Assignment Tracking System

12B 09 Lau Ka Yue / 25-26 HKDSE ICT SBA

Table of Contents

School-based Assignment (SBA) . . . 1
Project Mission . . . 2
Web Application Development . . . 3
- The Core Stack . . . 3
- Programming and Markup Languages . . . 4
- Website Map . . . 6
- Project Structure . . . 6
- Test Users . . . 7
- Main Features . . . 8
- Other Highlights . . . 25

Relational Database Design . . . 27
- Database Schema . . . 27
- ER Diagram . . . 29
- Normalisation . . . 30

SQL Implementation . . . 32
Testing and Evaluation . . . 38
- Error Logging . . . 38
- Access Control . . . 38
- Data Validation . . . 39
- Data Verification . . . 43
- Data Integrity Constraints . . . 44
- Making a Major Design Change . . . 45
- Potential Improvements to the System . . . 46

Conclusion . . . 47

School-based Assignment (SBA)

The School-Based Assignment (SBA) accounts for 20% of the total subject mark in the ICT curriculum. It consists of two guided tasks that focus on the development of an information system. These tasks are centred around two key stages of system development:

- **Task 1: Design and Implementation (25 marks)**
- **Task 2: Testing and Evaluation (15 marks)**

Students are required to complete Task 1 in Grade 11 and Task 2 in Grade 12. Both Task 1 and Task 2 marks are to be submitted in one go in Grade 12.

The SBA in ICT is designed with the following objectives:

1. **Enhancing Assessment Validity**: SBA evaluates a wider range of learning dimensions, including:
    a. Knowledge and understanding
    b. Generic skills
    c. Practical skills
2. **Reducing Dependence on Public Examinations**:
    a. Public exams may not always accurately reflect a student's abilities
    b. SBA provides a complementary approach to evaluate students more holistically
3. **Promoting Positive "Backwash Effect"**:
    a. SBA encourages meaningful learning activities
    b. It motivates students, teachers, and school staff to focus on practical and real-world ICT applications.

This SBA combines components from **2A: Databases** and **2B: Web Application Development**.

(Written referencing official HKDSE ICT SBA guidelines)

Project Mission

Currently, ABC College uses a paper-based system to manage assignment submission records. This process is time-consuming, prone to errors, and makes it difficult for teachers to access, analyse, and manage data efficiently.

The **Assignment Tracking System (ATS)** aims to streamline the process of managing assignment records.

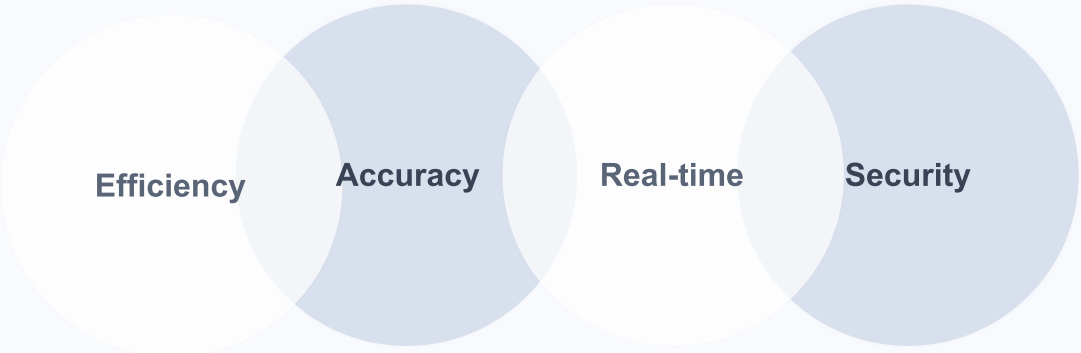
Requirements:

1. **Teachers** should be able to create assignments, view submission records, and generate reports and statistics.
2. **Subject Monitors** should be able to update assignment submission statuses for their assigned classes.
3. **Administrators** should be able to manage users and monitor the system.
4. **Students** should be able to view their own assignment records and deadlines.

The new system must address the drawbacks of the current process and provide the following improvements:

- **Efficiency**: Faster data entry and retrieval through a digital interface
- **Accuracy:** Reduction of human errors with validation features
- **Real-time Access:** Teachers and instantly view and analyse submission records
- **Enhanced Security:** Role-based access control ensures data privacy

(Written referencing school ICT SBA guidelines)



Web Application Development

We now live in a world where computers have become ubiquitous. However, schools are still tracking assignments by paper. Frustrated with the status quo, we aspired to build something better. Something that students and teachers would actually use efficiently and accurately.

What started as a simple assignment tracker, has since evolved into a powerful subject, assignment, and user tracking system that streamlines workflows across the entire homework handling process. The tool is far from perfect, but we are heading there.

The Core Stack

Next.JS	React	TailwindCSS	Neon
(v16.0.0)	(v19.0.0)	(v4.0.0)	(v1.0.0)

(Note: Server-side scripting is performed in Server Components.)

JS Library: React

React is a library for web and native user interfaces. It enables developers to build user interfaces out of components, which are essentially JavaScript functions that is then compiled into HTML code. State variables allows components to change as a result of an interaction without requiring full page reloads, providing a smooth user experience.

https://react.dev/

Full-Stack Framework: Next.JS

Next.js is a React framework for building full-stack web applications. It offers a comprehensive set of features with minimal configuration. It also allows developers to write both server-side and client-side code in Javascript, giving more control and a sense of cohesion. Next.js is maintained by Vercel, a cloud platform specializing in front-end infrastructure and deployment.

Developing with an existing full-stack framework is highly recommended for React builds. Having built two applications in Next.JS, I intend to continue working with this framework to avoid context switching and friction in the development pipeline.

https://nextjs.org/

CSS Framework: TailwindCSS

Tailwind CSS is a CSS framework that allows developers to style their websites directly within HTML using utility classes. Unlike traditional CSS, Tailwind CSS promotes rapid development by eliminating the need to write custom styles for every component. It also offers extensive control over the design.

https://tailwindcss.com/

Database: Neon

Neon is a fully managed serverless, cloud-native PostgreSQL database. It is also the recommended database for Vercel applications.

ATS utilises the Neon serverless driver which is a Postgres driver for JavaScript, allowing for queries from serverless and edge environments over HTTP or WebSockets. In ATS, the driver is used over HTTP.

https://neon.com/

Deployment: Vercel

Vercel is a hosting site that simplifies the deployment process for web applications. As such, ATS can be presented as an actual web application with a functional URL: assignment-tracking-system-v0.vercel.app.

Text Editor: Visual Studio Code

Visual Studio Code is a free source-code editor from Microsoft. It provides support for debugging, syntax highlighting, intelligent code completion, code refactoring, and embedded Git.

https://code.visualstudio.com/

Designing Software: Figma

Figma is a popular cloud-based design and prototyping tool that is widely used in the UI/UX design field.

ATS was designed in Figma without AI assistance.

https://www.figma.com/

Version Control: GitHub

GitHub is a web-based platform that uses Git, a version control system, to help developers manage and track changes in their code.

Programming and Markup Languages

HTML

HTML is the standard markup language for documents designed to be displayed in a web browser. It defines the content and structure of web content.

CSS

CSS is a style sheet language used for specifying the presentation and styling of a document. CSS is designed to enable the separation of content and presentation, including layout, colours, and fonts.

TailwindCSS is a CSS framework.

JavaScript

JavaScript is a programming language used for building interactive web applications, supporting both client-side and server-side development.

JSX is a syntax extension to JavaScript that allows developers to write HTML-like markup inside a JavaScript file. Most React developers use JSX to write components.

SQL

SQL is a query language used to manage data in a relational database management system, allowing users to query, manipulate and create data.

SQL is the primary language used to interact with PostgreSQL databases.

Packages and extensions

ATS utilises an array of packages and extensions to assist the development process and offer additional functionalities.

@heroicons/react

Heroicons are a set of free MIT-licensed high-quality SVG icons by the makers of Tailwind CSS. It allows icons to be imported individually as a React component without having to upload every single icon.

bcrypt

Bcrypt is a library that hashes passwords. JavaScript does not have a built-in function for hashing password, unlike PHP that has password_hash(). Although it can be achieved with 'crypto', the recommended approach is usually to use the bcrypt library.

Hashing is the process of converting some information into a key using a function. The original information cannot be retrieved from the hash key by any means.

clsx

clsx is a utility for constructing className strings conditionally.

PapaParse

PapaParse is a CSV parser that converts between CSV and JSON, used to allow administrators to create users by CSV.

Recharts

Recharts is a composable charting library built on React components.

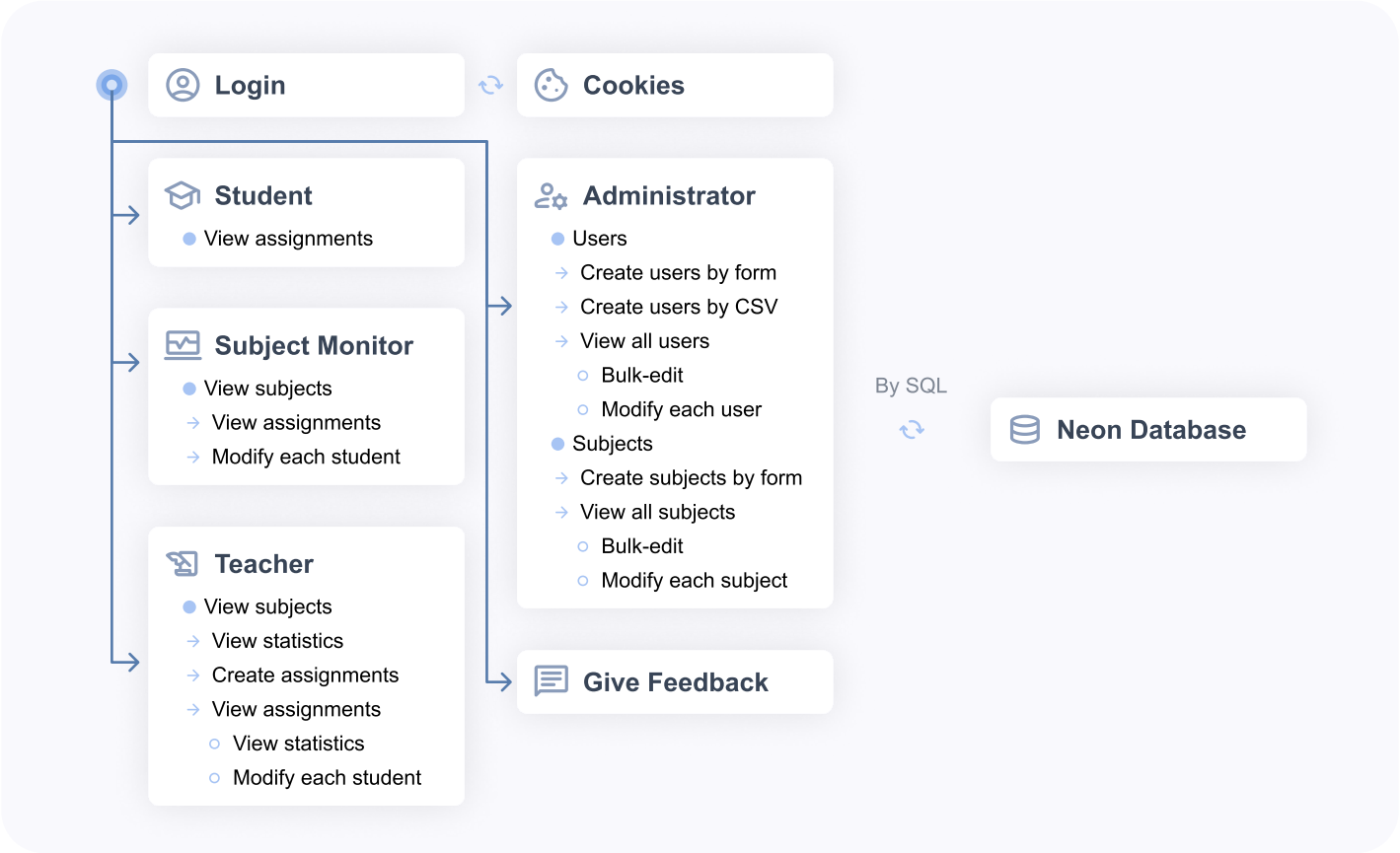
Prettier

Prettier is a code formatter that can be installed as an extension in VS code. It enforces a consistent style by parsing code and re-printing it with a set of formatting rules that take the maximum line length into account, wrapping code when necessary.

(Descriptions written referencing official websites, www.geeksforgeeks.org and en.wikipedia.org)

Website Map

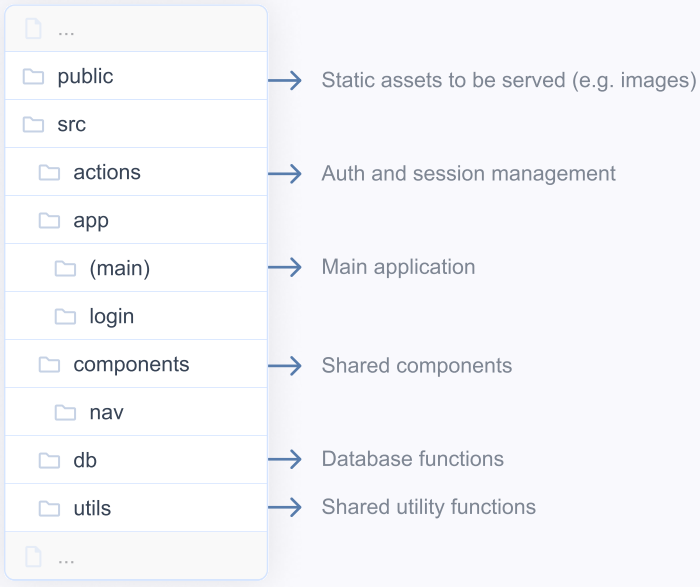
Based on functionality, the web pages are grouped into four role-based categories and two utilities groups: **Student**, **Subject Monitor**, **Teacher**, **Administrator**, as well as **Login** and **Feedback**. They manipulate and fetch from **Cookies** and **Neon Database** through server components.



(Simplified website map designed in Figma)

Project Structure

ATS follows the folder and file conventions of Next.js. Check out the source code here.



In Next.js, folders define URL segments. Nesting folders nests segments. [segment] are dynamic routes, where values can be accessed via the params prop. Code can be organised without changing URLs with router groups (group).

For details, refer to the Next.js documentation on project structure: https://nextjs.org/docs/app/getting-started/project-structure.

Test Users

To facilitate testing, test records were inserted into the database.

