Timely

Contents

Timely

Analysis

The Problem

Research shows that people have different “peak productivity times” according to their circadian rhythm and chronotype (“Morning person” or “Evening person”). For example, *Reilly et al. (1998)* found that even under controlled conditions, human performance (reaction time, processing rate, hand-eye coordination) varies by 3-11% across the day indicating there are highs and lows in productivity, confirming that there is indeed a time of “peak productivity”. Other studies show that these peaks differ between people. *The Korean Work, Sleep and Health Study (2022-2024)* reported that evening chronotypes were 2.29 times more likely to produce poor work and suffered 5.36% productivity loss when working in the morning. Similarly, *Gaggero & Tommasi (2023)* demonstrated an upside-down “U” relationship in exam performance, with students performing best at 1:30 PM and gains of 4-6% if exams are rescheduled to their “peak productivity” time. Altogether, these findings confirm the theory that productivity is not fixed but rather fluctuates throughout the day and every person has a natural period where they are most effective.

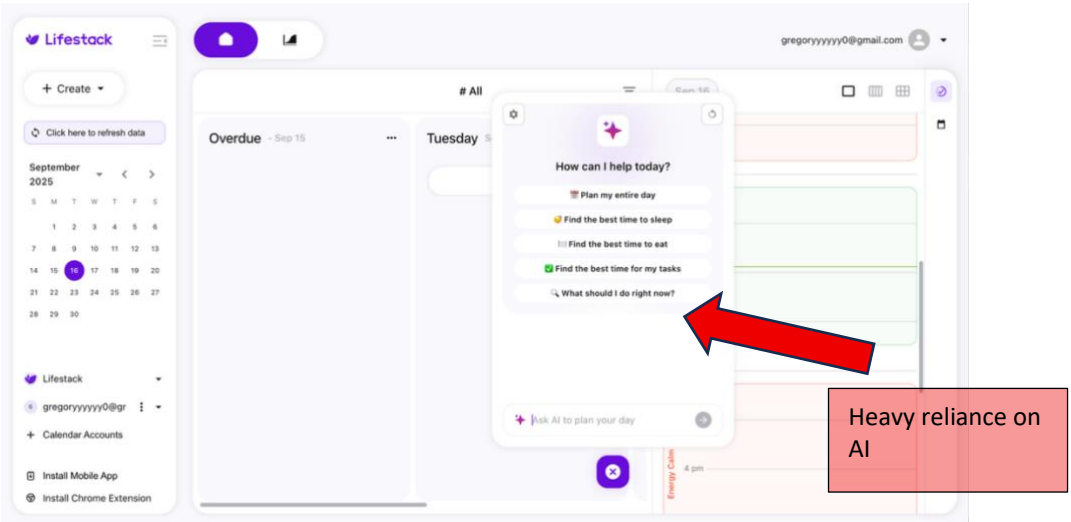
Timely

The solution

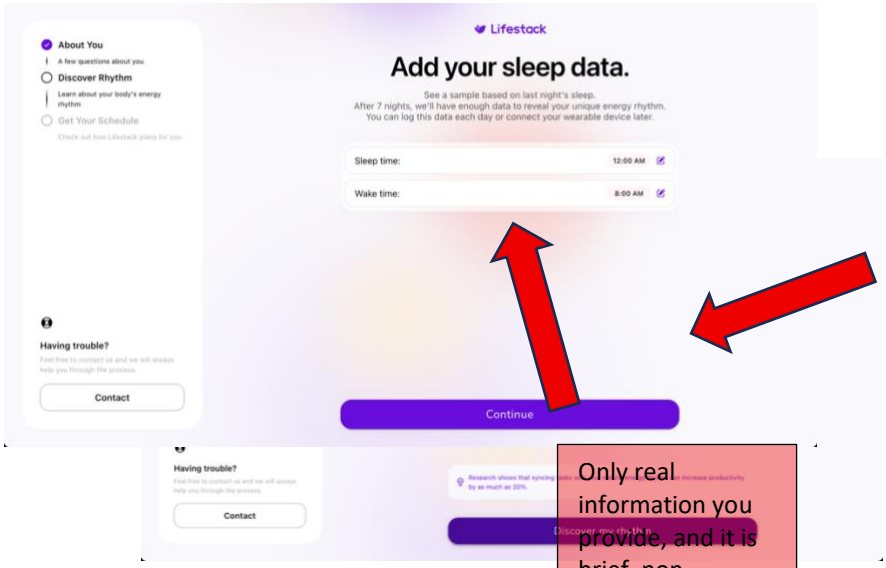
To keep my work high quality while staying stress-free and being time efficient, productivity is crucial. I find that spotting periods of “peak productivity” is simple, but the real issue lies with fitting tasks into a small time frame. Occasionally, I would end up cramming tasks which, in turn, slows me down because I can’t give any task enough attention.
On top of that, I would often misjudge the length of a task or get distracted from an inexplicable interruption which would increase the time taken to complete a given task, encroaching on time allocated for other tasks. After voicing this problem to friends and family, I discovered that this was a common perception of many. To tackle this, I created an app that helps organise tasks in a simple, yet effective manner and taking times of “peak productivity” into account.

Competition

From my research, I’ve found only one other product that offers a similar service: Lifestack. It has a timetable-style design though it relies heavily on automation. For example, it asks you for the time you sleep and wake up, then automatically generates recommendations such as “the best time to eat lunch”. These recommendations rely on limited data and overlook that decisions like when to eat are highly personal, often influenced by social or lifestyle factors that cannot be captured through the small amount of information you provide. Because of this reliance on automation, Lifestack feels over-engineered. Most of its features are centred around a generic AI model, which risks producing irrelevant or outdated information instead of tailoring schedules to a user’s actual needs. The result is a tool that prioritises AI-driven actions over productivity, making the experience feel very bloated.

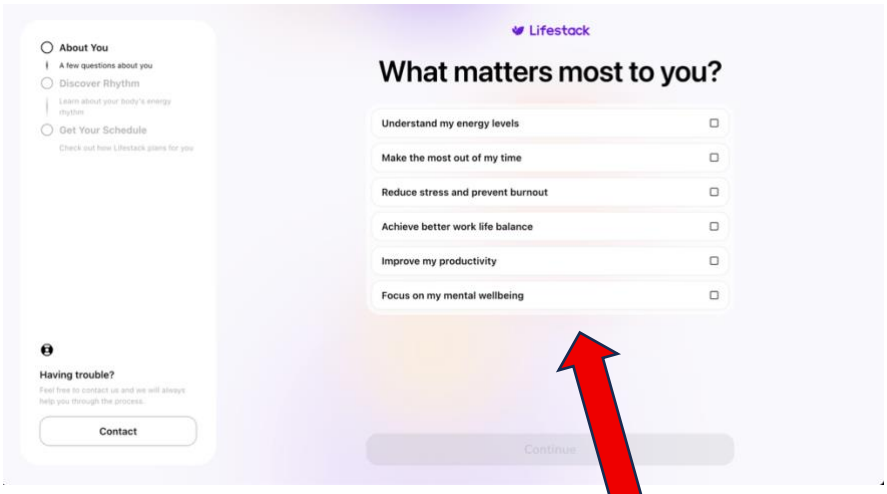


Timely



False promises (impossible to measure productivity on a scale)

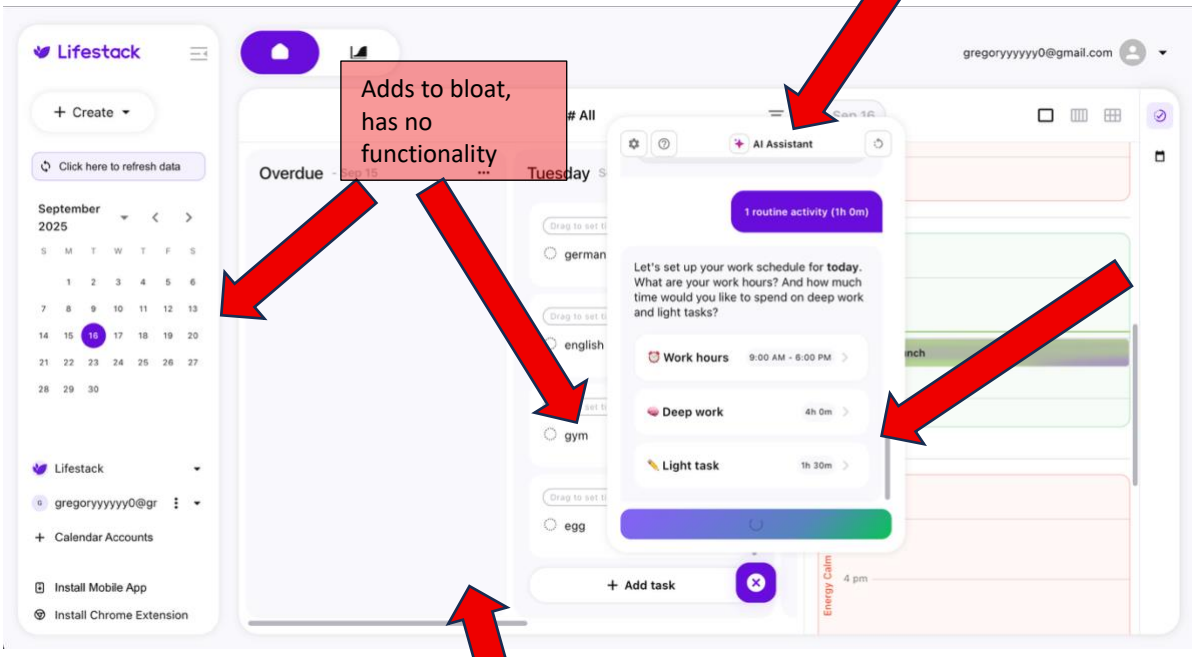
Only real information you provide, and it is brief, non-definitive and not enough to base information off of



