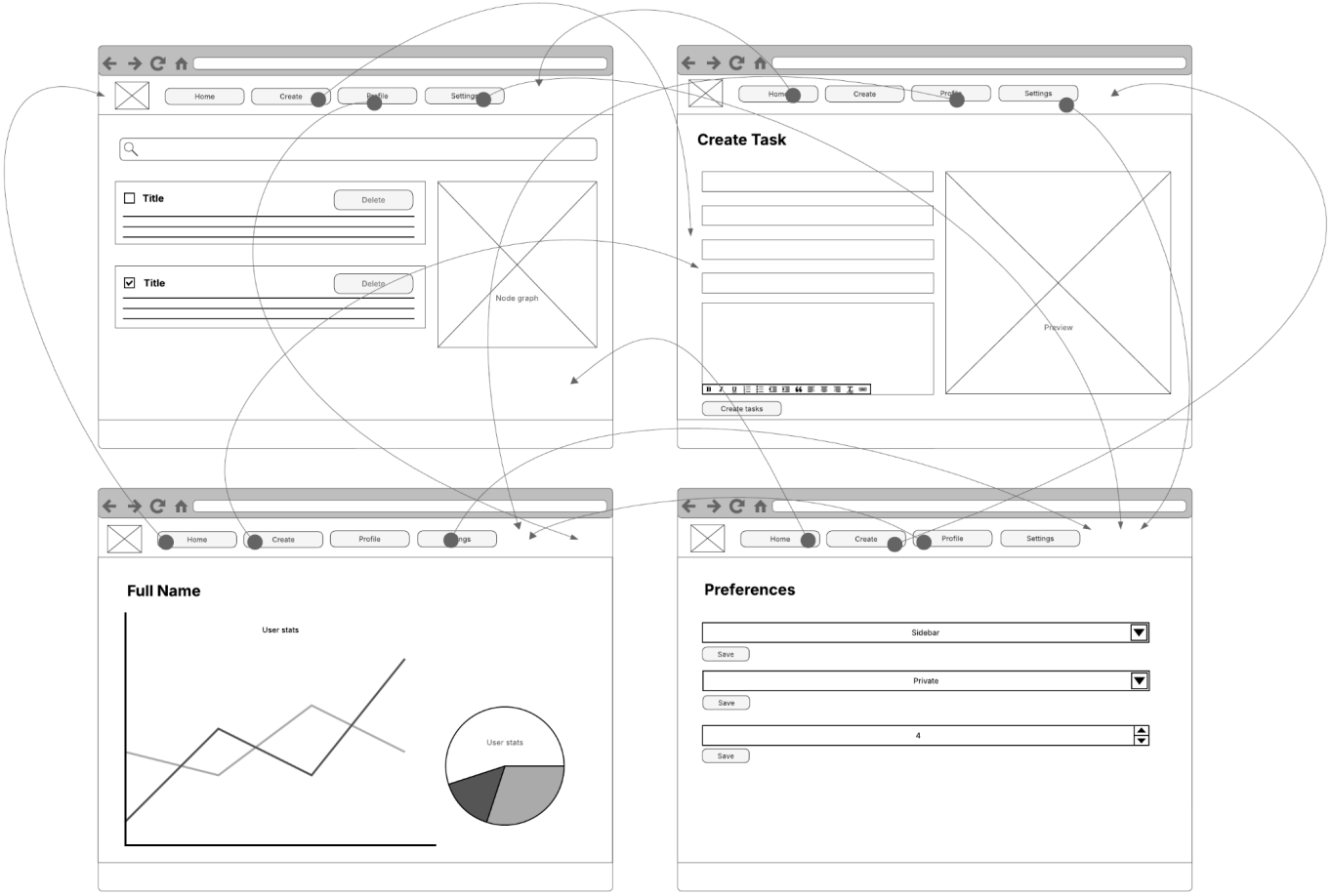
Class diagrams

Students **develop** class diagrams demonstrating how each class is related to the other.