Test Users

Alice Johnson
#000000001
Password: “admin”
Role: admin

Michael Chen
#000000011
Password: “admin”
Role: admin

Emily Brown
#000000002
Password: “1234”
Role: teacher
Taught Subjects:
2526-12b-math
2526-g12-bio-1
2526-g12-ict-2

James Rodriguez
#000000012
Password: “1234”
Role: teacher
Taught Subjects:
2425-11b-eng
2526-g12-chem-1
2526-g12-phy-1

Jacky Chu
#000000022
Password: “1234”
Role: teacher
Taught Subjects:
2526-g11-ict-3

Noah Davis
#201401000
Password: “1234”
Role: student
Enrolled Subjects:
2526-12b-math

Olivia Garcia
#201401046
Password: “1234”
Role: student
Enrolled Subjects:
2526-12b-math
2526-g11-ict-3
2526-g12-chem-1
2526-g12-ict-2
2526-g12-phy-1

Emma Wilson
#201401111
Password: “1234”
Role: student
Enrolled Subjects:
2425-11b-eng
2526-g12-bio-1
2526-g12-chem-1
2526-g12-ict-2
2526-g12-phy-1

Assignments

Hand in Elective A Workbook for homework inspection #1
Submit workbook during lesson. Check that all corrections are completed.
Subject Id: 2526-g12-ict-2
Due Date: 2025-09-01 00:00:00+00
Full Grade: 40

Hand in Core B Workbook #4
Submit workbook during lesson.
Subject Id: 2526-g12-ict-2
Due Date: 2025-11-18 00:00:00+00
Full Grade: 50

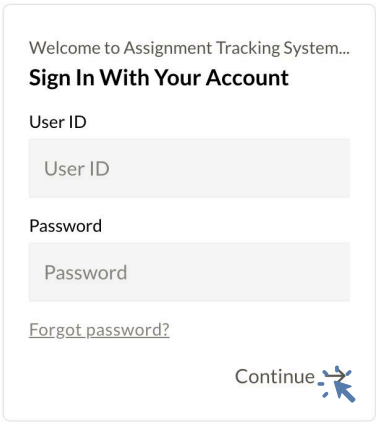
Quadratic Equations Worksheet #2
Solve problems 1-25 from chapter 4.
Subject Id: 2526-12b-math
Due Date: 2025-10-02 00:00:00+00
Full Grade: 12

Organic Chemistry Nomenclature #3
Name 30 compounds + reactions.
Subject Id: 2526-g12-chem-1
Due Date: 2025-12-03 00:00:00+00
Full Grade: 40

Main Features

The following section explains the main features of ATS. Follow along with the website: assignment-tracking-system-v0.vercel.app.

1. Login



Authentication

The user enters the account id and password. Characters within the password field are masked. Upon submission, the form data is passed to an authentication function. The password provided by the user (password) is compared with the hashed password in the database (userData.password).

ATS - Authentication

```
const isMatch = await bcrypt.compare(password, userData.password)
```

(Code image built with ray.so)

Bcrypt is a library that allows algorithms to hash passwords. In PHP, the user-input password would be hashed through password_hash(), then compared with the stored password.

User sessions

A cookie session storing the user's id is created upon successful authentication. Users are automatically logged in within 7 days of initial attempt. Afterwards, the cookie expires and the user must enter their credentials again. All components across the application can obtain the current user id through verifySession() . It also blocks users from accessing unauthorised functions.

ATS - Cookies

```
export async function createSession(userId) {
  const expirationDate = new Date()
  expirationDate.setDate(expirationDate.getDate() + 7)

  const cookieStore = await cookies()

  cookieStore.set('session', userId, {
    httpOnly: true,
    secure: true,
    expires: expirationDate,
    sameSite: 'lax',
    path: '/',
  })
}
```

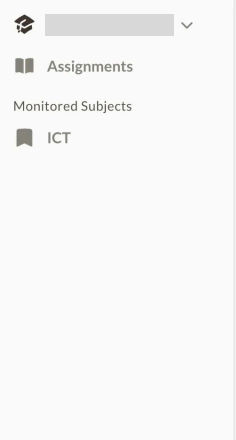
ATS - Cookies

```
// auth.js

export async function verifySession() {
  const sessionUserId = (await cookies()).get('session')?.value

  if (!sessionUserId) {
    redirect('/login')
  }

  return { isAuth: true, userId: sessionUserId }
}

// createUser.js

export async function createUser() {
  const session = await verifySession()
  if (!session) return null

  const userId = session.userId

  // ...
}
```

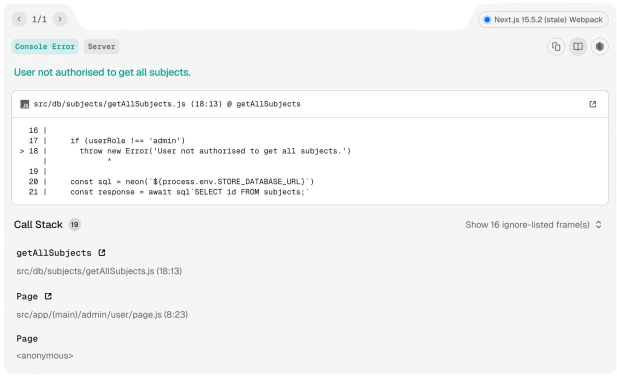
Authorisation

The user is redirected to different web pages according to their role. The nav bar also updates accordingly.

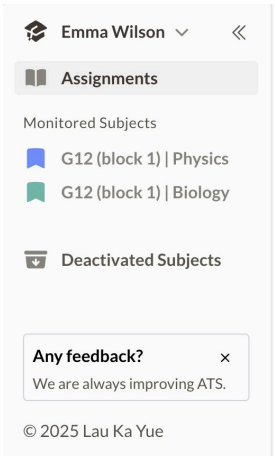
Users cannot access unauthorised web pages simply by entering the url. For example, when a student visits /admin/user , the web pages returns an error.

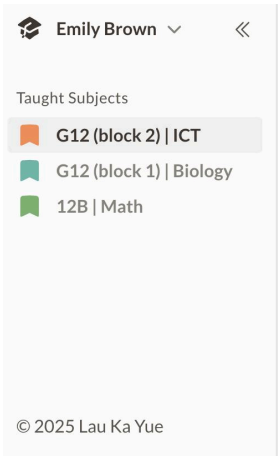


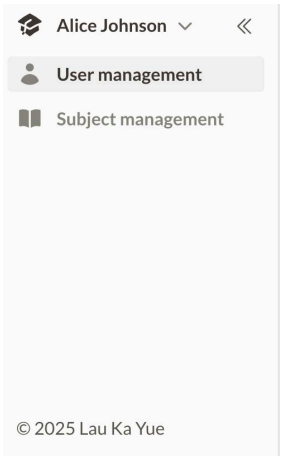
You do not have the rights to access this page.



2. Navigation bar

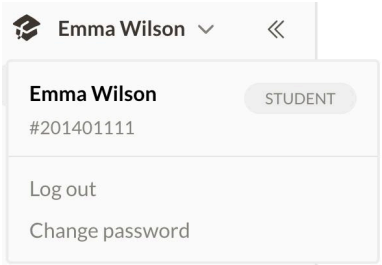






Personal information

When pressed, the top of the navigation bar allows users to view their personal information, including their name, id, and role.



Logging out

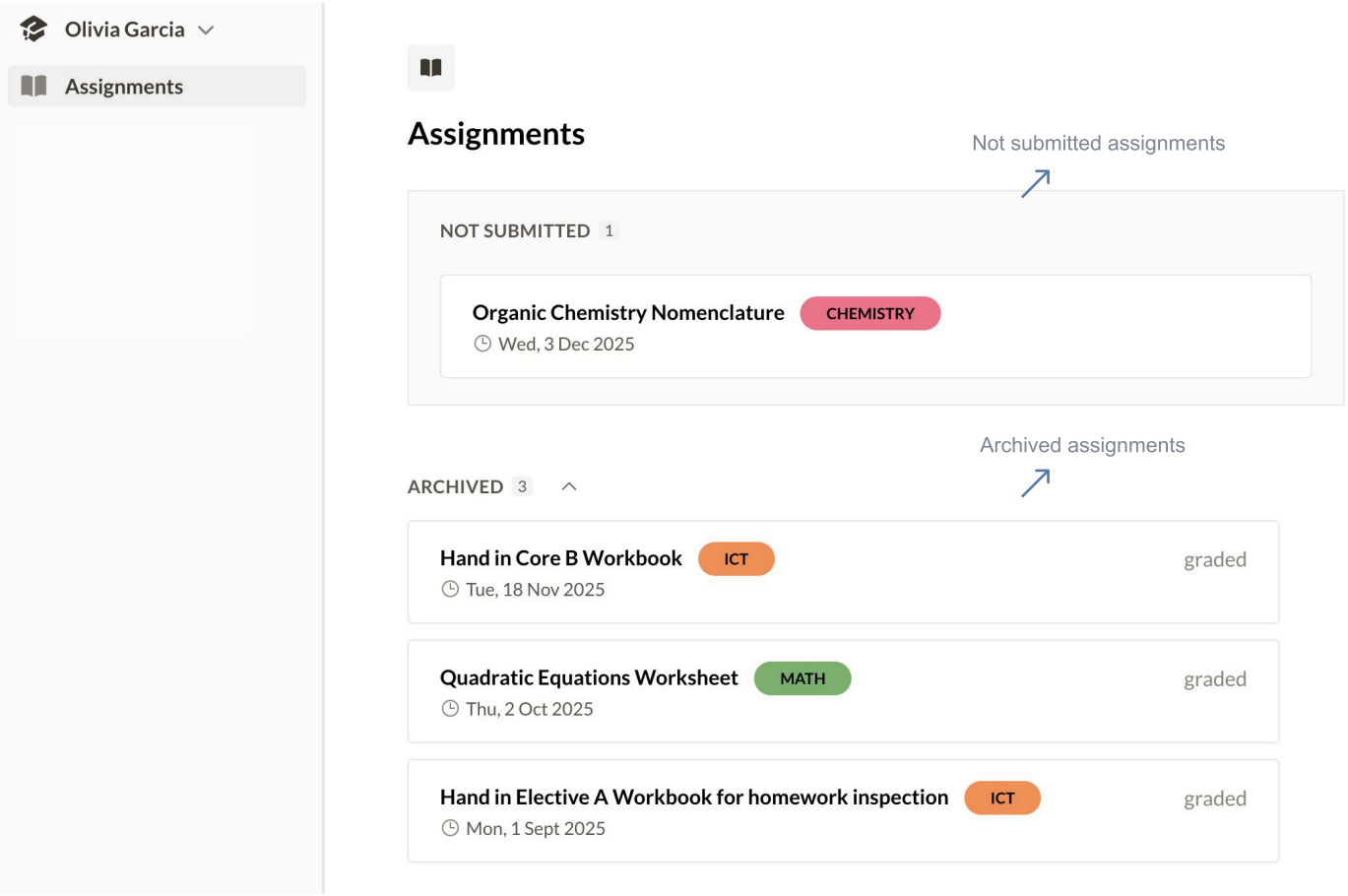
The top of the nav-bar also includes an action menu where users can log out of their accounts, which essentially deletes the session and redirects the user to the login page.

ATS - Logging out

```
// @/components/nav/UserDisplay.jsx

import { deleteSession } from '@/actions/userSession'
import { redirect } from 'next/navigation'

async function logOut() {
  await deleteSession();
  redirect("/")
}

// @/actions/userSession.js

export async function deleteSession() {
  const cookieStore = await cookies()
  cookieStore.delete('session')
}
```

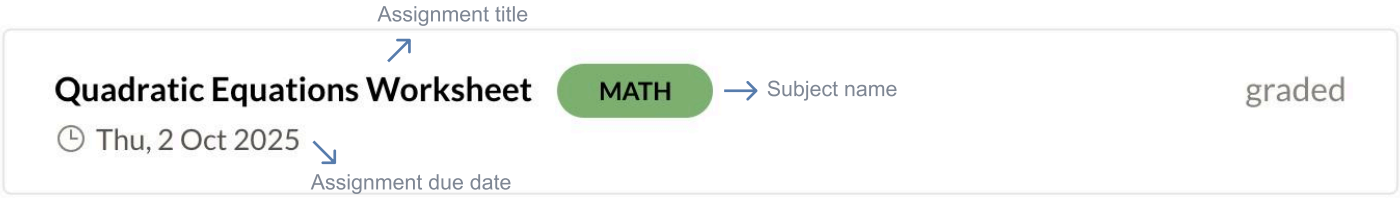
3. Student View

All students, including subject monitors, can access the **Student View**.



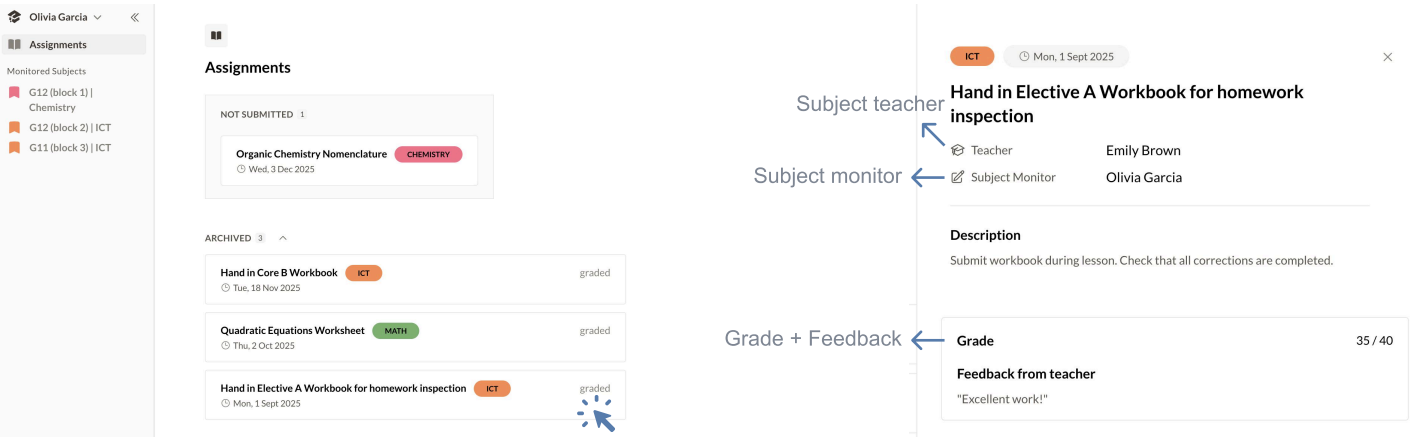
Assignments

Students can see their assignments categorised into three groups: not submitted, submitted and archived. They can see if the assignment has been graded or not, as well as other details at a glance.



Assignment details

When an assignment is selected, its details are shown through a side window.



useState hook

The model can display the details of different assignments without the user refreshing the page. This is only possible using the useState hook from React.

useState is called at the top level of a component to declare a state variable. It returns an array with two values: the current state, and the set function that allows users to update the state to a different value and trigger a re-render.