This information has no impact on planning a timetable

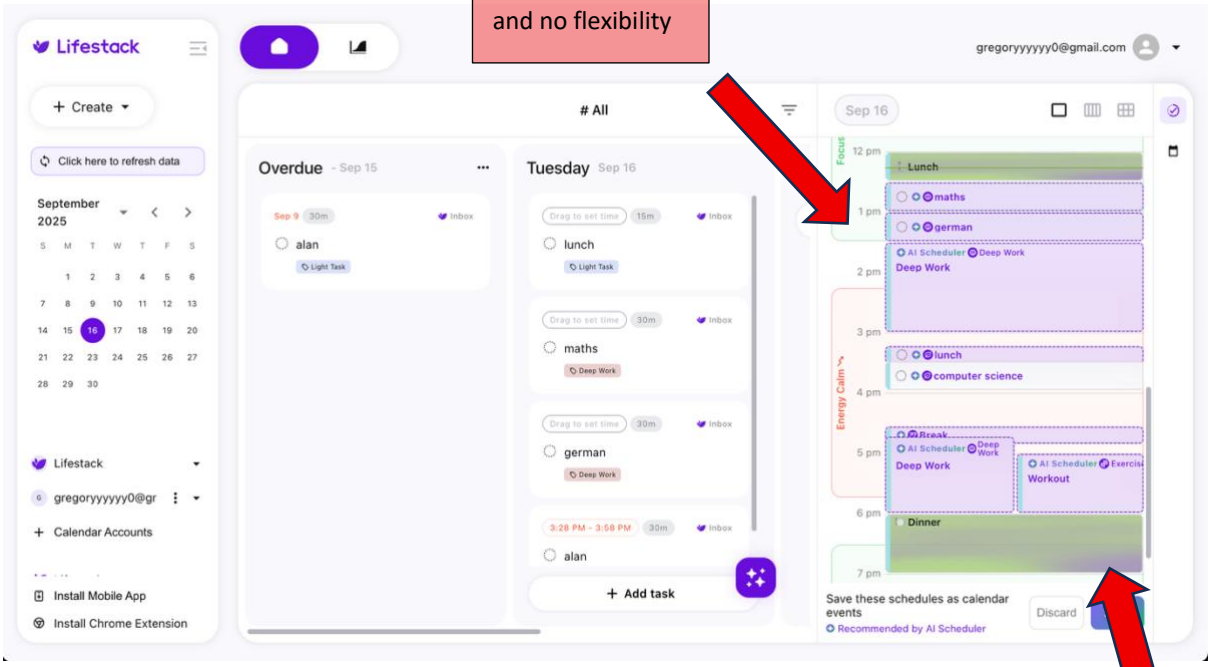
Does not account for personal, social or lifestyle factors

Timely


Generic AI model
Adds to bloat, has no functionality
Prioritises AI-driven actions over real productivity
Unclear and cluttered UI
Lifestack
+ Create
Click here to refresh data
September 2025
Lifestack
Calendar Accounts
Install Mobile App
Install Chrome Extension
gregoryyyyyy0@gmail.com
All
Overdue
Tuesday
AI Assistant
1 routine activity (1h 0m)
Let's set up your work schedule for today. What are your work hours? And how much time would you like to spend on deep work and light tasks?
Work hours 9:00 AM - 6:00 PM
Deep work 4h 0m
Light task 1h 30m
german
english
gym
egg
+ Add task
4 pm
Energy Calm

Timely

Doesn't tailor data to the user's needs. Instead, it lumps all tasks together with no room for breaks and no flexibility



Produces non-sensical data (does not set tasks at times of peak energy levels)

Interview with client

The following is a transcript from an interview in which I (D) interviewed an end-user of my project (B) who is a teacher and would like help with organising her schedule:

B: I'd like something to help me organise my time with my work-related tasks that must be completed.
D: Do you have regular tasks you have to do?
B: Yes, such as review my weekly planning.
D: What do you mean by review?
B: Well, I go through what I have planned with my colleague and check that I have all the resources ready that I need and spend time thinking about differentiating for the different abilities in my class. I also spend time thinking about how I am going to deploy the adults

Timely

that I have in class so that I get a chance to spend some quality time with all the children during the week understanding where they are at and their next steps.
D: Do you do this for every lesson?
B: Yes, I find it helpful.
D: Do you spend the same days with the same group of children?
B: I prefer to spend different days with the children each week depending on the task.
D: How long does this take?
B: Not sure as I have never timed myself.
D: Do you always use you lessons from last year?
B: No, I like to research and see what is new and see if there is a better lesson idea for the objective.
D: When you find new lesson, how do you create a lesson?
B: First, I spend some time looking for previous lessons on different website for ideas, plans and inspirations. Then I think about the end task that I want the children to complete and create 3 versions to cater for all the abilities. Finally, I either download a PowerPoint or multiple PowerPoints and edit them, if none of them work I create my own.
D: That sounds like a long task.
B: Yes, it is, but doing the research and finding the inspiration is what takes the longest.
D: How long do you think it takes?
B: Again, I don't know, sometimes it's quick if I find a good inspiration, or it can take me ages as I download and check multiple times.
D: How many times do you think you download PowerPoints to investigate?
B: Probably 3 to 4 before I give up and make my own.
D: What else do you do to be prepared for the week?
B: Not much else, I just print off my final timetable and 1 copy of anything that I created.
D: To help me understand what you do and the time it takes, is it ok if I watch you and time you?
B: Sure

D: So, what is the first thing that you are doing?
B: looking at the timetable I created at school during my PPA time to think about what I still need if anything.
D: I will time you
**** D timing task – 8 minutes
D: What are you doing now?
B: Going through the maths PowerPoints and videos so I know what I am doing this week
D: Why, is it not the same as last year?
B: We use a maths scheme, and they do change the lessons content and order.
My colleague downloads all the PowerPoints and she creates the adapted worksheets, so I always like to see what I will be teaching this week
**** D timing task – 21 minutes
D: Do you have any new lessons?
B: Yes, I do. It's my task to make the English lessons for this week and we want to focus on writing simple sentences concentrating on using capital letters at the start of a sentence, finger spaces and full stops at the end of the sentence.