Storyboards

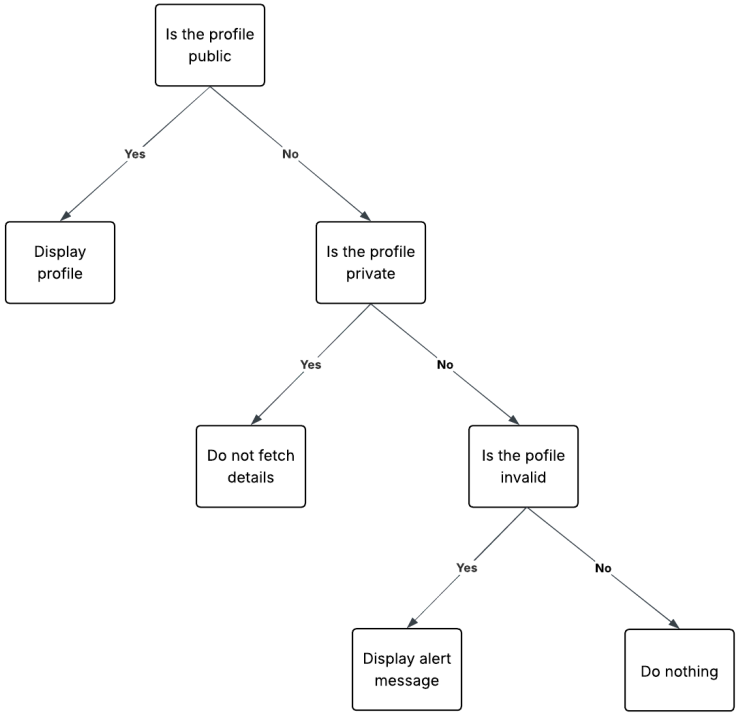
Students **develop** storyboards, visually representing the software solutions they will build.



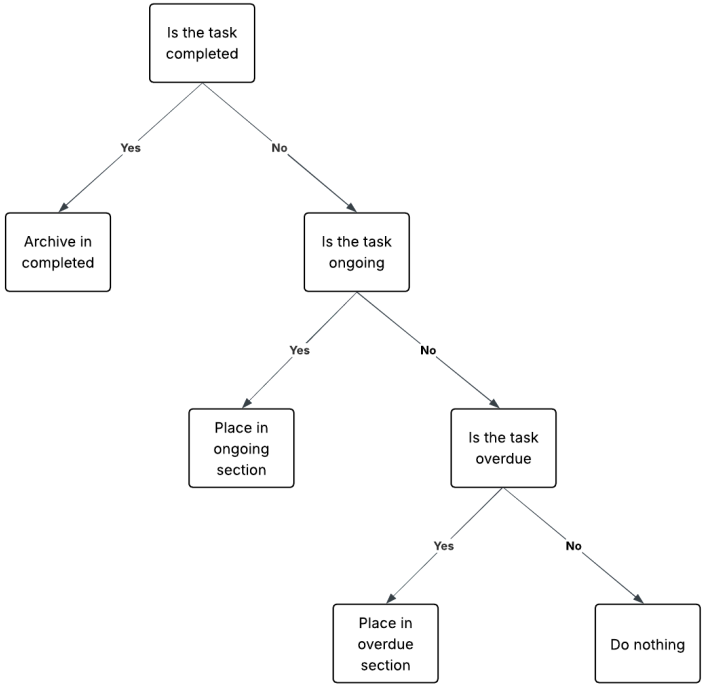
Decision trees

Students **develop** decision trees to visually outline the logic flow and chain of decisions or selections the final solution will need.