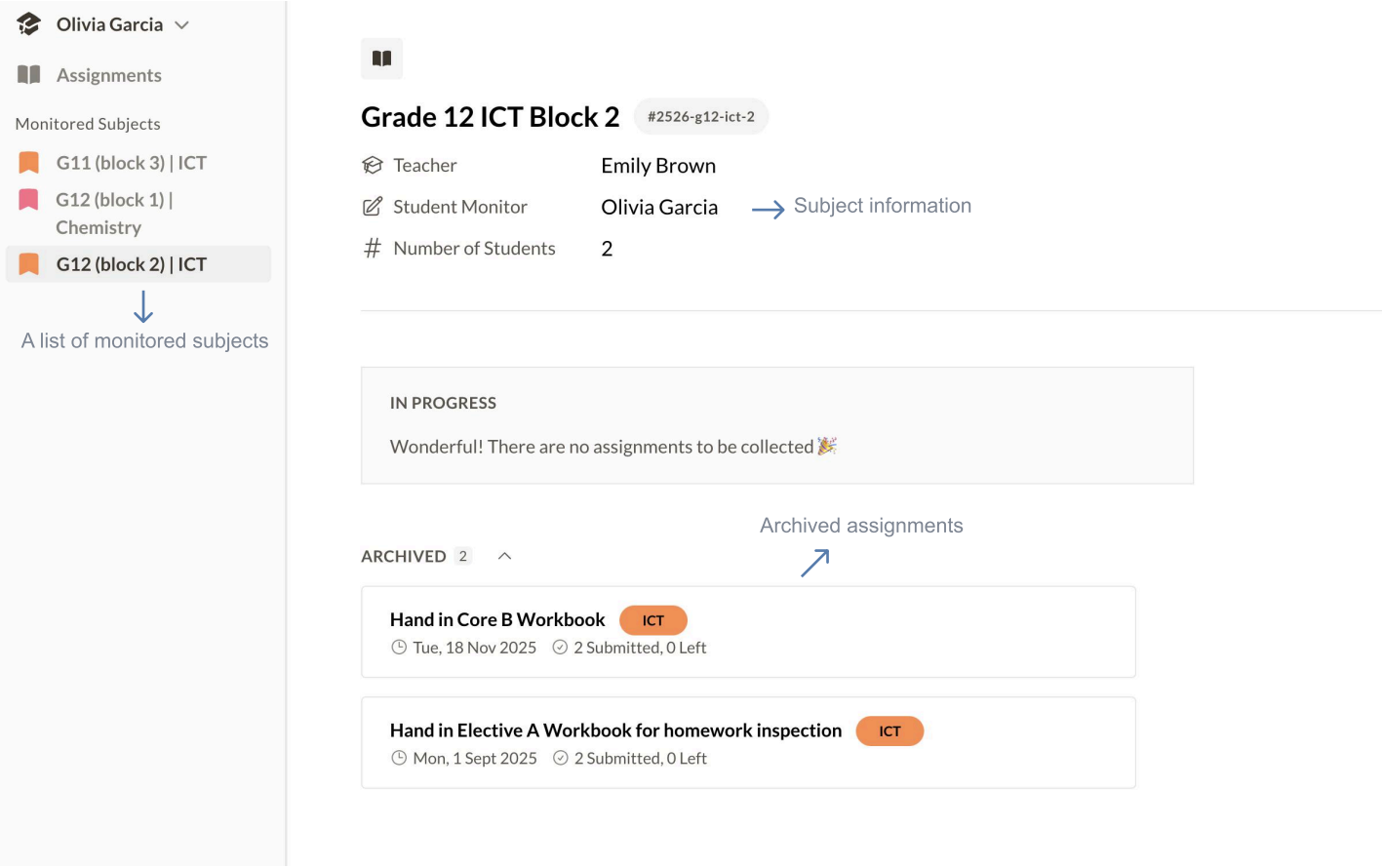
On the right, the set function is passed to AssignmentItem component as a prop, which is called when an assignment is selected. This sets isOpened to true and shows the AssignmentModel, supplied with the correct assignment details.

ATS - Assignment Model

```
// @/(main)/assignments/page.js
import { useState } from 'react'
import AssignmentModel from './AssignmentModel'

const [assignmentModel, setAssignmentModel] = useState({
  isOpened: false,
  assignmentId: '',
})

return (
  // ...
  {assignmentModel.isOpened && (
    <AssignmentModel
      assignment={
        assignments.filter(
          (a) => a.assignment_id === assignmentModel.assignmentId
        )[0]
      }
      onClose={() =>
        setAssignmentModel({ isOpened: false, assignment: [] })
      }
    />
  )}
)

// ./AssignmentItem.jsx

export default function AssignmentItem({a, setAssignmentModel }) {
  return (
    // ...
    <button
      onClick={() =>
        setAssignmentModel({
          isOpened: true,
          assignmentId: a.assignment_id
        })
      }
    >
    </button>
    // ...
  )
}
```

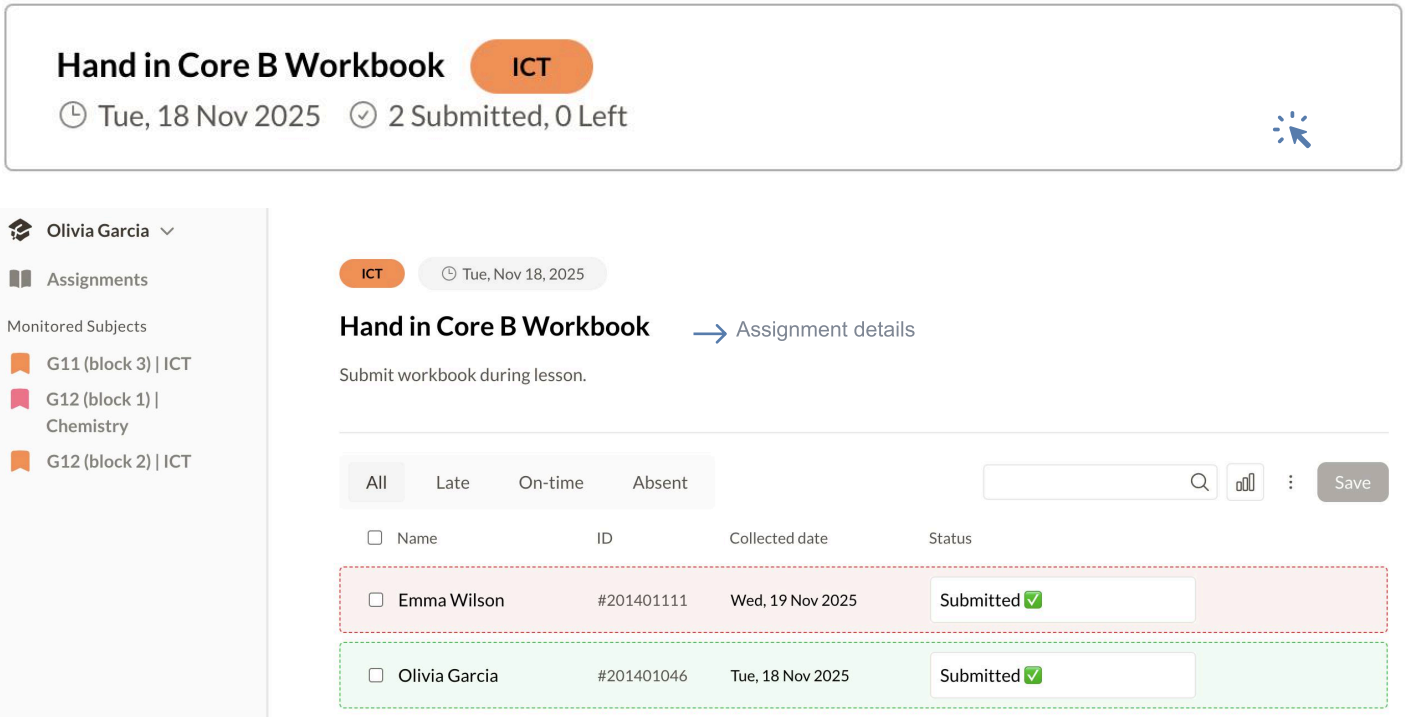
4. Student Monitor View

Subject monitors can access the **Student Monitor View** and inspect the subjects they monitor.



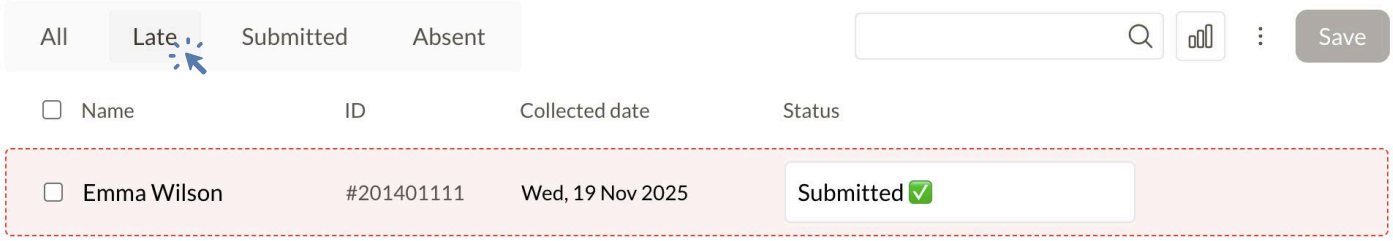
Assignments

The assignments preview include extra details such as the amount left to collect. When an assignment is selected, the user is redirected to view the submission statuses of students (bottom).

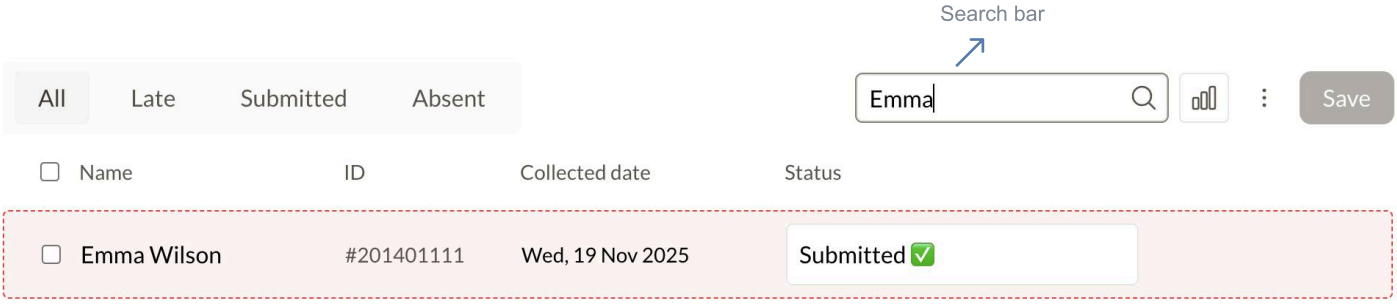


Filtering assignments

They can filter students' assignments based on status, e.g. "All", "Late", "Submitted" and "Absent".

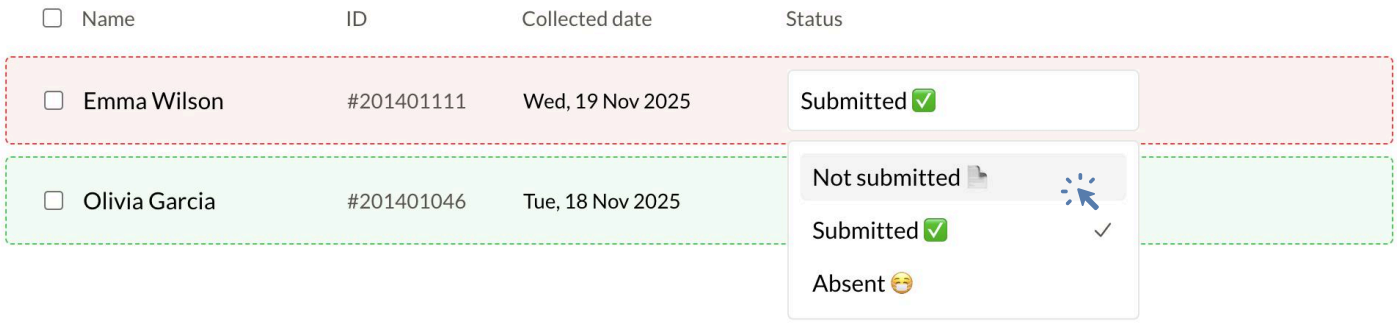


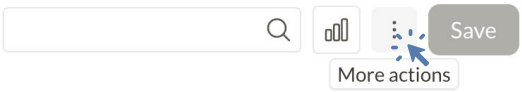
Alternatively, they can filter assignments with the search bar. It searches for substrings within Name and Id.



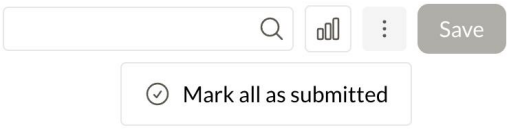
Setting statuses

They can update the submission status of individual students. Assignments marked submitted after the deadline are automatically labelled as "Late," and the records are shown in red. Assignments submitted on-time are shown in green.





Alternatively, they can mark all assignments as submitted under the action menu.

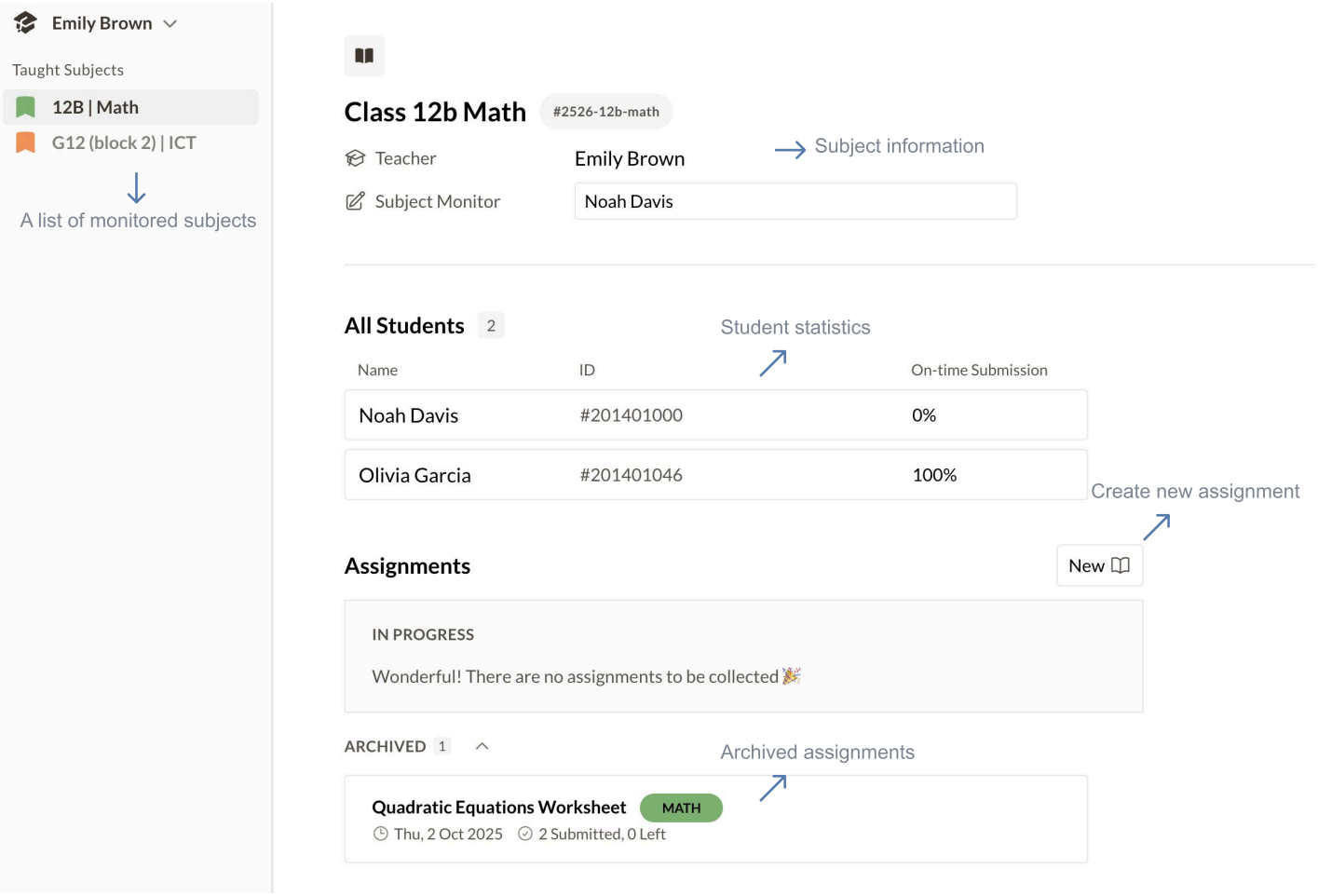


Confirming changes

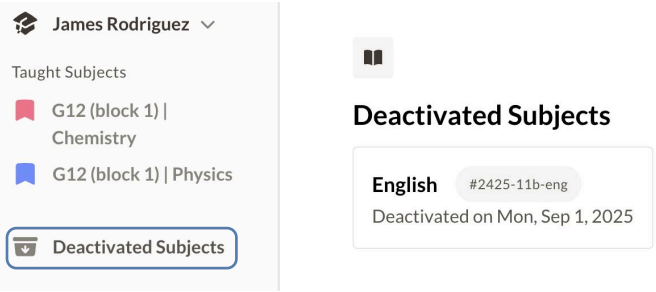
Once changes are made, the user can "Undo" or "Save" their actions. "Undo" reverts all edits, while "Save" pushes the changes to the database.