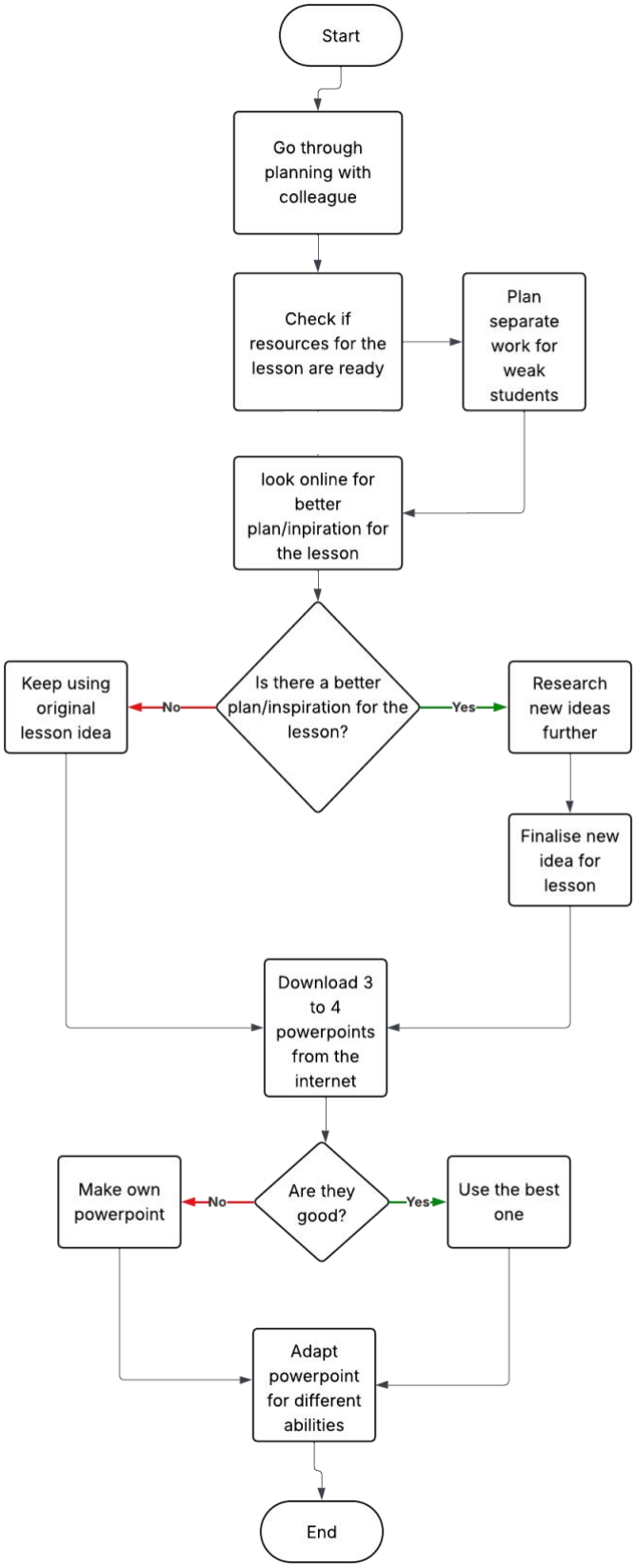
Timely

D: It's very basic.
B: Yes, I agree, but we got to get it right to make writing easier.
D: I am going to watch and ask you what you are doing, or would you mind telling me what you are doing and why.
B: OK
B: I am going to look though some websites for simple images that the children can write a sentence about using CVC words.
**** D observing and timing task - 12 minutes
B: I can't find any suitable, so I am going to see what AI can generate given a prompt.
D: Do you think you usually spend this much time looking on website before using AI?
B: I think so, I do try to limit it to 15 minutes otherwise I could be looking for a long time.
D: Why didn't you use AI at the start?
B: Because usually I can find what I am looking for as there are lots of good teacher resources websites on the net.
**** D observing and timing tasks. The AI takes a long time to generate images that B is happy with. – 27 minutes
Total time finding pics – 39 minutes.
D: Have you been teaching this already?
B: Yes, we did some last week. So, I am going to use the same PowerPoint as last week but update the pics. Familiarity works well when repeating tasks, it helps to build their confidence.
**** time taken to update PP – 10 minutes.
D: Are you finished now?
B: not yet, I am just going to print off the pics and laminate them so the children can write sentences on the laminated sheets.
**** D timing task – B adds pics to word doc adds lines, prints – 10 minutes
**** D timing task – B laminates 10 pages – 7 minutes
B: the English lesson is done.
**** B spending time looking at other lesson spending on average 5-7 minutes checking PPs.
B: that's it, I am done for this week.
D: Thanks B, Can I watch you again next time to see how your timing is differs?
B: Sure.

Following week, B spending roughly the same amount of time on some of the tasks.
She also researched and created a new PP for a RE lesson which took her 45 minutes.

Timely

Overview of lesson planning (pre-solution)



Timely

The issues are evident: it's an incredibly long process just to plan a lesson that must be repeated at least 3 times per day (one for each lesson). Though it promotes new and innovative ideas, the entire process is scrambled, chaotic and can be simplified to increase efficiency and reduce the amount of time wasted. The process of preparing a day's lessons took around two hours, highlighting how time-intensive these routine planning tasks are without support. A smarter, tailored tool could greatly reduce this burden.

We held another meeting to discuss issues with the current workflow and how *Timely* can aid these issues in a simple, streamlined way

Interview with client (2)

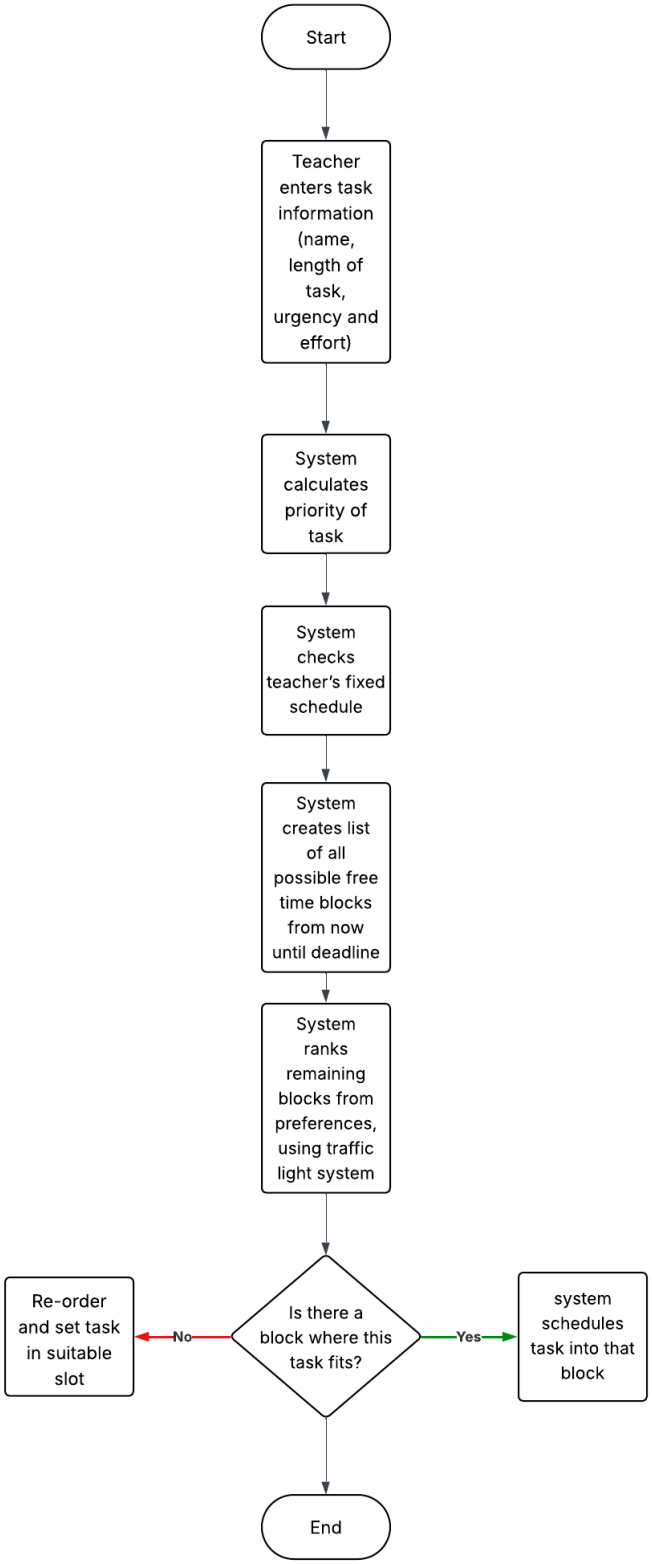
D: I have been thinking about how I can create something to help you.
D: I think having a list of the regular tasks is good. I can create this from the observations. What do you think?
B: How will the list be helpful?
D: I am going to assign a time for each task based on when I observed you.
B: How will it work?
D: The site will have a list of tasks and you choose which ones you are doing that day. This will create a list with the pre-determined times.
B: Sounds good, will I have any control over the list? Can and add and remove regular tasks?
D: I will add that into the page.
B: I think the user the power to adjust the time for the task will be helpful too.
D: Would you like an alarm to tell you that the task time is over? Would that be helpful?
B: No, I don't think so, I think that would be more frustrating.
D: Is there anything else that would like?
B: Not that I can think of. I am looking forward to trying this out.

Final client requirements

1. The system should automatically assign pre-determined times to each task based on previous observations.
2. The system should have a list of tasks that the user can add and be automatically assigned to an appropriate time.
3. The client should have control over the task list, including the ability to add or remove regular tasks.
4. The client should be able to adjust the time allocated for each task.
5. The system should not include distracting notifications e.g. alarms.
6. The overall design should be simple and user-friendly for easy interaction.