Profile Lookup By ID



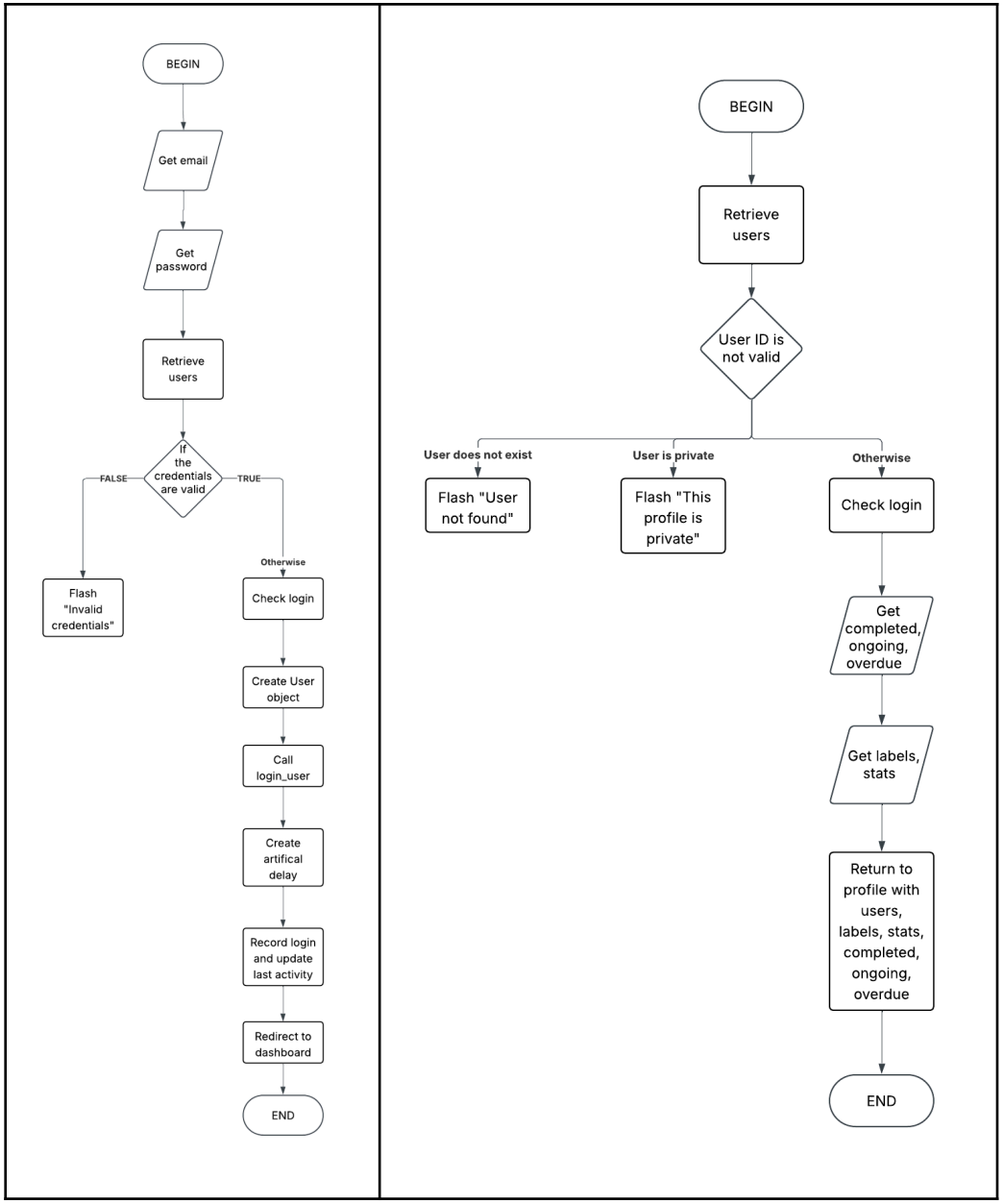
Task Status



Algorithm design

Students **develop** algorithms using methods such as pseudocode or flowcharts to solve the problem and meet the needs from Section 1.1. These algorithms should explicitly include the variables from the data dictionaries created in the previous section.