5. Teacher View

Teachers can access the **Teacher View** and inspect the subjects they teach.

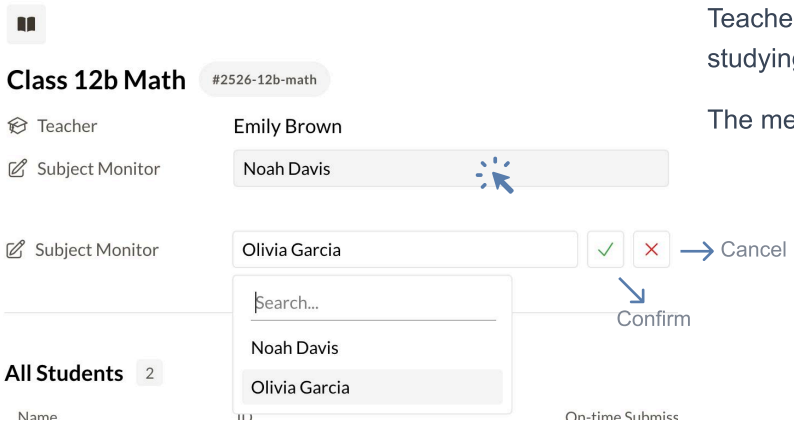


Deactivated subjects



Deactivated subjects are hidden from the main view and shown in a grid layout.

Updating subject monitor

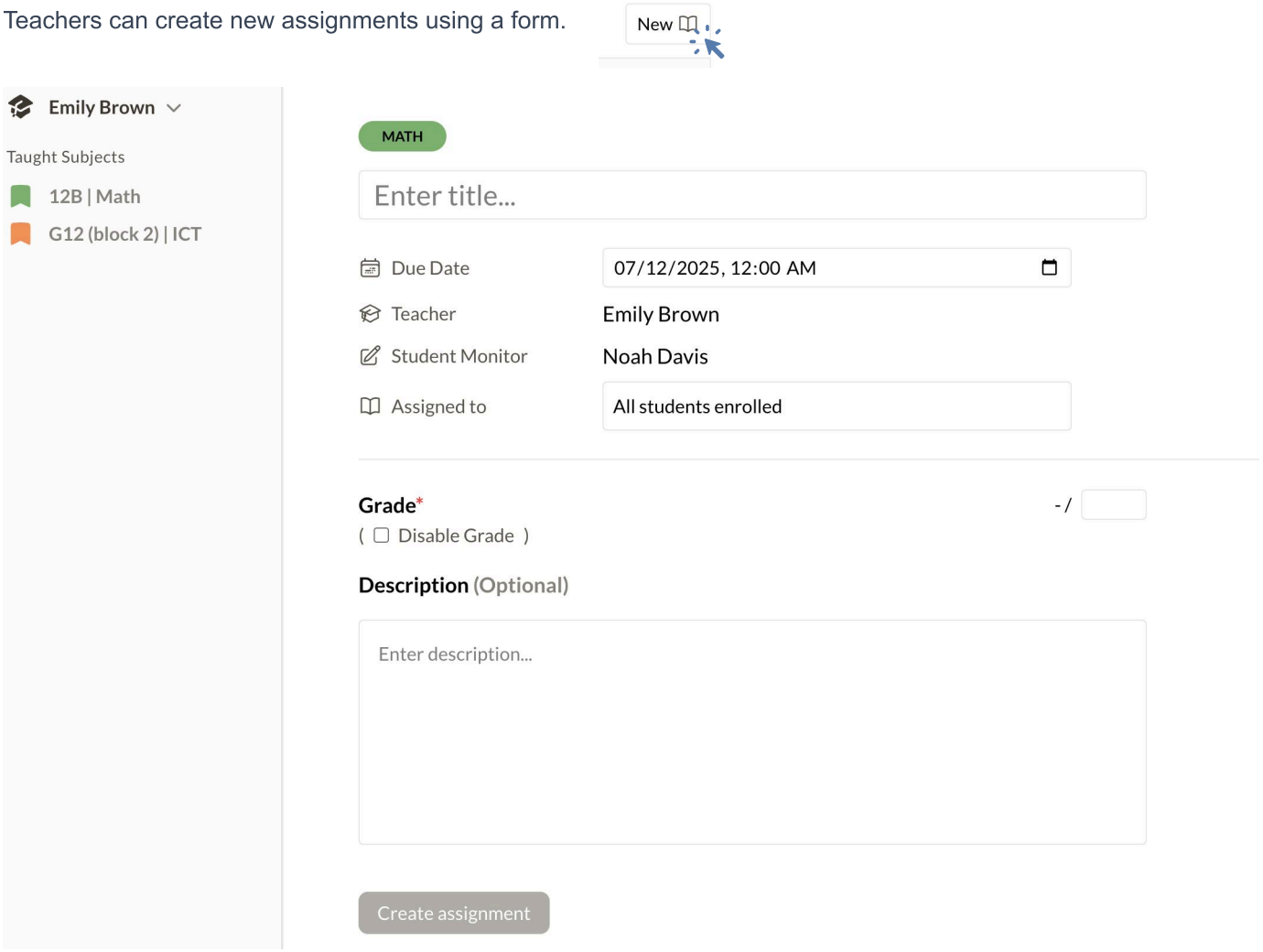


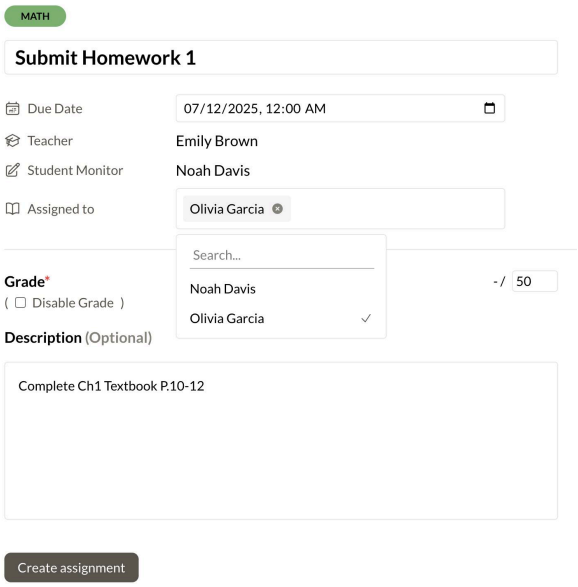
Teachers can change the subject monitor to any student studying the subject.

The menu includes a search bar.

Creating a assignment

Teachers can create new assignments using a form.



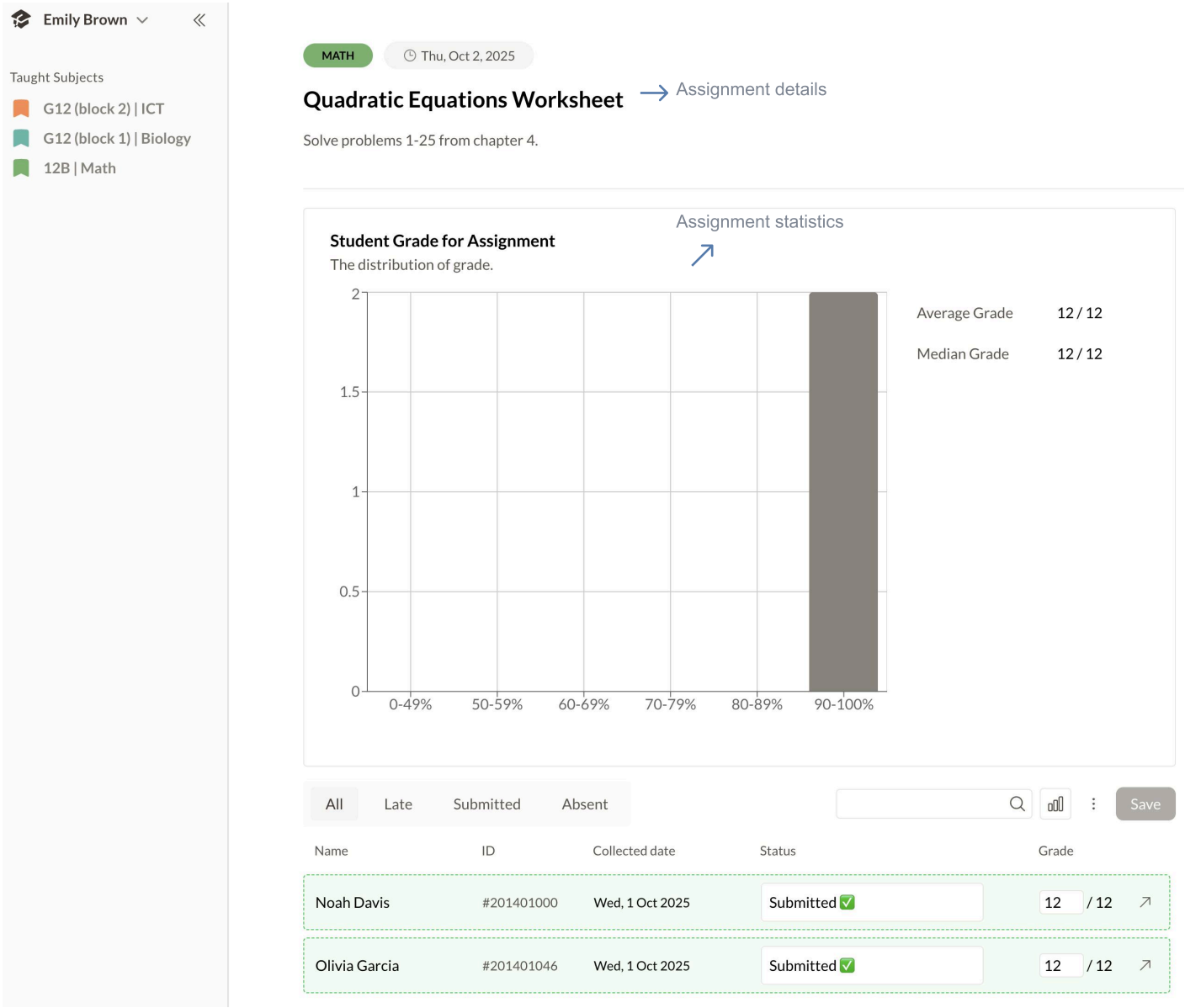


They can edit the assignment's title, due date, grade, description (optional), and also the students to assign to. Teachers can choose to disable grade. The "Create assignment" button is also disabled until all compulsory information is filled in.

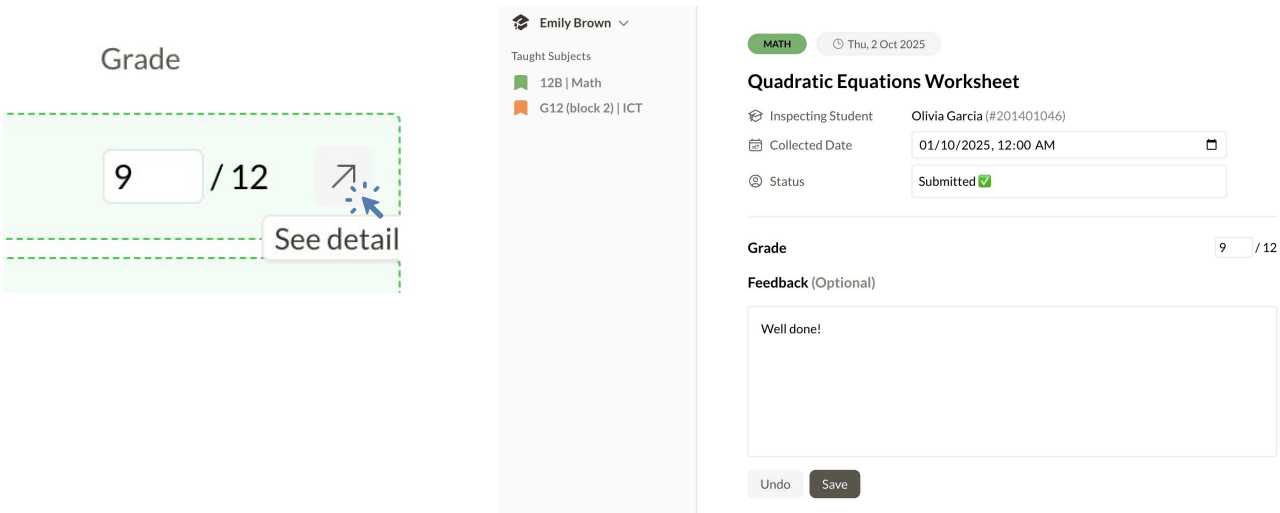
Assignments

When an assignment is selected, the user is redirected to view the submission statuses of students. It is largely similar to that of subject monitors', except teachers can see detailed statistics, offer grade and provide feedback.

The chart is made with Recharts, a react library for rendering charts.



In addition to being able to filter records and set statuses, teachers can also edit the details of each student's assignment, adjust the collected date, and provide feedback in a separate view.