Timely

Overview of lesson planning (post-solution)



Timely

Final Objectives

1. The solution will be user-friendly
    a. Scores 68+ on the System Usability Scale (SUS)
        i. SUS is a 10-question survey yielding a score 0-100
        ii. Industry benchmark is around 68
2. Database is used to store information
    a. Personal information stored in database
        i. Includes: Username, Password, Chronotype
    b. Previous tasks stored in database
        i. Includes: Name, Effort, Urgency, Length
3. Task history tracking
    a. System stores min. 50 previous task entries per user
    b. Task data is stored for 90 days
    c. Database query returns task information in <2 seconds
4. The solution can assign pre-determined times to each task based on previous observations.
    a. Uses frequency analysis from previous database entries to predict time slot of a task
5. The solution will list recommended tasks
    a. Solution will query task that has been added the most
    b. List the task(s) in the sidebar
6. Client can add regular tasks
    a. Client clicks “Add regular task”
    b. New window opens with regular task information that the user inputs
    c. Regular task is added to timetable based on the information the client inputs
7. Client can remove regular tasks
    a. Client clicks “remove regular task”
    b. New window opens where the user selects the regular task they want to remove
    c. Regular task is removed from timetable
8. Client can add tasks
    a. Client clicks “add task”
    b. New window opens with task information that the client inputs
    c. Task is added to timetable
9. Client can adjust task
    a. Client clicks “adjust task”
    b. Client can edit task name and task length
    c. Information is updated on timetable
10. Client information is secure
    a. Password is encrypted with 256-bit encryption

Timely

   b. Password requirements: minimum 8 characters, 1 uppercase, 1 number, 1 special character
11. Tasks are assigned by priority and chronotype
   a. Priority is calculated from Effort, Urgency and Length
   b. Score range: 0-10
   c. High priority (>7) assigned to chronotype peak hours in 90%+ of cases
   d. Low-priority tasks (priority score <4) scheduled outside peak hours in 90%+ of cases
      i. Chronotype refers to the period when the user produces their best work/has the most focus
12. Length of task corresponds to Length on timetable
   a. E.g. If task length is 2 hours, it takes up 2 hours on the timetable
13. Tasks do not Interfere with each other
   a. System prevents overlapping task assignments with 100% accuracy
   b.
14. Task validation
   a. All task inputs validated before database entry (name: 1-100 characters, length: Integer, effort/urgency: 1-10 scale)
   b. Zero duplicate task entries with identical timestamps
15. Quick response time
   a. Page load time <3 seconds on average Wi-Fi (521 Mbps)
   b. Timetable UI reacts to updates within 1 second
16. Colour coding
   a. Peak Productivity times are green coloured
   b. Regular tasks are blue
   c. Tasks are yellow
17. Data backup
   a. Client can export their data as csv

Timely

Documented Design

Introduction

I am creating timetable app that helps the user plan their week ahead that uses science-based methods to boost productivity. First, users will sign up using their email and creating a password. Their password will be encrypted to ensure security. Users will enter a short profile that includes personal preferences such as chronotype (morning person or evening person) and how much water they drink daily. Then users will add the tasks that have a specific time to start and end. Finally, users will enter the tasks that they want to complete later, alongside information about the task, such as task length (hours), task effort and task value. Using the information given from their personal preferences and task details, the app will automatically assign their tasks to a specific time which is tailored to the user's preferences, allowing them to complete their tasks productively and prevent energy wastage. If the user has assigned a specific task a few times before, the app will try to predict the time the user would like to begin the task based on previous entries. The app will include various other features which work separately to boost the user's productivity. One example of this is a reminder that pops up to remind users to take a short, 10-minute break. I will be using Python 3 for this project

Algorithms

The following are algorithms I will use for different features in the system. I will also give an example of how I will implement them using pseudocode:

The app will use a weighted scoring system to calculate priority. By combining effort, urgency, and length into a single quantifiable score, it provides an objective method for ranking tasks. This ensures users can make informed decisions about what to tackle first rather than relying on subjective judgment.
Pseudocode:

```
FUNCTION calculatePriority(effort, urgency, length, w1, w2, w3):
    // Validate weights sum to 1
    IF w1 + w2 + w3 != 1 THEN
        THROW error "Weights must sum to 1"
    END IF

    // Calculate weighted priority score
    priority = (effort × w1) + (urgency × w2) + (length × w3)

    RETURN priority
END FUNCTION
```

For the time allocation algorithm, I have decided to use a Greedy Scheduling algorithm, where tasks are sorted by priority first, and each task will find earliest available time slot that fits and if the task is high priority, search within chronotype peak hours, assign task to slot, mark slot as occupied. By sorting tasks by priority and assigning them to the earliest

suitable slots, it maximizes schedule efficiency. The integration of chronotype peak hours for high-priority tasks ensures users work on important items during their most productive periods, optimizing both time usage and work quality.

Pseudocode:

```
FUNCTION greedyScheduling(tasks, schedule, chronotypeHours):
    // Sort tasks by priority (descending order)
    sortedTasks = SORT(tasks BY priority DESC)

    FOR EACH task IN sortedTasks DO
        // Determine search window based on priority
        IF task.priority >= HIGH_PRIORITY_THRESHOLD THEN
            searchWindow = chronotypeHours
        ELSE
            searchWindow = ALL_AVAILABLE_HOURS
        END IF

        // Find earliest available slot
        slot = findEarliestSlot(schedule, task.length, searchWindow)

        IF slot IS NOT NULL THEN
            // Assign task to slot
            schedule.add(task, slot.start_time, slot.end_time)
            // Mark slot as occupied
            markOccupied(schedule, slot.start_time, slot.end_time)
        ELSE
            // No available slot found
            addToWaitlist(task)
        END IF
    END FOR

    RETURN schedule
END FUNCTION
```

For the time prediction algorithm, the app will use frequency-based matching, using previous entries of the task in the database from the last 30 days to see which time is most common to begin a given task. This creates a personalized scheduling experience that adapts to individual habits and preferences over time, making the app increasingly useful with continued use.

Pseudocode:

```
FUNCTION predictTaskTime(taskName, database):
    // Retrieve task entries from last 30 days
    thirtyDaysAgo = CURRENT_DATE - 30
    entries = database.query(
        FROM history_records h
        JOIN tasks t ON h.taskId = t.taskId
        WHERE t.task_name = taskName
```

Timely

```
        AND h.date >= thirtyDaysAgo
    )

    IF entries.isEmpty() THEN
        RETURN NULL
    END IF

    // Count frequency of start times
    timeFrequency = MAP()
    FOR EACH entry IN entries DO
        startTime = entry.startTime

        IF timeFrequency.contains(startTime) THEN
            timeFrequency[startTime] = timeFrequency[startTime] + 1
        ELSE
            timeFrequency[startTime] = 1
        END IF
    END FOR

    // Find most common start time
    mostCommonTime = MAX_KEY(timeFrequency BY value)
    RETURN mostCommonTime
END FUNCTION
```

To prevent conflicts between tasks, I will use a conflict detection and resolution algorithm which will be an Interval Tree algorithm where scheduled tasks are stored as intervals in the tree and for each new task insertion: Tree is queried for overlapping intervals, if overlap found, return conflict warning, add to nearest available appropriate slot. This ensures the schedule remains valid and practical, preventing the frustration of double-booked time slots.

Pseudocode:

```
STRUCTURE IntervalNode:
    startTime
    endTime
    task
    max               // Maximum end time in subtree
    left
    right
END STRUCTURE

FUNCTION insertTask(root, newTask):
    // Query for overlapping intervals
    conflicts = queryOverlaps(root, newTask.startTime,
newTask.endTime)

    IF conflicts IS NOT EMPTY THEN
        // Conflict detected - find nearest available slot
```

Timely

```
        RETURN handleConflict(root, newTask, conflicts)
    ELSE
        // No conflict - insert into tree
        root = insertInterval(root, newTask.startTime,
newTask.endTime, newTask)
        RETURN SUCCESS
    END IF
END FUNCTION


FUNCTION queryOverlaps(node, startTime, endTime):  parameters
    IF node IS NULL THEN
        RETURN EMPTY_LIST
    END IF

    overlaps = LIST()

    // Check if current node overlaps
    IF node.startTime < endTime AND startTime < node.endTime THEN
        overlaps.add(node)
    END IF

    // Search left subtree if possible overlap exists
    IF node.left IS NOT NULL AND node.left.max > start_time THEN
        overlaps.addAll(queryOverlaps(node.left, startTime,
endTime))
    END IF

    // Search right subtree
    IF node.right IS NOT NULL THEN
        overlaps.addAll(queryOverlaps(node.right, startTime,
endTime))
    END IF

    RETURN overlaps
END FUNCTION

FUNCTION handleConflict(root, task, conflicts):
    // Issue warning
    DISPLAY "Conflict detected with existing task(s)"

    // Find nearest available slot after conflicts
    latestConflictEnd = MAX(conflicts.map(c => c.endTime))
    task.startTime = latestConflictEnd
    task.endTime = latestConflictEnd + task.length

    // Recursively check new slot
```

Timely

```
    RETURN insertTask(root, task)
END FUNCTION
```

Users should have short, regular breaks to reduce mental fatigue. Since these breaks are short and can be hard to plan, they will be calculated dynamically using a pomodoro-based algorithm that inserts a 10-minute break after each 50-minute segment. By automatically inserting regular breaks after work segments, it promotes sustainable productivity without requiring conscious effort from the user. This ensures the schedule supports long-term wellbeing rather than just short-term task completion.