Login	*User Profile*



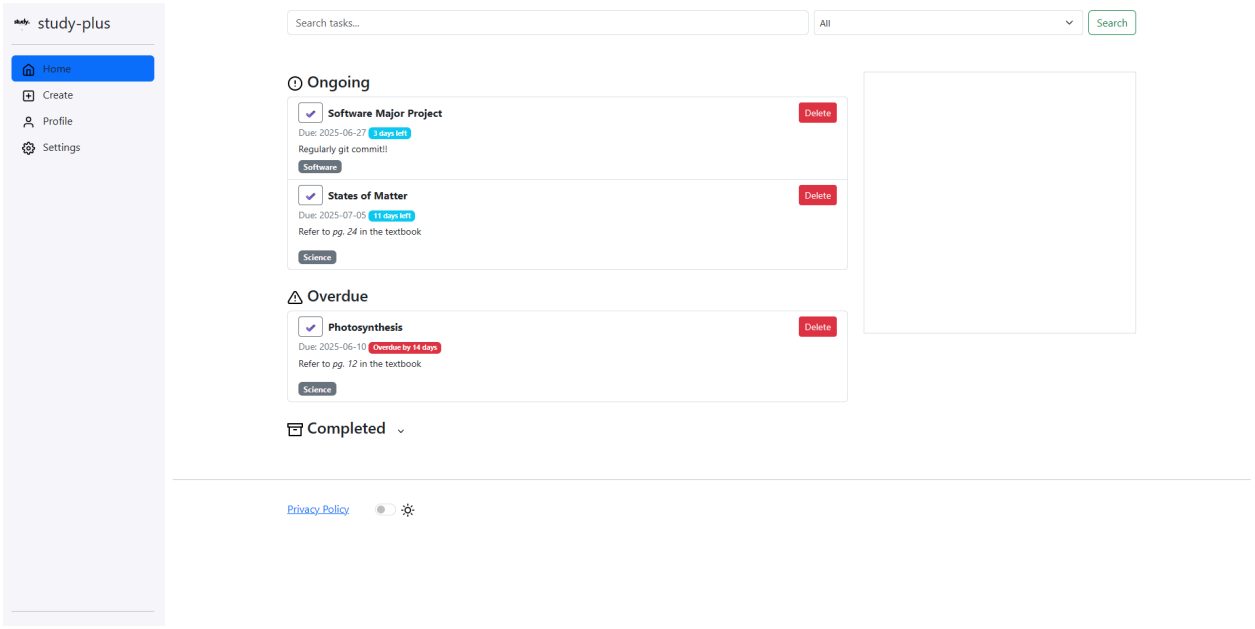
3. Producing and implementing

Solution to software problem

Students are to **include** screenshots of their final developed solution here. Each screenshot should include a caption that **explains** how it links to the:

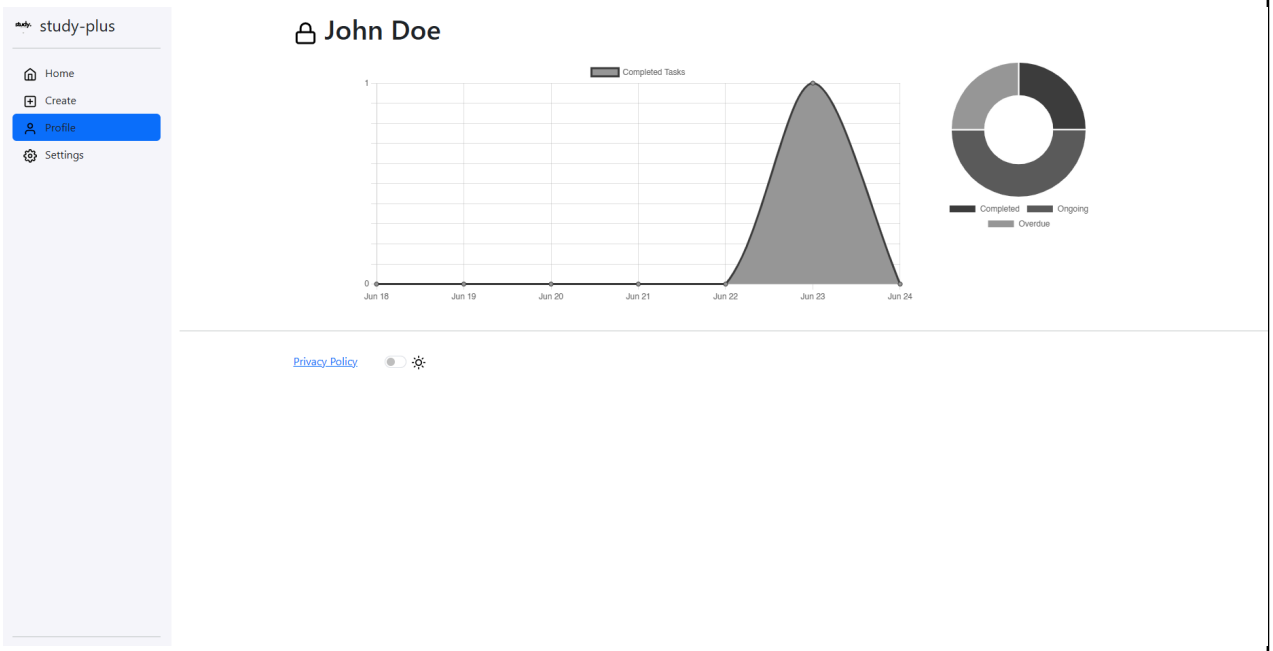
- Needs identified in Section 1.1.
- Components of Section 2.3. such as storyboards, data dictionaries and so on.