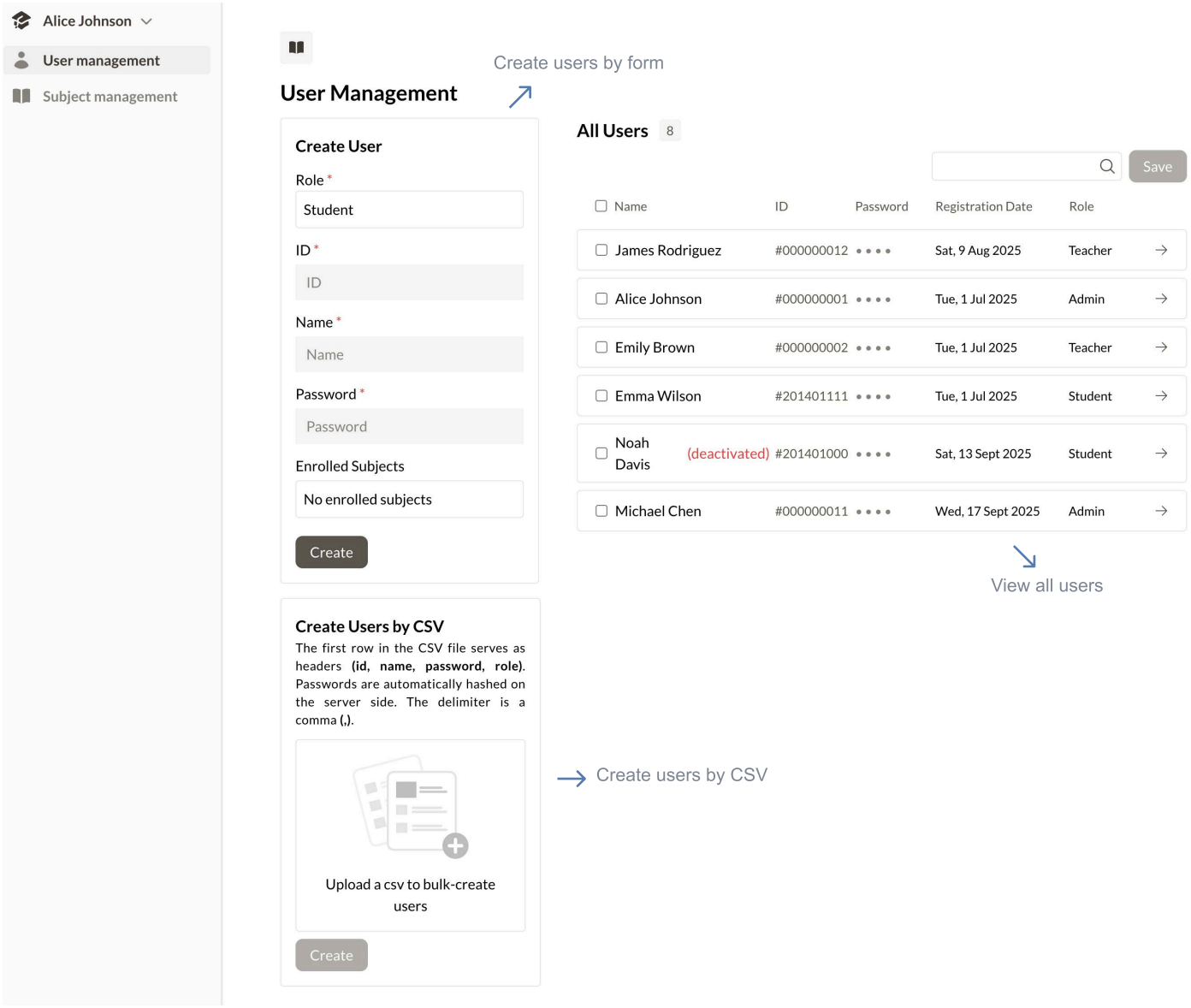


6. Administrator View

Administrators can access the **Administrator View**.

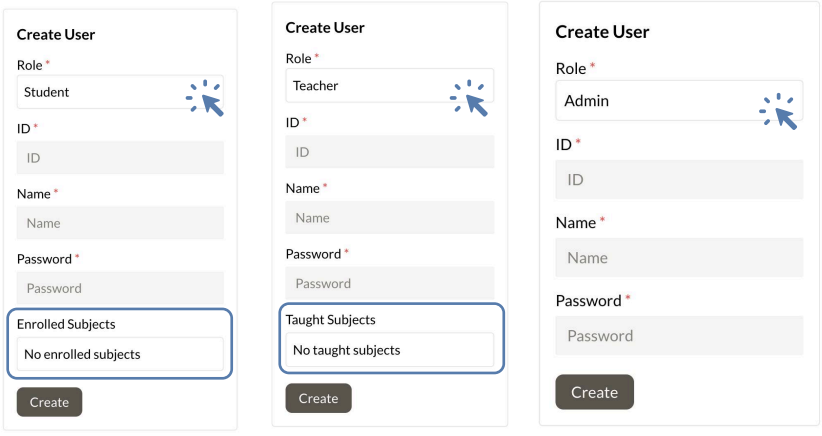
User management

Administrators can create new users by form or CSV, and view all existing users.

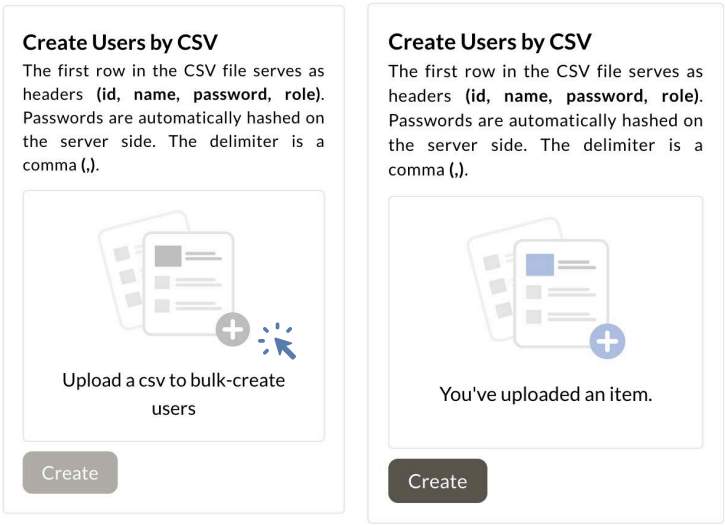


Create users by form

Administrators can enter user information through a form. The form changes with the user's roles: Student, Teacher, Administrator. (from left to right)



Create users by CSV



Administrators can also upload a CSV file to bulk-create users.

The first row in the CSV file should serve as headers (id, name, password, role). Passwords are automatically hashed on the server side. The delimiter is a comma (,).

(Graphics from open-source illustration platform unDraw.)

ATS

```
const [usersArr, setUsersCSV] = useState([]);

function handleCSVChange(e) {
  const file = e.target.files[0];
  if (!file) return;
  Papa.parse(file, {
    header: true,
    complete: (res) => {
      setUsersCSV(res.data);
    },
    error: (err) => {
      console.error(err);
    },
  });
}
```

ATS uses PapaParse to parse CSV into JSON file for further processing.

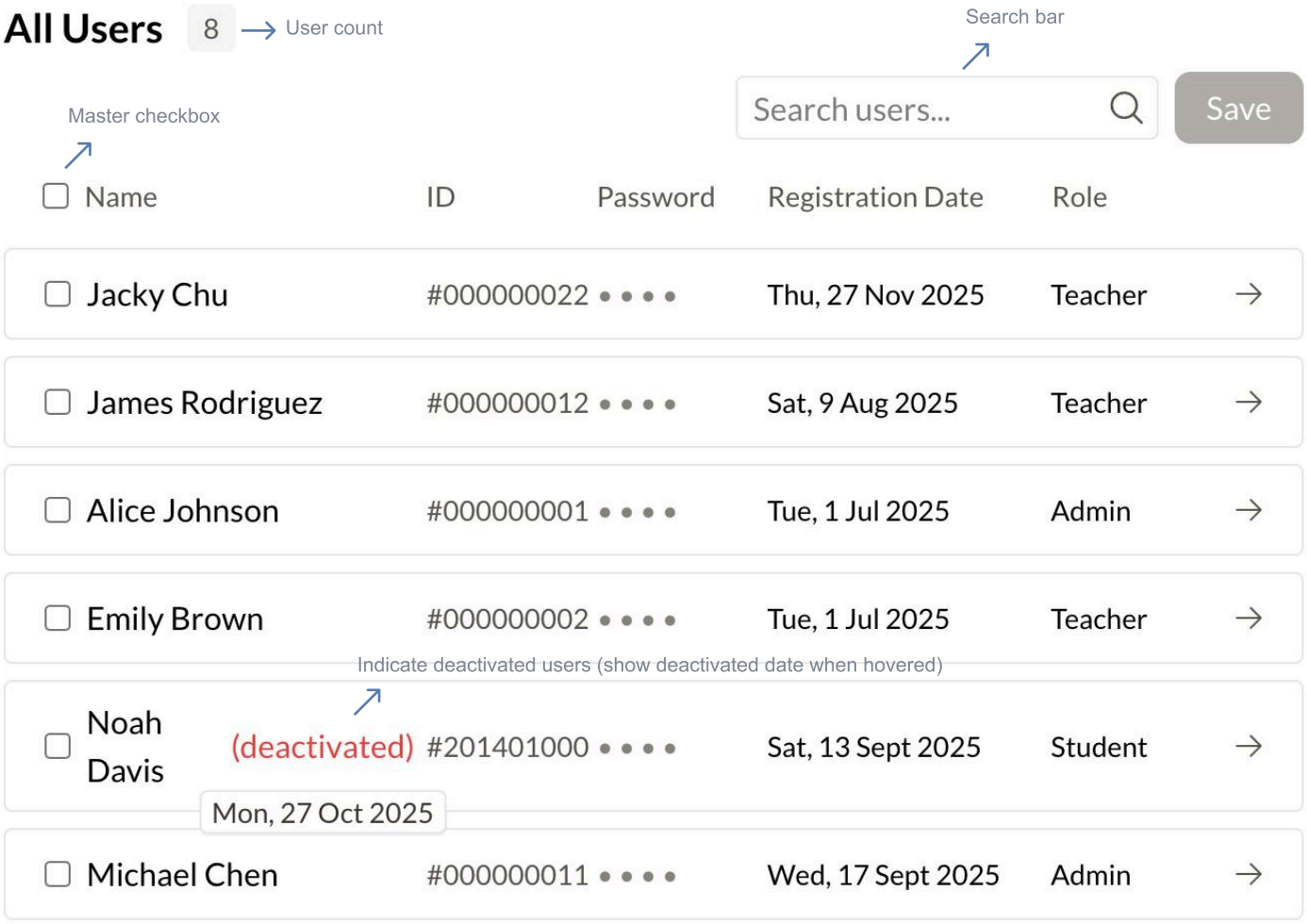
PapaParse can directly parse files. Alternatively, a URL can be generated using the URL.createObjectURL() method, from which the contents can then be fetched.

There are also JavaScript functions to parse CSV data, which utilise Regular Expressions (regex).

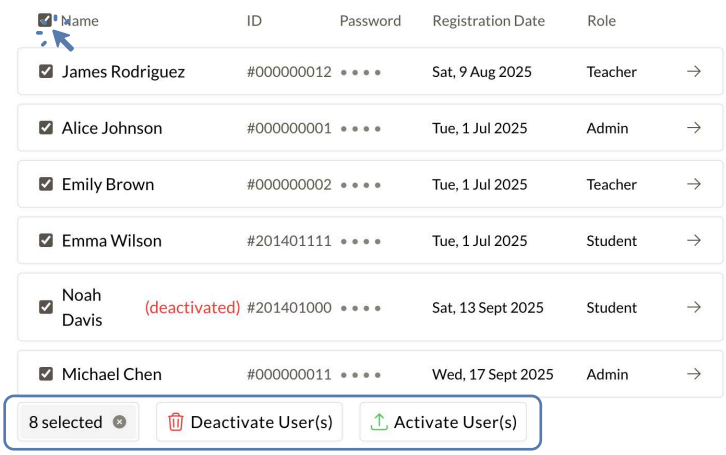
View all users

In the same view, administrators can view all users and perform basic database manipulations.

It should be noted that administrators CANNOT edit or view user's passwords, not even hashed passwords, through the ATS interface. Major changes should still be performed within the database with SQL.



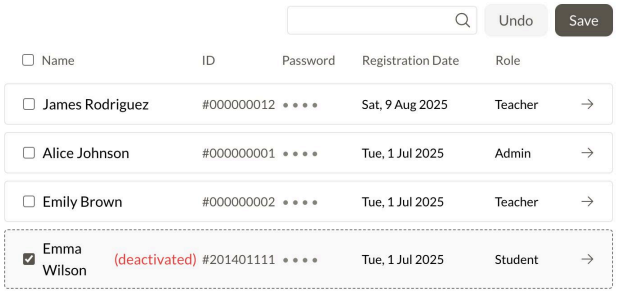
Bulk actions



Administrators can deactivate or reactivate users in bulk. The action menu appears when a checkbox is selected.

Deactivated users cannot log in.

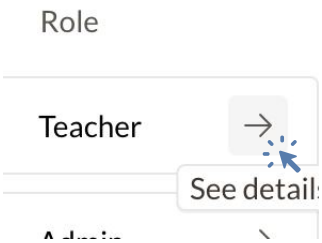
Confirming changes



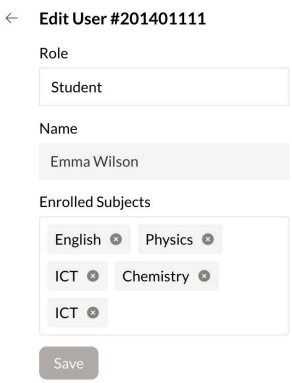
Once changes are made, the user can “Undo” or “Save” their actions. “Undo” reverts all edits, while “Save” pushes the changes to the database.

Altered rows are show in grey.

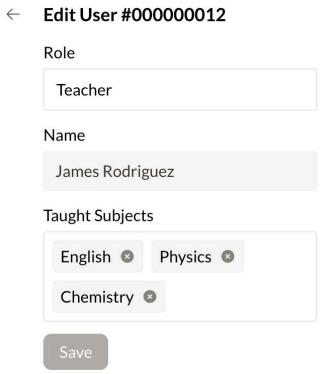
Editing each user



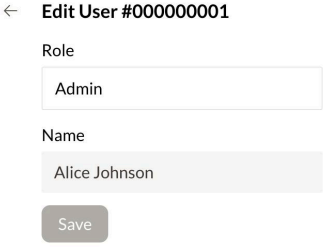
They can also view and edit the details of each user. The user’s role cannot be changed and is for reference only. If a student is a subject monitor, removing the student from the subject sets the subject’s monitor_id to null.