Pseudocode:

```
FUNCTION insertPomodoroBreaks(schedule):
    WORK_DURATION = 50 // minutes
    BREAK_DURATION = 10 // minutes

    modifiedSchedule = LIST()

    FOR EACH task IN schedule DO
        taskStart = task.start_time
        taskDuration = task.length * 60    //convert hours to
minutes

        // Calculate number of work segments
        numSegments = CEILING(taskDuration / WORK_DURATION)
        currentTime = taskStart

        FOR i = 1 TO numSegments DO
            // Add work segment
            segmentDuration = MIN(WORK_DURATION, taskDuration)
            workSegment = CREATE_SEGMENT(
                type: "WORK",
                task: task,
                start_time: currentTime,
            )
            modifiedSchedule.add(workSegment)

            currentTime = currentTime + segmentDuration
            taskDuration = taskDuration - segmentDuration

            // Add break after segment (unless last segment)
            IF i < numSegments AND taskDuration > 0 THEN
                breakSegment = CREATE_SEGMENT(
                    type: "BREAK",
                    start_time: currentTime,
                )
                modifiedSchedule.add(breakSegment)
                currentTime = currentTime + BREAK_DURATION
            END IF
```

Timely

```
        END FOR
    END FOR

    RETURN modifiedSchedule
END FUNCTION
```

Finally, I will use a SHA-256 hashing algorithm with unique salts to ensure security and robustness. By securely storing passwords using industry-standard cryptographic hashing with salt, it protects user accounts and privacy. This establishes trust and meets modern security requirements.

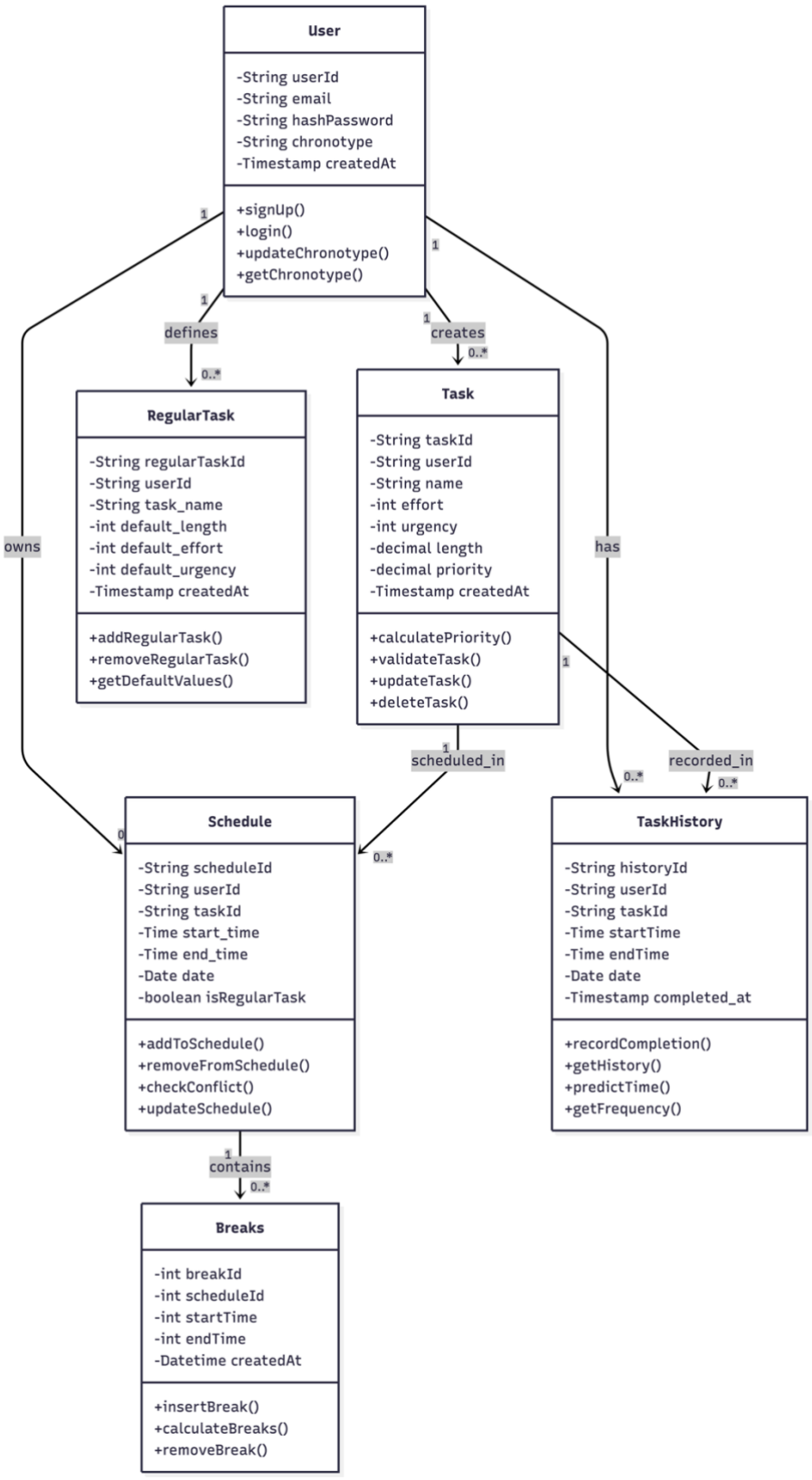
Pseudocode:

```
FUNCTION hashPassword(password):
    // Generate random salt
    salt = generateRandomBytes(16)

    // Combine password with salt
    saltedPassword = password + salt

    // Hash using SHA-256 library function
    hash = SHA256_LIBRARY.hash(saltedPassword)

    // Store both salt and hash
    storedCredential = salt + ":" + hash

    RETURN storedCredential
END FUNCTION

FUNCTION verifyPassword(inputPassword, storedCredential):
    // Split stored credential into salt and hash
    parts = storedCredential.split(":")
    salt = parts[0]
    storedHash = parts[1]

    // Hash input password with the same salt
    saltedInput = inputPassword + salt
    inputHash = SHA256_LIBRARY.hash(saltedInput)

    // Compare the hashes (timing-safe comparison)
    IF secureCompare(inputHash, storedHash) THEN
        RETURN TRUE  // Password matches
    ELSE
        RETURN FALSE  // Password doesn't match
    END IF
END FUNCTION
```

Timely

UML class diagrams

Here are some class diagrams that represent the core structure and relationships within the system, showing how the main classes interact, what data they hold, and which operations they provide:



Timely

Data Tables

Here are some data tables to represent tables in my database and outline the key fields, constraints, and relationships between them. For this project, I will be using Supabase as my cloud database.

Table: User

Field	Key	Data Type	Validation
userId	Primary	varchar	
email	Unique	varchar	include '@', length>5, not null
hashPassword		varchar	not null
chronotype		varchar	'Early' or 'Middle' or 'Late'
createdAt		timestamp	

Table: Task

Field	Key	Data Type	Validation
taskId	primary	varchar	
userId	foreign	varchar	not null
name		varchar	not null
effort		int	1-10
urgency		int	1-10
length		decimal	0-10
priority		decimal	not null
createdAt		timestamp	not null

Table: regularTask

Field	Key	Data Type	Validation
regularTaskId	primary	varchar	
userId	foreign	varchar	not null
task_name		varchar	not null
default_length		int	not null
default_effort		int	not null
default_urgency		int	not null
created_at		timestamp	

Timely

Table: taskHistory

Field	Key	Data Type	Validation
historyId	primary	varchar	
userId	foreign	varchar	not null
taskId	foreign	varchar	
startTime		time	not null
endTime		time	not null
date		date	not null
completed_at		timestamp	0-23

Table: Schedule

Field	Key	Data Type	Validation
scheduleId	Primary	varchar	
userId	foreign	varchar	not null
taskId	foreign	varchar	not null
start_time		time	not null
end_time		time	not null
date		date	not null
isRegularTask		boolean	default false