1. *Dashboard*



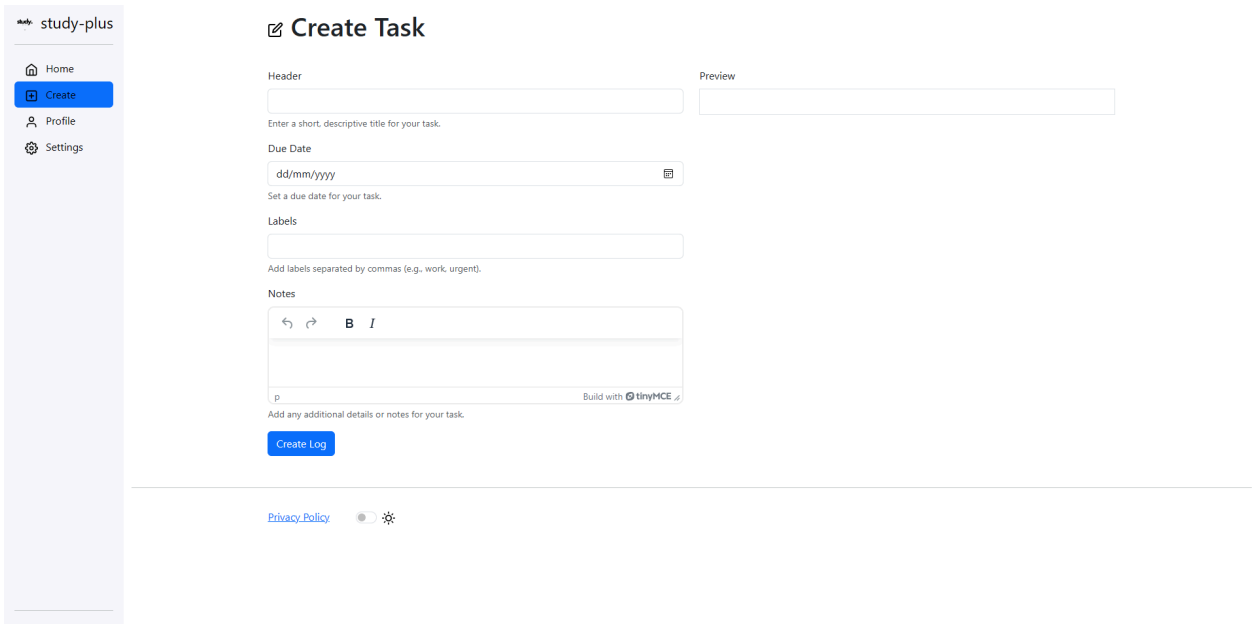
- *A clear and intuitive design with the option to complete and delete a task*
  - *Variables involved: todos_id, title, body, due_date, label*

2. *Profile*



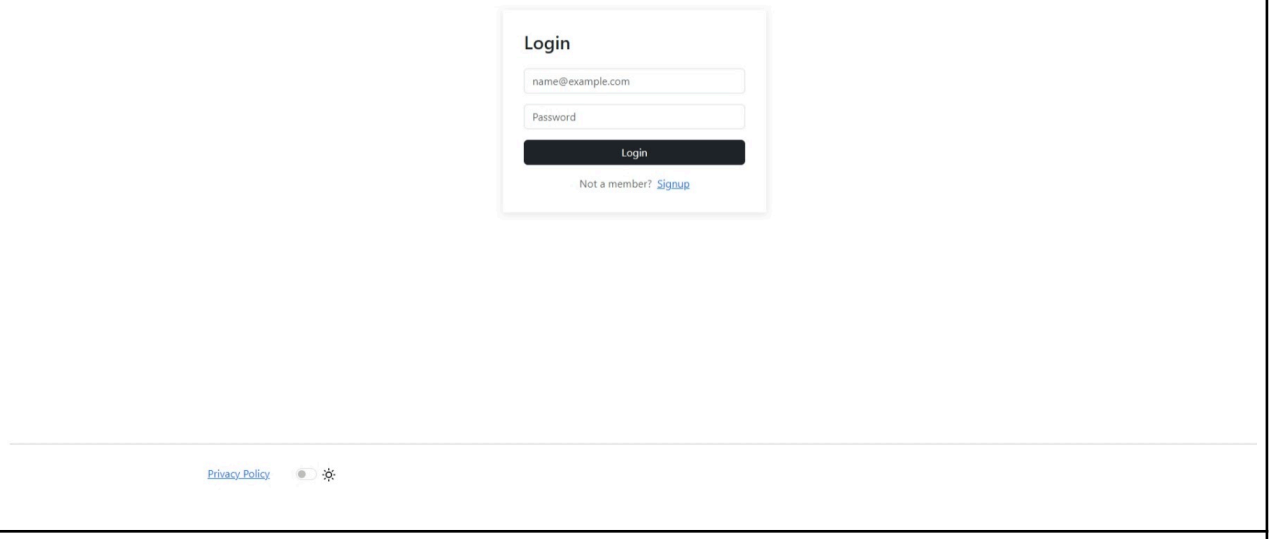
- *A visual representation of statistics with the use of a pie chart and line graph*
  - *Variables involved: completed_task, ongoing_task, overdue_task, firstname, lastname*