(Students)



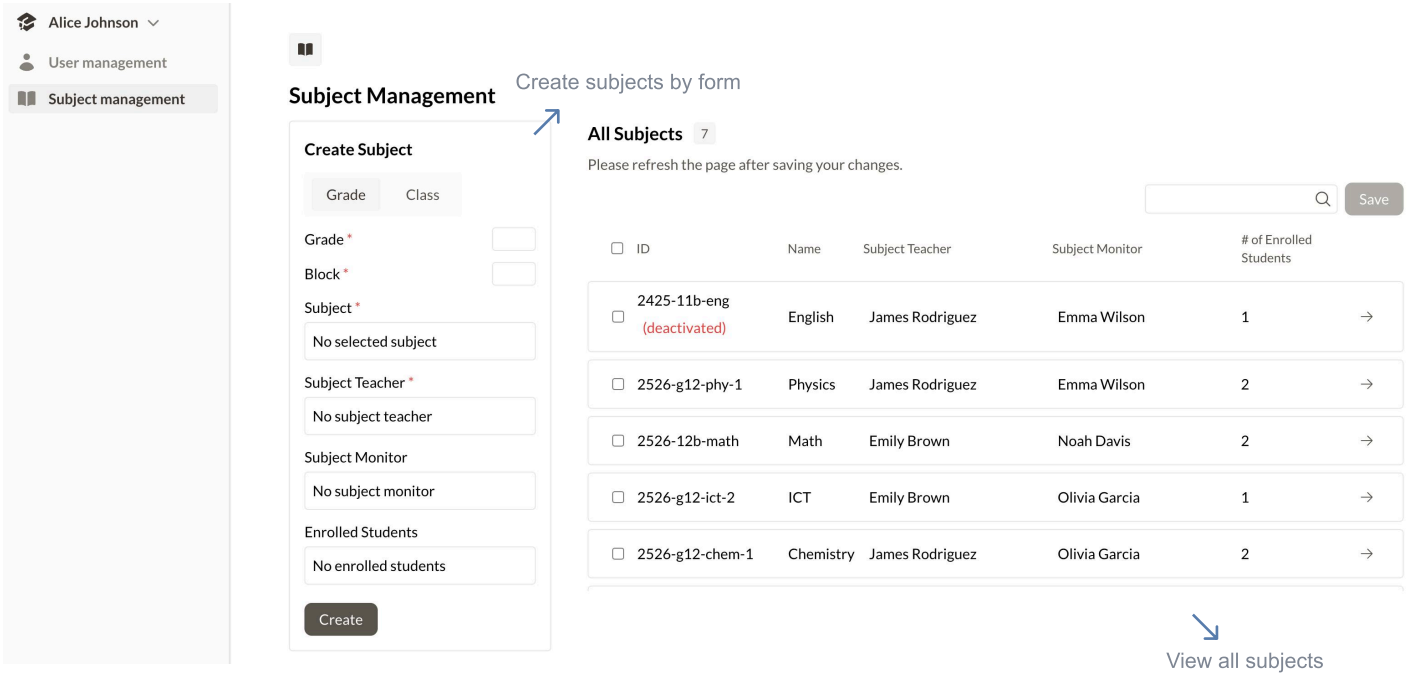
(Teachers)



(Administrators)

Subject management

Administrators can create new subjects by form, and view all existing subjects.



Create subjects by form

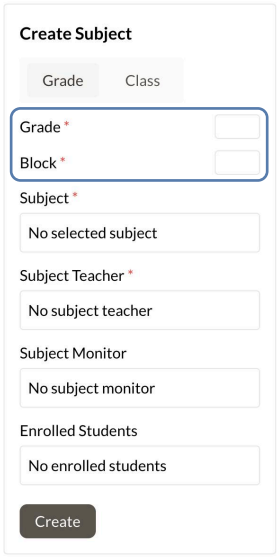
Administrators can enter subject information through a form. There are two types of subjects: grade and class. Subjects that are enrolled by students from a grade have block numbers, while subjects within a class do not.

Subject ids follow the following format:

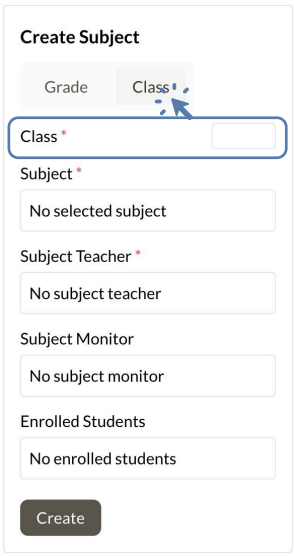
- **Per Grade**: year-grade-name-block (e.g. “2425-g12-phy-1”)
- **Per Class**: year-class-name (e.g. “2526-12b-math”)

All subject names have a shorthand and a corresponding hex code colour.

```
ATS
const subjectShorthands = {
  chi: { name: "Chinese", color: "#D88282" },
  eng: { name: "English", color: "#6B9BFF" },
  math: { name: "Math", color: "#6BB36B" },
  cs: { name: "CS", color: "#FF9D5C" },
  ls: { name: "LS", color: "#A87BB8" },
  bio: { name: "Biology", color: "#54B8A9" },
  chem: { name: "Chemistry", color: "#FF6B8A" },
  phy: { name: "Physics", color: "#6A8CFF" },
  ict: { name: "ICT", color: "#FF8A4D" },
  chis: { name: "Chinese History", color: "#C4856B" },
  his: { name: "History", color: "#C56BC5" },
  geo: { name: "Geography", color: "#48C9C9" },
  eco: { name: "Economy", color: "#B8C94D" },
  m1: { name: "M1", color: "#E6C34D" },
  m2: { name: "M2", color: "#87C787" },
  art: { name: "Arts", color: "#FF8C90" },
  mus: { name: "Music", color: "#6BB8FF" },
  pe: { name: "PE", color: "#A8A8A8" },
  epa: { name: "EPA", color: "#E07FA6" },
  lit: { name: "English literature", color: "#6FA3FF" },
};
```



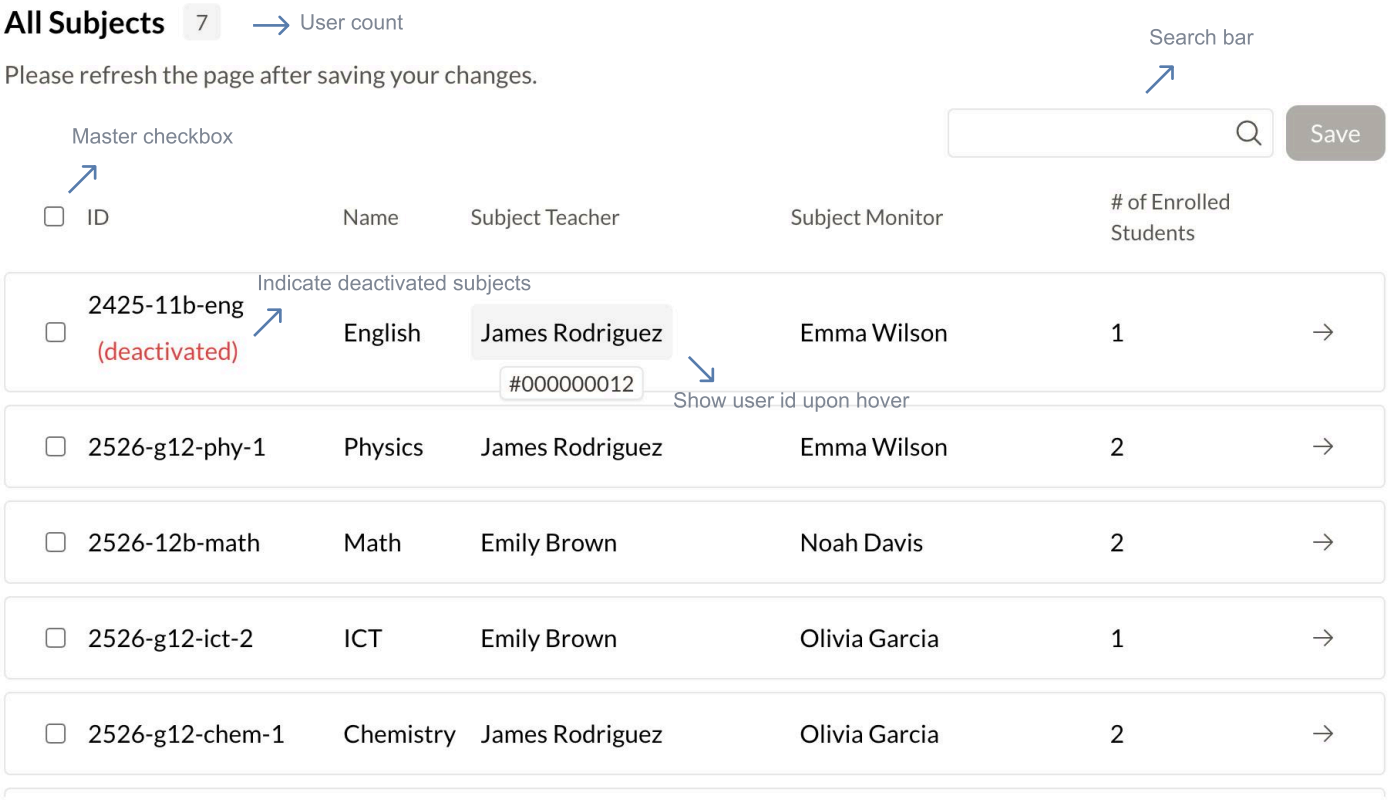
(Per grade)



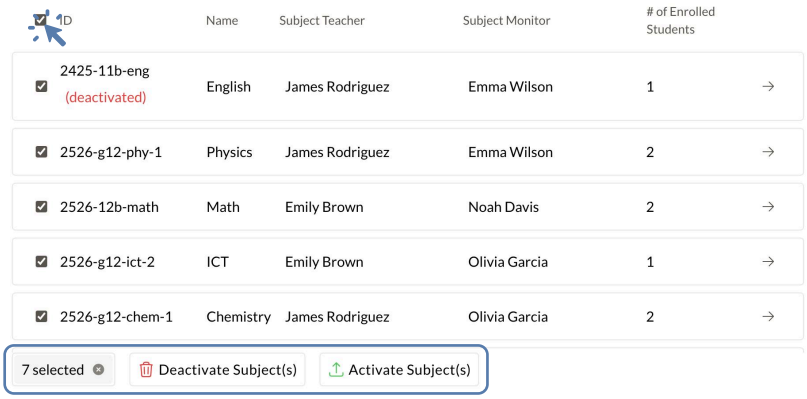
(Per class)

View all subjects

In the same view, administrators can view all subjects and perform basic database manipulations.



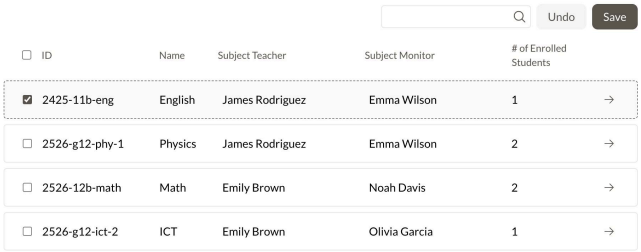
Bulk actions



Administrators can deactivate or reactivate subjects in bulk. The action menu appears when a checkbox is selected.

Deactivated subjects appear in a separate section for teachers and subject monitors.

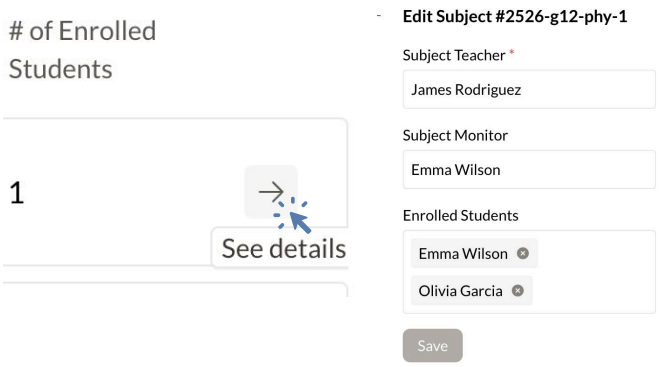
Confirming changes



Once changes are made, the user can “Undo” or “Save” their actions. “Undo” reverts all edits, while “Save” pushes the changes to the database.

Altered rows are show in grey.

Editing each subject

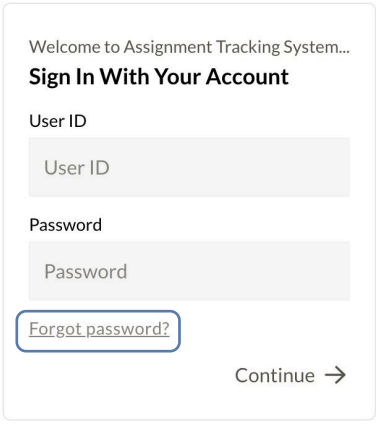


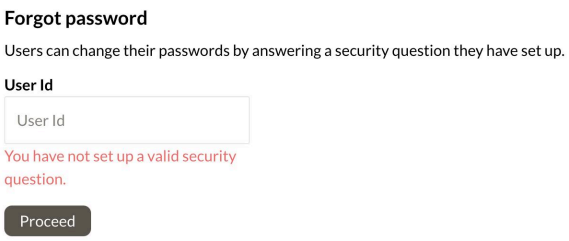
They can also view and edit the details of each subject. If a student is a subject monitor, removing the student from the subject sets the subject's monitor_id to null.

Administrators cannot exit this page until changes are saved or reverted.

6. Forgot Password

Users can change their passwords by answering a security question they have set up if they have forgotten their password.

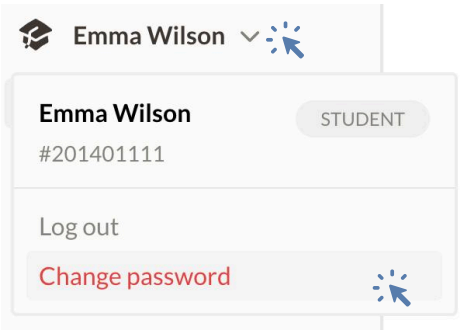


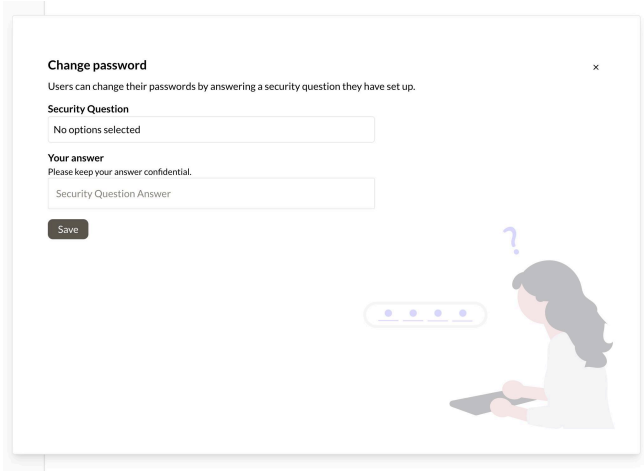


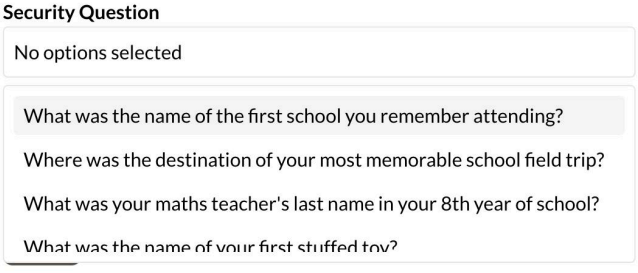
The user cannot reset their password if they have not set up a valid security question.