Table: Breaks

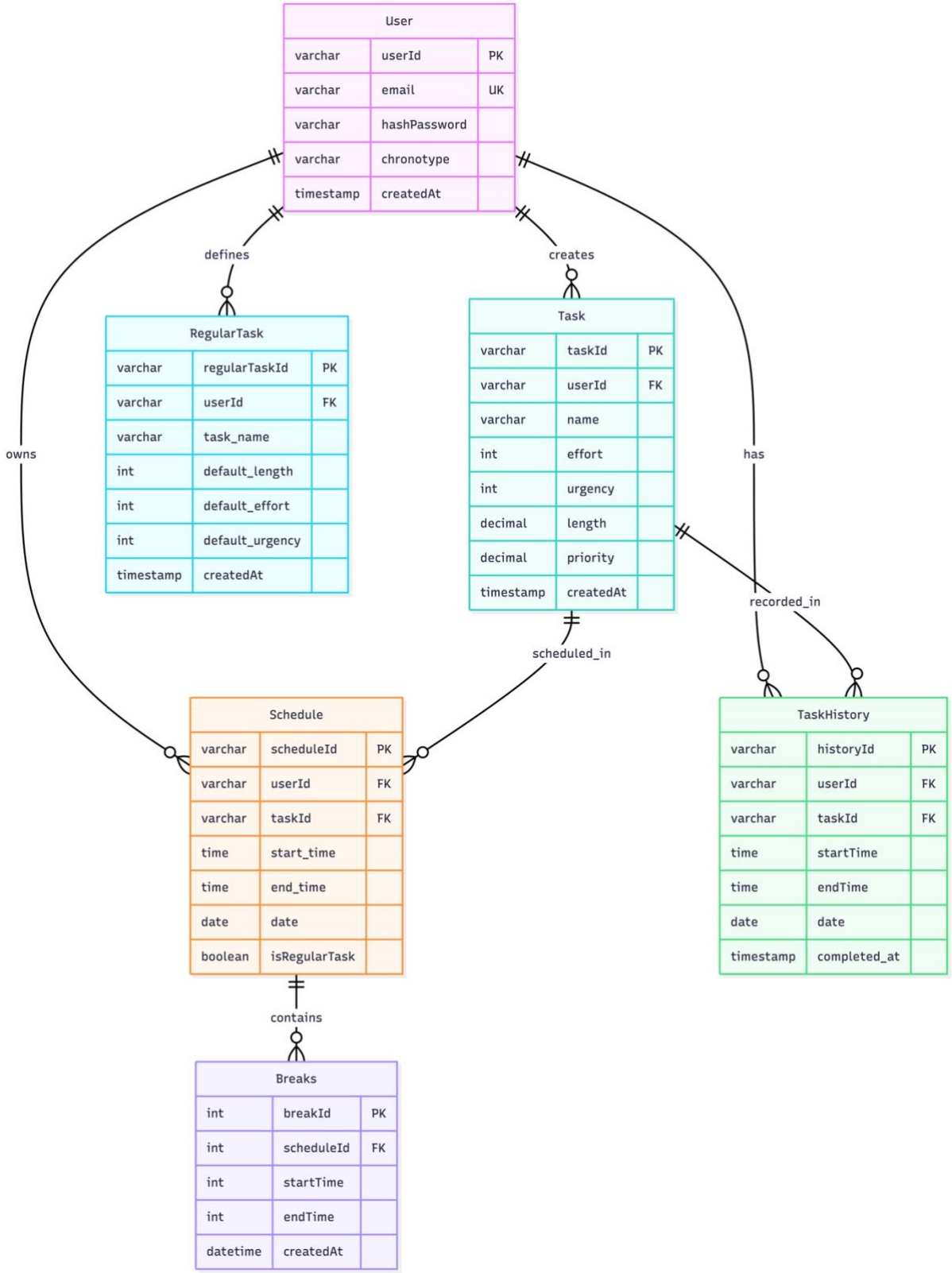
Field	Key	Data Type	Validation
breakId	primary	int	
scheduleId	foreign	int	not null
startTime		int	0-23
endTime		int	0-23
createdAt		datetime	not null

Timely

Database Design

Entity-Relationship Diagram

Here is an ER diagram to display how the system's entities (User, Tasks, Schedule and History Records) interact with each other.



Timely

```
UPDATE tasks
SET task_name = ?, length = ?, effort = ?, urgency = ?, priority = ?
WHERE task_id = ? AND user_id = ?;
```

7. Delete task

```
DELETE FROM tasks
WHERE task_id = ? AND user_id = ?;
```

8. Get task By ID

```
SELECT task_id, task_name, effort, urgency, length, priority
FROM tasks
WHERE task_id = ? AND user_id = ?;
```

9. Insert scheduled task

```
INSERT INTO schedule (schedule_id, user_id, task_id, start_time, end_time, date, is_regular_task)
VALUES (?, ?, ?, ?, ?, ?, ?);
```

10. Check for conflicts

```
SELECT schedule_id, start_time, end_time
FROM schedule
WHERE user_id = ?
  AND date = ?
  AND ((start_time < ? AND end_time > ?) OR (start_time < ? AND end_time > ?));
```

11. Delete scheduled task

```
DELETE FROM schedule
WHERE schedule_id = ? AND user_id = ?;
```

12. Update scheduled task time

```
UPDATE schedule
SET start_time = ?, end_time = ?
WHERE schedule_id = ? AND user_id = ?;
```

13. Insert Regular Task

```
INSERT INTO regular_tasks (regular_task_id, user_id, task_name, duration)
VALUES (?, ?, ?, ?);
```

14. Delete regular task

Timely

```sql
SELECT regular_task_id, task_name, duration
FROM regular_tasks
WHERE user_id = ?
ORDER BY task_name;
```

15. Insert task history record

```sql
INSERT INTO history_records (history_id, user_id, task_id, start_time, end_time, date, completed_at)
VALUES (?, ?, ?, ?, ?, ?, CURRENT_TIMESTAMP);
```

16. Retrieve Task history (last 90 days)

```sql
SELECT h.history_id, t.task_name, h.start_time, h.end_time, h.date, h.completed_at
FROM history_records h
JOIN tasks t ON h.task_id = t.task_id
WHERE h.user_id = ?
  AND h.date >= CURRENT_DATE - INTERVAL '90 days'
ORDER BY h.date DESC, h.start_time DESC
LIMIT 50;
```

17. Delete old history records

```sql
DELETE FROM history_records
WHERE user_id = ?
  AND date < CURRENT_DATE - INTERVAL '90 days';
```

18. Get most common start time for task

```sql
SELECT start_time, COUNT(*) as frequency
FROM history_records h
JOIN tasks t ON h.task_id = t.task_id
WHERE h.user_id = ?
  AND t.task_name = ?
  AND h.date >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY start_time
ORDER BY frequency DESC
LIMIT 1;
```

19. Export user data as CSV

Timely

```sql
SELECT
    t.task_name,
    t.effort,
    t.urgency,
    t.length,
    t.priority,
    s.date,
    s.start_time,
    s.end_time
FROM schedule s
JOIN tasks t ON s.task_id = t.task_id
WHERE s.user_id = ?
ORDER BY s.date, s.start_time;
```

20. Check for duplicate tasks

```sql
SELECT COUNT(*) as duplicate_count
FROM schedule
WHERE user_id = ?
  AND task_id = ?
  AND start_time = ?
  AND date = ?;
```

Timely

Data Structures

Priority Queue

The system will use a priority queue to manage task scheduling efficiently. In my greedy scheduling algorithm, tasks must be processed in order of their calculated priority score (derived from effort, urgency, and length). A priority queue naturally maintains this ordering, ensuring the highest-priority task is always retrieved first. Python's 'heapq' module provides a min-heap implementation that I will adapt for max-priority behavior by negating priority scores during insertion. The key advantage of this structure is its O(log n) insertion and extraction complexity, which is significantly faster than maintaining a sorted list (O(n) insertion). This is crucial for meeting Objective 15a. When a task cannot be scheduled immediately due to unavailable slots, it enters the priority queue waitlist. The queue guarantees that when a slot becomes available, the most important task is selected first, maintaining intelligent scheduling that differentiates my solution from competitors like Lifestack. Implementation will involve creating a wrapper class around 'heapq' that handles priority comparison and timestamp tie-breaking

Interval Tree

An interval tree will be implemented to detect scheduling conflicts with complete accuracy, as required by Objective 13. Each scheduled task represents an interval with a start and end time, and the tree structure allows efficient querying of overlapping intervals. The critical advantage over a linear search through all tasks is the O(log n) query time, where n is the number of scheduled tasks. For a user with 30 tasks scheduled across a week, this reduces conflict detection from 30 comparisons to approximately 5 tree traversals. Each node in the tree stores an interval (start time, end time, task reference) and maintains the maximum end time of its subtree, enabling the algorithm to crop unnecessary branches during overlap queries. The tree structure supports dynamic insertion as users add tasks throughout the day, and the self-balancing properties prevent degradation to O(n) worst-case performance. This is essential for meeting Objective 15b because every task addition, adjustment, or removal triggers conflict detection. The interval tree integrates with the greedy scheduling algorithm by verifying slot availability before assignment. This implementation guarantees zero overlapping tasks (Objective 13b) while maintaining fast performance, even as the schedule fills with tasks throughout the week. The tree structure is suited to the dynamic nature of user scheduling and provides better reusability than a simple sorted list, as the same tree can handle queries for any time range without restructuring.

Hash Table

Hash tables will serve two critical functions: user authentication lookups and rapid task retrieval by unique identifier. Python's built-in 'dict' type implements a hash table with O(1) average-case lookup, insertion, and deletion. For authentication, the system stores user credentials with the email as the key, enabling instant credential verification without iterating through a user database. This directly supports Objective 3c (database queries in under 2 seconds) by eliminating linear searches. Additionally, when the timetable UI needs to display or update a specific task, a hash table mapping task IDs to task objects provides immediate access without traversing lists or trees. The speed advantage is substantial: for a

database with 1,000 users, hash table lookup remains O(1) while a list search averages 500 comparisons.

Stacks

A stack-based undo/redo system will enhance usability by allowing users to reverse scheduling mistakes. The (LIFO) structure models the undo operation: the most recent action should be reversed first. Two stacks will maintain the system state, an undo stack storing previous schedule snapshots and a redo stack for forward navigation. When a user performs an action, such as adding, removing, or adjusting a task, the current schedule state is pushed onto the undo stack. Pressing undo pops this state and pushes it to the redo stack, restoring the previous version. This design prevents the common user frustration identified in my competition analysis of Lifestack, where irreversible changes force users to manually reconstruct their schedules. Python's list type provides native stack operations (‘append()’ for push, ‘pop()’ for pop) with O(1) complexity, making implementation straightforward.