3. *Create Task*



- *A clear and intuitive design with the option to create tasks*
    - *Variables involved: title, body, due_date, label*

4. *Login*



- *A secure landing page while provides a simple way to login*
    - *Variables involves: email, password*

Version control

Students **describe** what version control system or protocol was implemented.

For this project, I opted to use **Git** as the primary version control system to track and manage changes to source code. Git allows for a clear commit history, branching for new features or sprints without affecting the main branch, and rollbacks for potential errors. The codebase was maintained through codespaces and pushed Github through a command-line interface.

Using Git significantly improved the development workflow especially with an agile approach and made the testing and debugging process more transparent and manageable.

4. Testing and evaluating

4.1. Evaluation of code

Methodology to test and evaluate code

I have used a variety of methodologies to **test and evaluate** code. Methodologies include:

- Unit, subsystem and system testing:
  - I tested individual components (functions, methods, classes) in isolation and ensured each part works before integration. This includes my profile lookup, todo completion etc.
- Black, white and grey box testing:
  - This allowed me to verify that features like question selection and score calculation worked correctly from a user’s perspective. This form of testing was conducted by a friend.
- Quality assurance.:
  - This allowed me to make sure my components (e.g. profile lookup by ID), were working correctly with a multitude of testers.

Code optimisation

I have used a variety of methodologies to **optimize code** so it runs faster and more efficiently. Methodologies include:

- Dead code elimination:
  - Removed unreachable or unused code to reduce execution time and binary size.
- Code movement:
  - Moved computations outside loops if they don’t change per iteration.
- Strength reduction:
  - Replaced expensive or resource intensive operations (e.g. multiplication) with cheaper ones (e.g. addition).
- Common subexpression elimination:

    - Avoided recalculating the same plans expressions multiple times by storing results
- Refactoring:
    - Restructured code for better readability and performance (modern compilers often do this automatically).
- Minification:
    - JS minification leads to faster load times especially with my website using many JS files for statistics and circumventing the strict CSP.
- Lazy loading:
    - Waiting to render content on a webpage until the user or the browser needs it.

4.2. Evaluation of solution

Analysis of feedback

Students **analyse** feedback given to them on the new system they have just created. This feedback can be in the form of an interview, survey, focus group, observation or any other applicable method. Students should also include overall positive, negative or neutral sentiments towards the new system in their response.

After an **interview** with my clients, the system received overall positive feedback. However, issues or feedback was centered on UI and UX design such as the completion box for tasks, which many users had mistaken for another button. They found the profile lookup quite intuitive with some advising a name search instead or UUID system. When logging in, users also assumed that they would be logged in automatically, and not lead back to the login screen which some found irritating.

Testing methods

Students **identify** the method or methods of testing used in this current project. For each they use, students are to **explain** how and why it was used.

Method	Applicability	Reasoning
Functional testing	Yes	Ensures all dashboard features such as search, filters, collapsable sections, rendering of tasks, work as expected and according to requirements.
User Acceptance testing	Yes	Validates that the system meets the clients needs and expectations including ease of use, correct data display and overall workflow. An example is the profile lookup by ID.
Live data	No	Risky to use in early testing stages due to data sensitivity.