Setting up security questions

Users are prompted to set up a security question if they haven't already. Alternatively, they can find the option under the action menu.



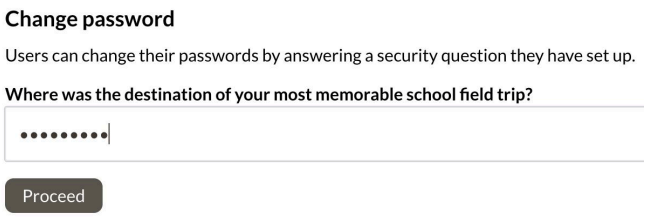




Users can select from a pre-defined list of security questions.

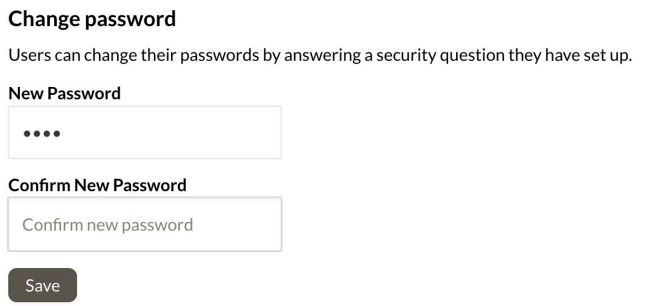
(Security questions selected from OWASP)

Answering security question



Users have to answer the security question if they wish to change their passwords.

Changing password

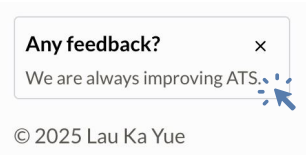


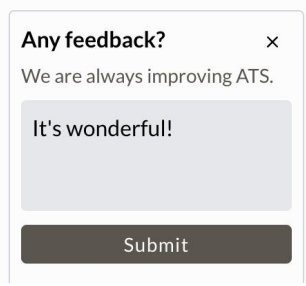
If answered correctly, users can change their passwords. They must input the new password twice.

Other Highlights

ATS has many features that provide additional capabilities.

Provide feedback

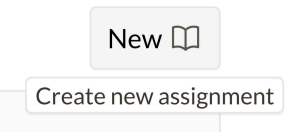




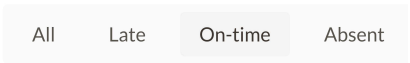
User feedback is crucial for the continuous development of ATS. Users have a 20% chance of receiving a pop-up prompting for feedback.

Custom components

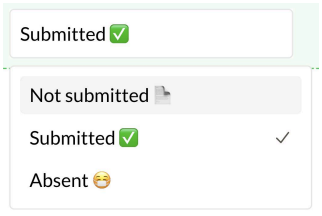
Many custom components were designed to ensure a consistent and look and feel across web pages.



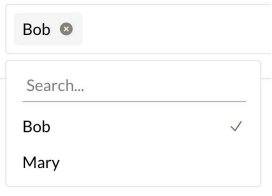
Button component with tooltip.



Radio buttons.

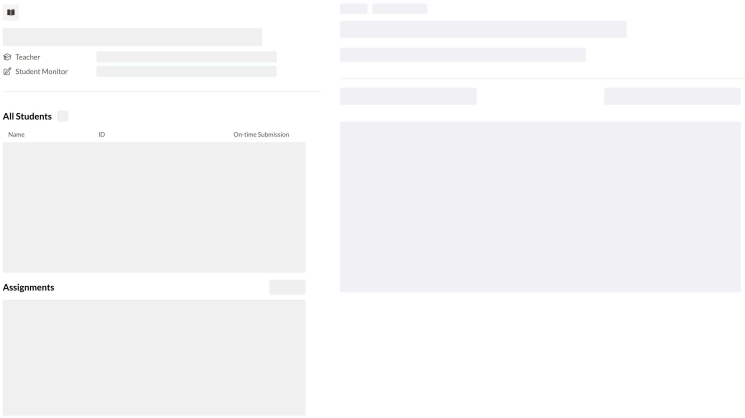


Select menu.



Multi-select component with search bar.

Fallback UI

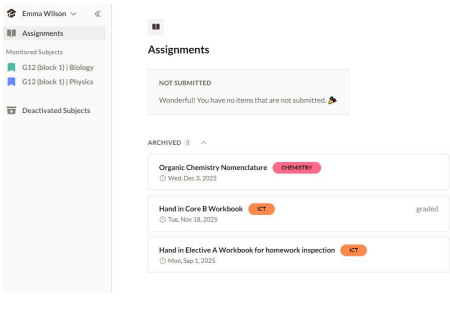


By utilising React Suspense and loading.js, an instant loading state (skeleton) is shown immediately upon navigation. The new content is automatically swapped in once complete. This helps users understand the app is responding and provides better user experience.

Responsive web design

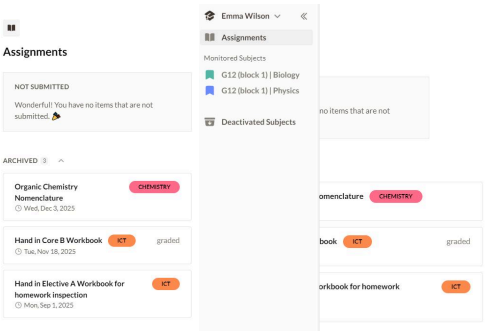
Every utility class in Tailwind can be applied conditionally at different breakpoints, allowing for complex responsive interfaces.

On devices with smaller viewports, the navigation bar, when opened, hovers over the main view. Else, on larger devices such as a computer, the navigation bar acts like a block element. Most pages in ATS, excluding the ones in administrator view, are fully responsive. Below shows screenshots from Assignments in Student View.

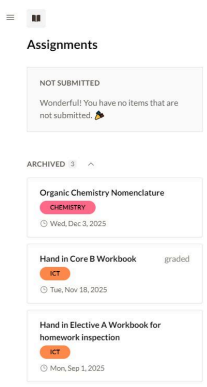


(Computer breakpoint)

(Tablet breakpoint)



(Smaller tablet breakpoint)

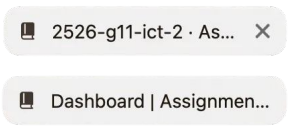


(Phone breakpoint)

Dynamic website titles

```
ATS
// layout.js

export const metadata = {
  title: {
    template: "%s | Assignment Tracking System",
    default: "Assignment Tracking System",
  },
  description: "Created for ABC College",
};

// /teach/[subjectId]/page.js

export async function generateMetadata({ params }) {
  const { subjectId } = await params;

  return {
    title: `${subjectId}`,
  };
}
```

By using the static metadata object and the dynamic generateMetadata function in Next.js, the website title updates dynamically.



Relational Database Design

The Assignment Tracking System is backed with **Neon**, a cloud-based, open-source **Postgres** database. There are a few reasons to choose Neon over a local database such as phpMyAdmin:

1. ATS is built with Next.js and hosted on Vercel. Neon is an out-of-the-box and popular server option for Vercel apps and requires minimal configuration.
2. Neon supports website deployment. Connection is simply made by configuring a environment variable to the Neon database's connection string.
3. Database operations can be handled in a server-side Next.js component.

Database Schema

Table: Users

Column Name	Data Type	Notes/ Constraints
id	CHAR(9)	Primary key
name	CHAR(50)	Not null
password	CHAR(256)	Not null
reg_date	DATETIME	Default NOW()
deactivated_date	DATETIME	
role	CHAR(50)	e.g. 'admin', 'teacher', 'student'

Three views are created from this table: **Students**, **Teachers**, and **Admins**.

Table: Subjects

Column Name	Data Type	Notes/ Constraints
id	CHAR(50)	Primary key
teacher_id	CHAR(9)	References teachers(id)
monitor_id	CHAR(9)	References students(id)
deactivated_date	DATETIME	

Table: Student_Subject

Column Name	Data Type	Notes/ Constraints
subject_id	CHAR(50)	Primary key, References subjects(id)
student_id	CHAR(9)	Primary key, References students(id)

Table: Assignments

Column Name	Data Type	Notes/ Constraints
id	INT	Primary key, Autoincrement
title	CHAR(150)	Not null
description	VARCHAR(500)	
subject_id	CHAR(50)	References subjects(id)
assigned_date	DATETIME	Default NOW()
due_date	DATETIME	
grade	INT	Max points (e.g. 100)

Table: Student_Assignment

Column Name	Data Type	Notes/ Constraints
assignment_id	INT	Primary key, References assignments(id)
student_id	CHAR(9)	Primary key, References student_subject(student_id)
collected_date	DATETIME	
status	CHAR(50)	e.g. "absent", "submitted", "not submitted"
grade	INT	Grade received
feedback	VARCHAR(500)	Teacher's feedback

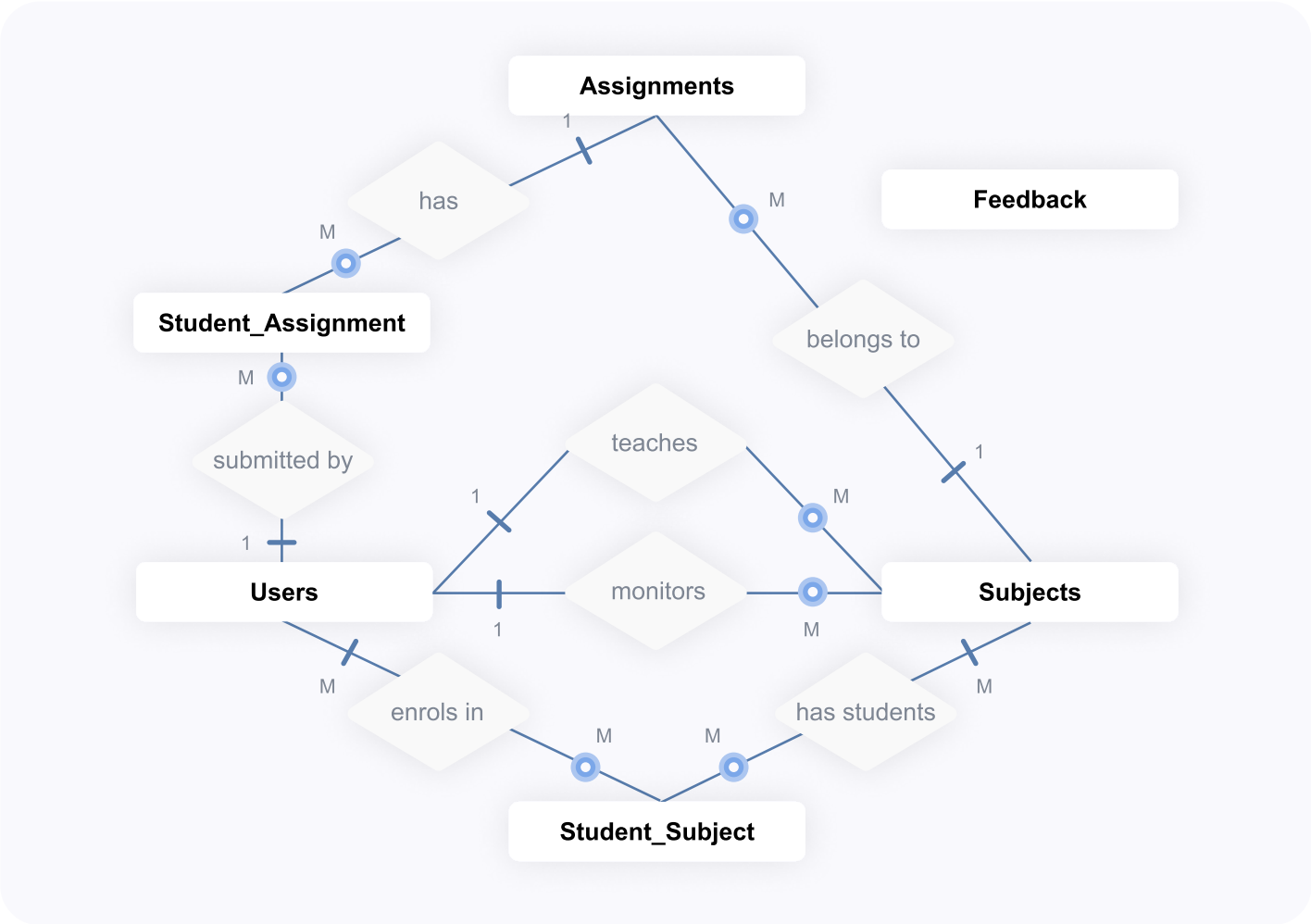
Table: Feedback

This table stores user feedback, an additional feature in ATS.

Column Name	Data Type	Notes/ Constraints
id	UUID	Primary key, Default gen_random_uuid()
description	VARCHAR(500)	
time	DATETIME	Default NOW()
user_id	char(9)	References users(id)

ER Diagram

The following ER diagram describes the relationship between entities. Note that **Feedback** is a standalone table.



Participation Constraint

Participation constraints refer to rules that determine the minimum participation of entities or relationships in a given database.

- **Mandatory side**: For each entity b, there must be at least one entity a.
- **Optional side**: For each entity a, there may or may not be at least one related entity b.

Cardinality

Cardinality of a relationship specifies the maximum participation of entities or relationships.

Normalisation

Normalisation is the process of designing database in a particular way that:

1. Reduce data redundancy
2. Prevent occurrences of anomalies
3. Improve flexibility

First Normal Form

A table in **First Normal Form (1NF)** should follow these rules: every attribute must not contain multiple values; there must not be any repeating attributes.

Criteria	ATS Database
Every attribute must not contain multiple values.	All fields only store a single value.
There must not be any repeating attributes.	There are no repeating attributes in the database.

Therefore, the ATS database is in the first normal form.

Second Normal Form

A table in **Second Normal Form (2NF)** should follow these rules: every normalisation rule of 1NF; there must not be any partial functional dependency between attributes.

Partial function dependency occurs when non-key attributes depend on part of the primary key. It is only possible in composite primary keys.

The following considers the tables with composite primary keys:

Criteria	ATS Database
There must not be any partial functional dependency between attributes.	**Student_Subject**(subject_id, student_id)
	Student_Assignment(assignment_id, student_id, collected_date, status grade, feedback) Hence, all non-key attributes (collected_date, status, grade, feedback) depends on the entire primary key (assignment_id + student_id).

Therefore, the ATS database is in the second normal form.

Third Normal Form

A table in **Third Normal Form (3NF)** should follow these rules: the database follows every normalisation rule of 2NF; there must not be any transitive functional dependency.

Transitive functional dependency occurs when a non-key attribute depend on another non-key attribute.

Criteria	ATS Database
There must not be any transitive functional dependency between attributes.	The main purpose of STATUS in table Student_Assignment is to indicate whether a student was absent or not. It is independent of COLLECTED_DATE. Hence, no non-key attribute depends on another non-key attribute.

Therefore, the ATS database is in the third normal form.