Records

Records structure the system's core entities: User, Task, Schedule, and HistoryRecord. Python's class system provides a natural way to define records with named attributes, methods, and relationships, which improve code readability and maintainability compared to dictionaries or tuples. Each record type encapsulates related data and behaviour, following the UML class diagrams on page 24. The Task class, for example, bundles name, effort, urgency, length, and calculated priority, along with methods for priority calculation and validation.

Pseudocode example implementation:

```
CLASS Task
    FUNCTION __init__(task_id, name, effort, urgency, length)
        self.task_id = task_id
        self.name = VALIDATE_NAME(name)
        self.effort = VALIDATE_SCALE(effort, "effort")
        self.urgency = VALIDATE_SCALE(urgency, "urgency")
        self.length = VALIDATE_LENGTH(length)
        self.priority = CALCULATE_PRIORITY()

    FUNCTION CALCULATE_PRIORITY()
        w1 = 0.3    // weight for effort
        w2 = 0.5    // weight for urgency
        w3 = 0.2    // weight for length
        RETURN (self.effort * w1) + (self.urgency * w2) +
(self.length * w3)
    END FUNCTION


    FUNCTION VALIDATE_NAME(name)
        IF length of name < 1 OR length of name > 100 THEN
            THROW error "Task name must be 1-100 characters"
        END IF
```

```
        RETURN name
    END FUNCTION


    FUNCTION VALIDATE_SCALE(value, field)
        IF value < 1 OR value > 10 THEN
            THROW error field + " must be between 1-10"
        END IF
        RETURN value
    END FUNCTION


    FUNCTION VALIDATE_LENGTH(length)
        IF length <= 0 THEN
            THROW error "Length must be positive number"
        END IF
        RETURN length
    END FUNCTION
END CLASS
```

This record structure implements Objective 14 (task input validation) by embedding validation logic within the Task class itself, ensuring data integrity at creation. Furthermore, records provide reusability through inheritance. For example, a RegularTask subclass extends Task with recurrence properties and simplifies database interactions by mapping directly to the database tables defined on pages 25-27.

File Structure

Overview

A well-organised file structure is essential for managing code complexity and ensuring the project remains maintainable throughout development. For this project, I need a structure that clearly separates different concerns - algorithms, database operations, user interface, and shared utilities while making it straightforward to locate specific functionality when needed.

Design Considerations

When planning the structure, several factors influenced my decisions:

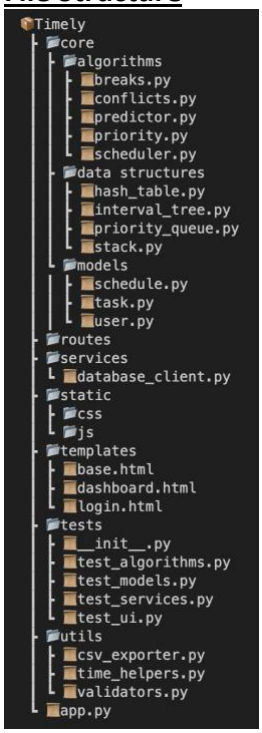
Clarity - Each file should have a clear purpose that's obvious from its name and location. When I need to modify the greedy scheduling algorithm or fix a bug in the task creation dialog, I should know exactly where to look without searching through multiple folders.

Separation of concerns - The scheduling algorithms shouldn't depend on whether I'm using Supabase or a different database. Similarly, the UI components shouldn't contain other logic. This separation makes testing individual components much simpler and allows me to modify one part without breaking others.

Mapping to objectives - My 17 objectives require specific implementations that should have obvious locations in the structure. For example, Objective 13 (conflict detection) clearly belongs with the interval tree implementation, while Objective 17 (CSV export) needs a

dedicated utility file. The structure should make these connections clear for both development and documentation purposes.
Balance - The structure needs enough organization to be professional and maintainable, but not so many nested folders that navigation becomes tedious. I've seen projects with excessive folder hierarchies that make simple tasks frustrating.
Chosen Structure
I've organized the project into four main modules based on their technical responsibilities:
core/ - This contains all the main logic, algorithms, and data structures. The algorithms subfolder houses the priority calculator, greedy scheduler, time predictor, conflict detector, and break insertion logic. The data structures subfolder implements the priority queue, interval tree, hash table, and undo stack. The models subfolder defines the User, Task, and Schedule classes from my UML diagrams. This module is self-contained and doesn't depend on external services.
services/ - This module handles all communication with Supabase. It includes the client initialisation, authentication operations, and separate service files for users and tasks. Isolating database operations here means that database queries for Objective 3 are centralised and easy to optimize or test.
ui/ - All user interface components live here, separated into windows and dialogs. The main window displays the timetable, while the dialogs handle specific operations like adding tasks (Objective 8), adjusting tasks (Objective 9), and managing regular tasks (Objectives 6-7). This separation keeps the presentation layer distinct from the main logic.
utils/ - Shared functionality that multiple modules need, including input validators (Objective 14), time manipulation helpers, and the CSV export utility (Objective 17). These are reusable.

File Structure



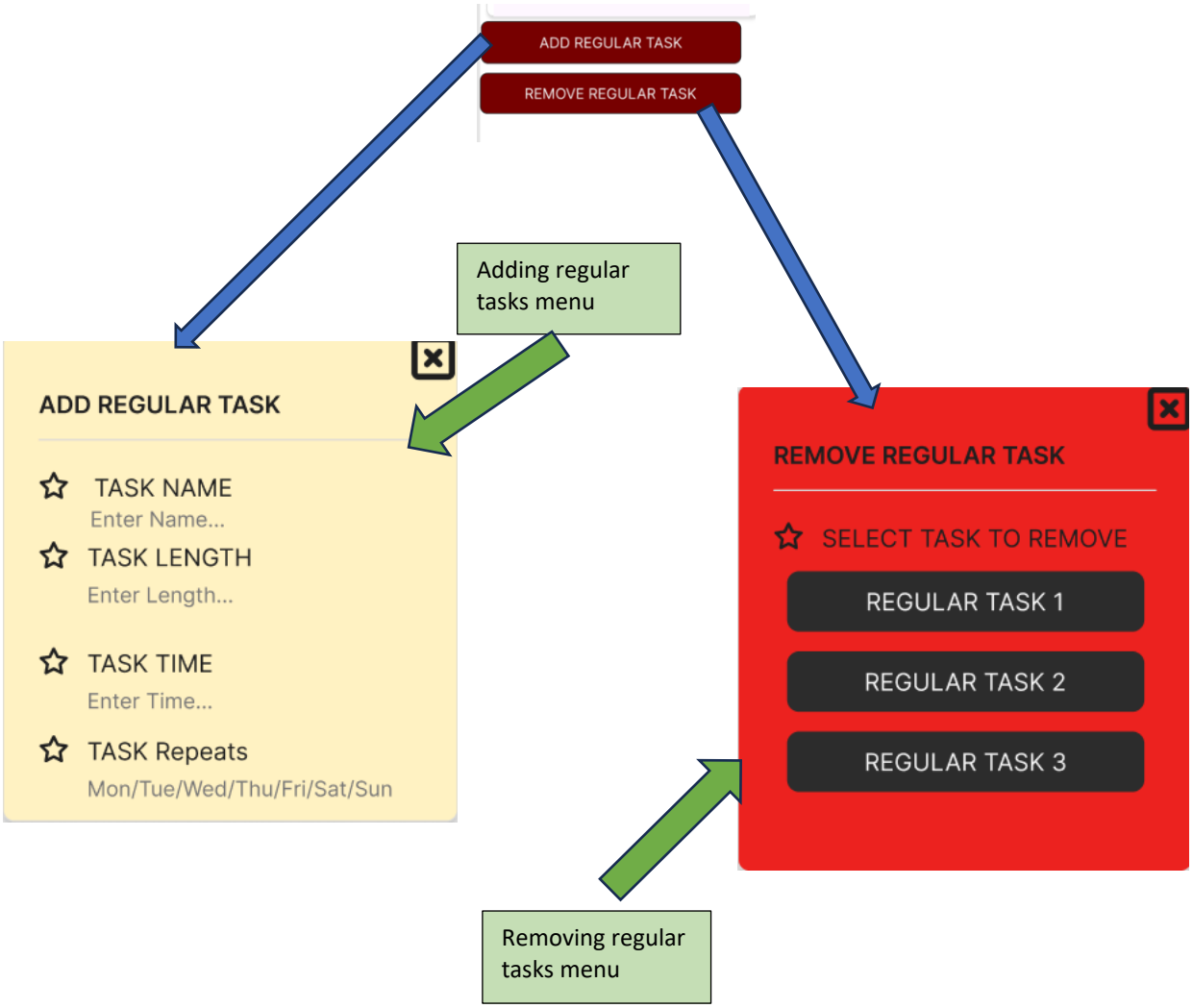
Timely

This organisation supports several of the objectives. The testing structure maps clearly to validation requirements in Objectives 1, 3c, 13b, and 14. The services layer centralizes database operations for Objective 3, while the algorithms folder makes the implementations for Objectives 4 and 11 easy to locate and document.