Method	Applicability	Reasoning
Simulated data	Yes	Allows for safe and repeatable testing with controlled scenarios, edge cases, and mock tasks without impacting real users.
Beta testing	Yes	Useful for gathering feedback from real students in a controlled rollout before deployment.
Volume testing	Yes	Can test how the dashboard and profile performs with large numbers of tasks, important for evaluating performance and load handling

Security Assessment

Students are to **perform** an extensive security assessment of their final application and e**xplain** the countermeasures implemented.

Threat	Countermeasure
Broken authentication and session management	Exposed session cookies can lead to impersonation by the threat actor. Poor authentication systems which lack security policies can lead to brute forcing and human error. 1. Strong passwords policies were enforced to prevent users from creating easily cracked passwords. This involves an 8-character minimum with at least 1 letter and 1 number. 2. The site had strict expiration policies and rate limiting for login, download user data and create post submissions. This secures user data and prevents brute forcing. 3. Flask login was opted as it has many inbuilt security features such as @login_required for authorized pages. 4. SSL and HTTPS were provided which encrypts data and information.
Cross-site scripting (XSS) and cross-site request forgery (CSRF)	Cross site scripting (XSS) involves injecting malicious code directly into the front end, allowing threat actors to execute arbitrary code on a user's browser. Cross-site Forgery (CSRF) tricks an unknowing user into submitting a potentially malicious request to a trusted website. This can lead to alteration in website contents and potential fraud.

Threat	Countermeasure
	Input validation and sanitization was applied across every user interaction. These functions are stored under ‘sanitise_and_validate.py.’ 1. A Strict Content Security Policy (CSP) was enforced which blocks iframe and unsafe JS code from executing. In-line scripts were run through a nonce, exempting them from being blocked. 2. Anti-CSRF tokens implemented with Flask CSRF Protect included with each request, being associated with the current user. The secret key is also securely stored in an environment unlike being hardcoded.
Invalid forwarding and redirecting	Threat actors can manipulate a poorly secured URL which looks like the provided domain but instead leads to a malicious site. This can lead to phishing attacks or stolen credentials. 1. URL parsing implemented with every GET request ensures the format is properly formatted. If not, the code returns false. 2. Forwards and redirects are done only through Flask. 3. Only authorized/allowed URLs were allowed which are listed.
Race conditions	Race conditions involve multiple threads which access shared data and result in the operation finishing too late or too early. This can lead to unexpected bugs.

Threat	Countermeasure
	1. Randomized cryptography was prominent in my authentication algorithm, which ensures salted outcomes are unique. 2. Flask’s Application Context allowed for each request to use its own gateway, making it impossible for simultaneous connections.

Test data tables

Students **identify** variables which were used for either path and/or boundary testing. Students **develop** these test data tables based on their algorithms versus their real code. Students then **state** the reason for including said variables.

Boundary testing

Variable	Maximum	Minimum	Default Value	Expected Output	Actual Output	Reason for Inclusion
label	12 characters	1 character	None	Blank rejected; max accepted	As expected	Check label input boundaries and truncation errors
firstname	16 letters	1 letter	None	Accept only letters, max 16	As expected	Check alphabetic-only input + trimming behaviour
body	128 characters	1 character	None	Blank rejected; max accepted	As expected	Check if Jinja filters and/or escape disrupt the max character limit

Analysis of solution against quality success criteria

Students are to take each quality success criteria from Section 2.2 and place it here. For each quality criteria, **analyse** the components of the solution that met or did not meet each quality criteria. Give reasons why each success criteria were or were not met.

Quality criteria	Met?	Analysis
The to-do list should effectively support organization and productivity.	Yes	The to-do list is intuitive and allows users to easily add, delete, and manage tasks. The layout with clear sections/dividers promotes a clear overview of responsibilities, improving productivity.
User statistics should dynamically reflect study patterns	Yes	The system tracks user activity and presents data such as time spent, study frequency, or task completion. This real-time feedback encourages better study habits.
User data must be securely stored and handled responsibly.	Yes	All user data is stored securely using appropriate encryption methods. Access is limited, and no sensitive information is shared or mishandled. Compliance with privacy principles is ensured.