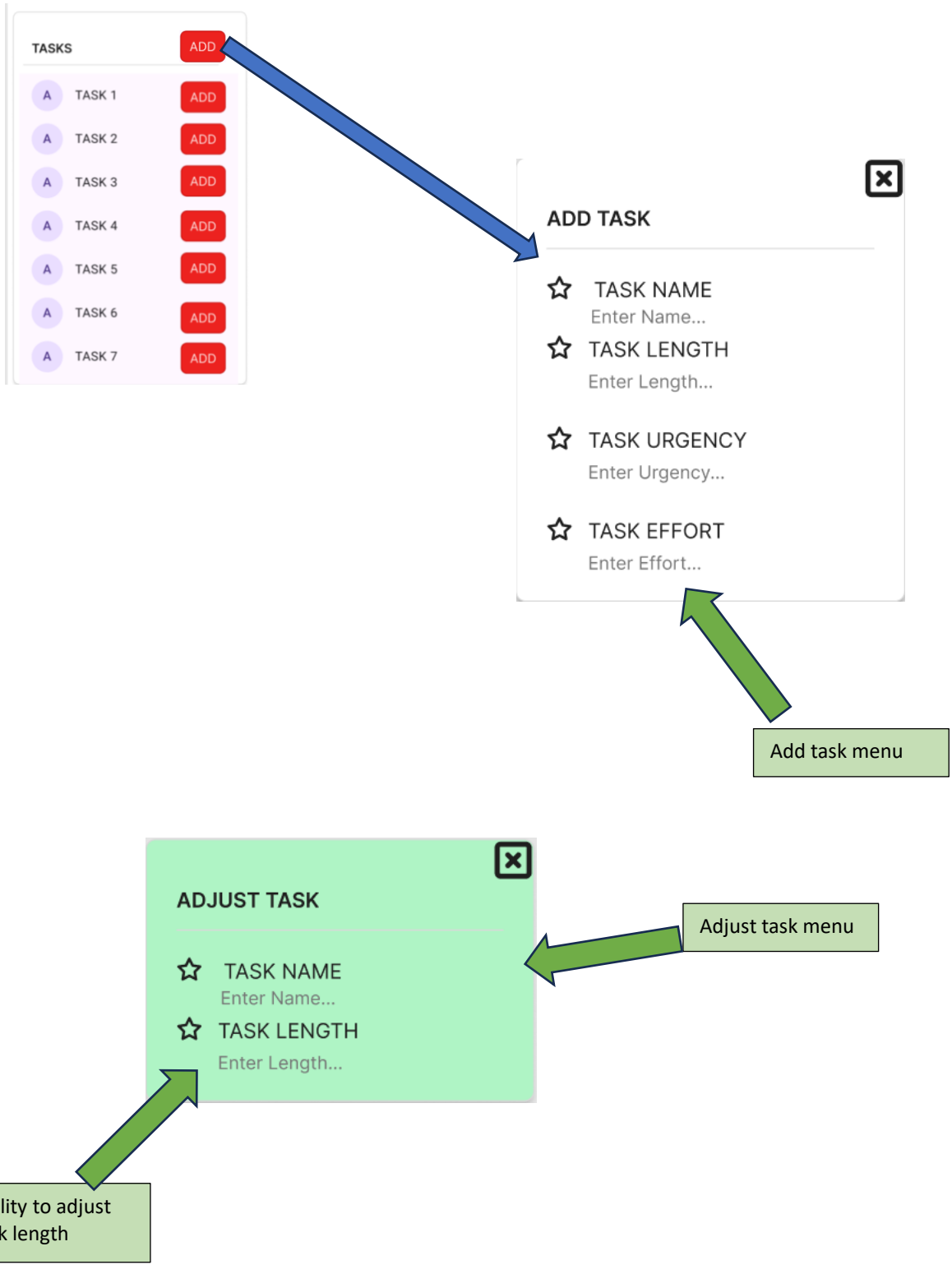
Screen Designs

The following screenshots show early interface prototypes. They outline the planned functionality in detail and provide a clear representation of the intended user interface layout and behaviour:

Timely


ADD REGULAR TASK
REMOVE REGULAR TASK
Adding regular tasks menu
ADD REGULAR TASK
TASK NAME
Enter Name...
TASK LENGTH
Enter Length...
TASK TIME
Enter Time...
TASK Repeats
Mon/Tue/Wed/Thu/Fri/Sat/Sun
REMOVE REGULAR TASK
SELECT TASK TO REMOVE
REGULAR TASK 1
REGULAR TASK 2
REGULAR TASK 3
Removing regular tasks menu

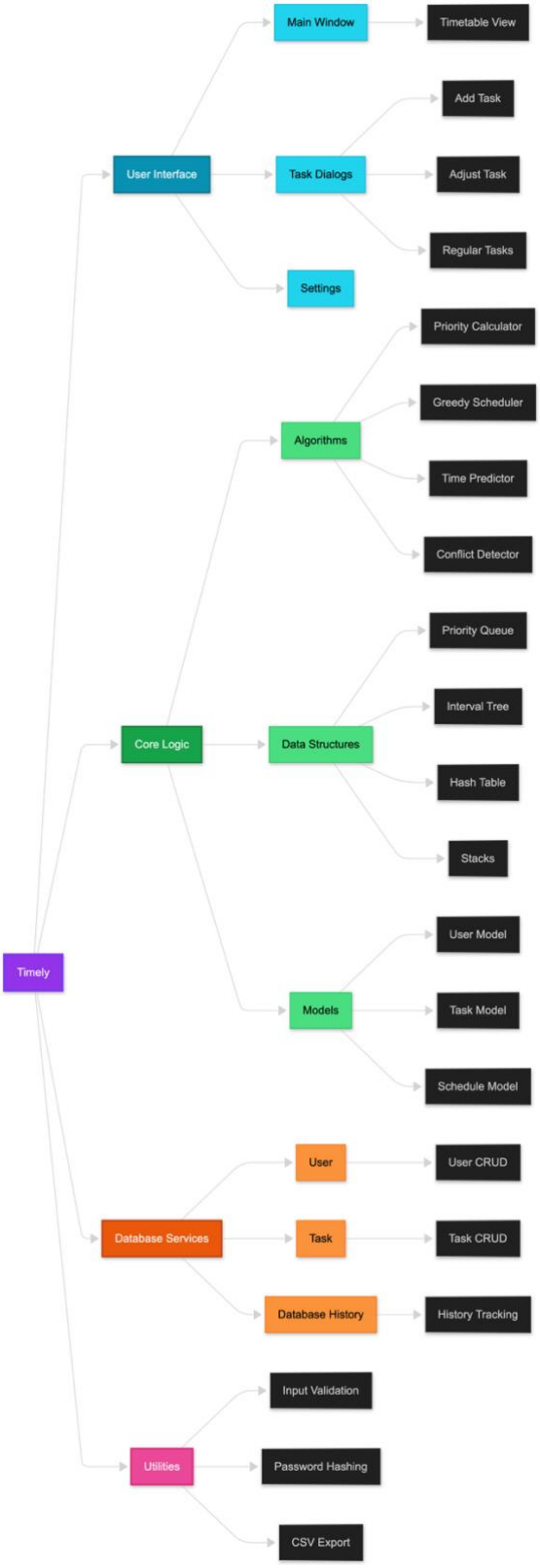
Timely


TASKS
ADD
TASK 1
TASK 2
TASK 3
TASK 4
TASK 5
TASK 6
TASK 7
ADD TASK
TASK NAME
Enter Name...
TASK LENGTH
Enter Length...
TASK URGENCY
Enter Urgency...
TASK EFFORT
Enter Effort...
Add task menu
ADJUST TASK
TASK NAME
Enter Name...
TASK LENGTH
Enter Length...
Adjust task menu
Ability to adjust task length

Timely

Structure Diagram

Due to the project's complexity, I created this hierarchy chart to illustrate the overall structure and flow:



Timely

Technical Solution

Testing

Evaluation

Bibliography

- Gaggero, A. A. & Tommasi, D. (2023) *Time of day, cognitive tasks and efficiency gains*. The Economic Journal, 133(654), pp. 2545–2572.
- Reilly, T., Atkinson, G. & Waterhouse, J. (1998) *Circadian variation in human performance*. Chronobiology International, 15(6), pp. 563–583.
- Yoon, S., Kim, J., Lee, H., Park, H. & Son, J. (2024) *Chronotype, work ability, and productivity loss: Findings from the Korean Work, Sleep, and Health Study*. Chronobiology International, 41(3), pp. 407–416.