SQL Implementation

Neon adheres closely to the SQL standard, meaning that many standard SQL commands will work as expected. However, PostgreSQL also includes its own extensions and features that offer additional capabilities

Assignments

Create assignment

ATS - Create Assignment

```
INSERT INTO
  assignments (
    title,
    description,
    subject_id,
    assigned_date,
    due_date
  )
VALUES (
  ${assignment.title},
  ${assignment.description},
  ${subjectId},
  ${assignedDate},
  ${assignment.dueDate})
RETURNING id;
```

Neon allows the use of SQL template tags. This article explains why SQL template tags are not vulnerable to SQL injection attacks under @neondatabase/serverless: https://neon.com/blog/sql-template-tags.

Alternatively, the following prepared statement can be used:

ATS - Create Assignment

```
// ...
const res = await client.query('INSERT INTO assignments(title,
description, subject_id, assigned_date, due_date) VALUES ($1, $2, $3,
$4, $5)', [assignment.title, assignment.description, subjectId,
assignedDate, assignment.dueDate])
// ...
```

Get monitored assignments

ATS

```
SELECT
  s.id AS subject_id,
  a.id AS assignment_id,
  a.title AS assignment_title,
  a.description AS assignment_description,
  a.assigned_date,
  a.due_date,
  a.grade AS assignment_grade,
  t.id AS teacher_id,
  t.name AS teacher_name,
  m.id AS monitor_id,
  m.name AS monitor_name,
  count(*) as number_of_students,
  COALESCE(
    JSON_AGG(
      JSON_BUILD_OBJECT(
        'id', st.id,
        'name', st.name,
        'status', sa.status,
        'grade', sa.grade,
        'feedback', sa.feedback,
        'collected_date', sa.collected_date
      )
    ) FILTER (WHERE st.id IS NOT NULL),
    '[]'::json
  ) AS students
FROM assignments a
LEFT JOIN subjects s ON s.id = a.subject_id
LEFT JOIN teachers t ON t.id = s.teacher_id
LEFT JOIN students m ON m.id = s.monitor_id
LEFT JOIN student_assignment sa ON sa.assignment_id = a.id
LEFT JOIN students st ON st.id = sa.student_id
WHERE s.id = ${subjectId}
GROUP BY
  s.id, a.id, a.title, a.description,
  a.assigned_date, a.due_date, a.grade,
  t.id, t.name, m.id, m.name
ORDER BY a.due_date DESC
```

This SQL statement fetches all the assignments in a particular subject for subject monitors.

(Note: JSON_AGG and JSON_BUILD_OBJECT are Postgres-only functions that prepares JSON data for front-end applications. JSON_BUILD_OBJECT is used to construct a JSON object from a set of key-value pairs. JSON_AGG collects values from multiple rows and returns them as a single JSON array. COALESCE returns the first non-null value in the array, so that '[]' is returned instead of null if no subjects are fetched. The :: operator signifies a type cast, which converts from one data type to another. It is functionally equivalent to the SQL-standard CAST() function.)

Get my assignments

```ATS
SELECT
  assignment_id,title as assignment_title,description as assignment_description,
  a.grade as assignment_grade,assigned_date,due_date,status,s.grade as grade,
  feedback,collected_date,subject_id,t.id as teacher_id,t.name as teacher_name,
  m.id as monitor_id,m.name as monitor_name
FROM
  assignments a,student_assignment s,subjects su,teachers t,students m
WHERE
  a.id = s.assignment_id AND s.student_id = ${session?.userId} AND
  a.subject_id = su.id AND su.teacher_id = t.id AND
  su.monitor_id = m.id
ORDER BY
  due_date DESC;
```

This SQL statement fetches all the assignments that a particular student has to submit.

Set assignment

```ATS
UPDATE
  student_assignment
SET
  collected_date = ${student.collected_date},
  status = ${student.status},
  grade = ${student.grade},
  feedback = ${student.feedback}
WHERE assignment_id = ${assignmentId} AND student_id = ${student.id};
```

This SQL statement allows subject monitors to update the submission details of a particular student's assignment.

However, this creates N queries for N students. Alternatively, json_array_elements can be used instead. See Create subject for more.

Subjects

Create subject

In order to allows multiple queries to be executed within a single request, a **transaction** was used to create a subject and insert the rows in the junction table.

```ATS
await sql.transaction([
  sql`
    INSERT INTO subjects (id, teacher_id, monitor_id)
    VALUES (${subjectId}, ${subjectTeacherID}, ${subjectMonitorId});`,
  sql`INSERT INTO student_subject (
        student_id,
        subject_id
      )
      SELECT
        (elem->>'student_id')::TEXT,
        (elem->>'subject_id')::TEXT
      FROM json_array_elements(${JSON.stringify(sqlValues)}::json) AS elem;`,
]);
```

A JSON array is used to store the student_id of the student which needs to be enrolled into a subject with subject_id. Two approaches are possible:

ATS - Enroll students in subjects

```
const res = await Promise.all(
  arr.map(async (item) => {
    await sql`
      INSERT INTO student_subject (
        subject_id,
        student_id
      )
      VALUES (${item.subject_id}, ${item.student_id})
    `
)
```

ATS

```
const res = sql`
INSERT INTO student_subject (
  student_id,
  subject_id
)
SELECT
  (elem->>'student_id')::TEXT,
  (elem->>'subject_id')::TEXT
FROM
  json_array_elements(
    ${JSON.stringify(sqlValues)}::json
  )AS elem;
`
```

The approach on the left requires N requests for a N-sized array, while the approach on the right only sends 1 request. This significantly reduces the processing time of the transaction.

(Note: The ->> operator extracts JSON object field with the given key, as text. (e.g. '{"a": 1, "b": 2}'::json ->> 'b' → 2)). Code built referencing https://medium.com/developer-rants/how-to-insert-a-json-array-into-a-postgresql-table-with-a-single-query-9afd10ac4e88)

Get all subjects

This SQL statement retrieves all subjects from the database.

ATS - Fetch all subjects

```
SELECT
  s.id AS id,
  t.id AS teacher_id,
  t.name AS teacher_name,
  m.id AS monitor_id,
  m.name AS monitor_name,
  deactivated_date,
  COALESCE(
    JSON_AGG(
      JSON_BUILD_OBJECT('id', st.id, 'name', st.name)
    ) FILTER (WHERE st.id IS NOT NULL),
    '[]'
  ) AS students
  FROM
  subjects s
  LEFT JOIN teachers t ON s.teacher_id = t.id
  LEFT JOIN students m ON s.monitor_Id = m.id
  LEFT JOIN student_subject ss ON ss.subject_id = s.id
  LEFT JOIN students st ON ss.student_id = st.id
  GROUP BY
  s.id, t.id, t.name, m.id, m.name, deactivated_date
```

ATS - Fetch all subjects

```
[
  {
    id: "2526-11b-math",
    teacher_id: "00000002",
    teacher_name: "Teacher Ada",
    monitor_id: "201401000",
    monitor_name: "David",
    deactivated_date: null,
    students: [
      {
        id: "201401000",
        name: "David",
      },
      {
        id: "201401111",
        name: "Mary",
      },
    ],
  }
]
```

Get monitored subjects

ATS

```
SELECT
  s.id as id,
  s.teacher_id as teacher_id,
  t.name as teacher_name,
  s.monitor_id as monitor_id,
  m.name as monitor_name,
  count(*) as number_of_students,
  s.deactivated_date
FROM subjects s
JOIN teachers t ON s.teacher_id = t.id
LEFT JOIN students m ON s.monitor_id = m.id
LEFT JOIN student_subject ss ON ss.subject_id = s.id
WHERE monitor_id = ${session.userId}
GROUP BY s.id, s.teacher_id, t.name, s.monitor_id, m.name, s.deactivated_date
```

This SQL statement fetches all the subjects a student has to monitor.

Get taught subjects

```ATS
SELECT
  s.id AS id,
  t.id AS teacher_id,
  t.name AS teacher_name,
  m.id AS monitor_id,
  m.name AS monitor_name,
  s.deactivated_date,
  COALESCE(
    JSON_AGG(
      JSON_BUILD_OBJECT('id', st.id, 'name', st.name)
    ) FILTER (WHERE st.id IS NOT NULL),
    '[]'
  ) AS students
FROM
  subjects s
  LEFT JOIN teachers t ON s.teacher_id = t.id
  LEFT JOIN students m ON s.monitor_id = m.id
  LEFT JOIN student_subject ss ON ss.subject_id = s.id
  LEFT JOIN students st ON ss.student_id = st.id
WHERE
  s.teacher_id = ${session.userId}
GROUP BY
  s.id, t.id, t.name, m.id, m.name, s.deactivated_date
```

This SQL statement fetches all the subjects a teacher teaches, along with the students studying those subjects.

Set subject

These SQL statements allow users to update subject details.

```ATS
await Promise.all(
  newlyEnrolled.map(async (item) => {
    const { subjectId, studentId } = item;

    await sql`
        INSERT INTO
            student_subject (student_id, subject_id)
        VALUES
            (${studentId}, ${subjectId});
    `;
  }),

  removedEnrolled.map(async (item) => {
    const { subjectId, studentId } = item;

    await sql`
        DELETE FROM
            student_subject
        WHERE
            student_id = ${studentId} AND
            subject_id = ${subjectId};
    `;

    await sql`
        UPDATE
            subjects
        SET
            monitor_id = null,
        WHERE
            id = ${subjectId} AND
            monitor_id = ${studentId}
    `;
  })
);
```

```ATS
updatedSubjectsObj.map(async (subject) => {
  await sql`
      UPDATE
          subjects
      SET
          teacher_id = ${subject.teacher_id},
          monitor_id = ${subject.monitor_id},
          deactivated_date = ${subject.deactivated_date}
      WHERE
          id = ${subject.id}
  `;
})
```

```ATS
await Promise.all(
  newlyTaught.map(async ({ subjectId, teacherId }) => {
    await sql`
        UPDATE
          subjects
        SET
          teacher_id = ${teacherId}
        WHERE
          id = ${subjectId};
    `;
  }),
  removedTaught.map(async ({ subjectId, teacherId }) => {
    await sql`
        UPDATE
          subjects
        SET
          teacher_id = null
        WHERE
          id = ${subjectId} AND
          teacher_id = ${teacherId};
    `;
  })
);
```

Users

Create user

```ATS
const sql = neon(`${process.env.STORE_DATABASE_URL}`);
await sql`
  INSERT INTO
  users (id, name, password, role, reg_date, deactivated_date)
  VALUES (${userId}, ${name}, ${hashedPassword}, ${role},
  ${new Date().toISOString()}, null)`;

if (enrolledSubjectIds && enrolledSubjectIds.length > 0) {
  await updateSubjectStudents(
    enrolledSubjectIds.map((item) => {
      return {
        subjectId: item.id,
        studentId: userId,
      };
    }),
  );
} else if (taughtSubjectIds && taughtSubjectIds.length > 0) {
  await updateSubjectTeachers(
    taughtSubjectIds.map((item) => {
      return {
        subjectId: item.id,
        teacherId: userId,
      };
    }),
  );
}
```

When a user is created on client-side, an INSERT statement creates a user with the provided details. Then, subjects are updated with the new teacher and their students.

Bulk create users

```ATS
const sql = neon(`${process.env.STORE_DATABASE_URL}`);
await sql.transaction(
  users.map(async (user) => {
    if (
      user.id === "" ||
      user.name === "" ||
      user.password === "" ||
      user.role === ""
    ) {
      return "";
    }

    const hashedPassword = await bcrypt.hash(user.password, saltRounds);

    return sql`
      INSERT INTO
        users (id, name, password, role, reg_date, deactivated_date)
      VALUES (${user.id}, ${user.name}, ${hashedPassword}, ${
        user.role
      }, ${new Date().toISOString()}, null)
    `;
  }),
);
```

Users are bulk-created over a transaction. The user will not be created if some required fields are empty.

Change user password

```ATS
UPDATE users SET password = ${hashedPassword} WHERE id = ${session.userId}
```

This SQL statement changes the user's password.

Get all users

```ATS
SELECT id, name, password, role, reg_date, deactivated_date FROM users;
```

This SQL statement fetches all users for admins.

Get user

```ATS
if (userId) {
  const userData = (await sql`SELECT * FROM users WHERE id = ${userId};`)[0]
  return userData
} else {
  const userData = (
    await sql`SELECT * FROM users WHERE id = ${session.userId};`
  )[0]
  return userData
}
```

This SQL statement fetches the information of a user.

Set users

```ATS
const results = await Promise.all(
  updatedUsers.map(async (user) => {
    const response =
      await sql`
        UPDATE users
        SET
          name=${user.name},
          role=${user.role},
          deactivated_date=${user.deactivated_date}
        WHERE id=${user.id}`
  })
)
```

This SQL statement allows admins to update user details.

Security questions

```ATS
SELECT id, security_id FROM users WHERE id=${session.userId};

UPDATE users SET security_id = ${securityQuestionId}, security_ans = ${hashedAns} WHERE id =
${session.userId};

SELECT id, security_id, security_ans FROM users WHERE id=${session.userId};
```

These SQL statements handle security questions.

Feedback

```ATS
INSERT INTO ats_feedback(
  description,
  time,
  user_id
) VALUES (${feedback.trim()}, ${
  additionalData.time
}, ${session.userId}) RETURNING id;
```

This SQL statements inserts a feedback, returning the unique UUID.

Testing and Evaluation

One of the main goals in ATS is to offer efficiency and accuracy. This starts with testing every feature and turning evaluation insights into part of the development process.

Error Logging

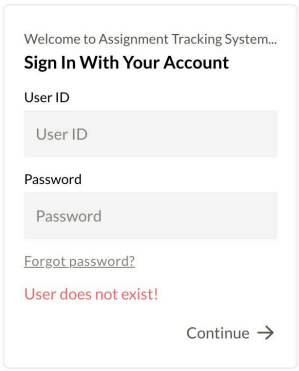
```
ATS
try {
  //...
  if (!response.ok) {
    return {
      success: false,
      message: "Password invalid.",
    };
  }
  //...
  return {
    success: true,
    message: `Login successful.`,
  };
} catch (err) {
  console.error("Error logging in:", err);

  return {
    success: false,
    message: `${err.message}. ${err.detail}` || "Error logging in.",
  };
}
```

Errors are inevitable. The key is to gracefully handle the "sad paths" so users do not get surprised by failing situations.

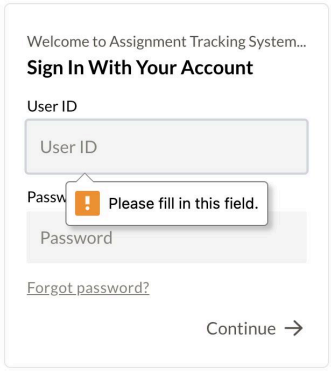
Errors can be divided into two categories: expected errors and uncaught exceptions. Expected errors are those that can occur during the normal operation of the application, such as those from form validation or failed requests. For these errors, expected errors are modelled as return values. Uncaught exceptions are unexpected errors that indicate bugs or issues that should not occur during the normal flow of the application. These are handled by throwing errors, which are then caught by a try/catch block.

Access Control

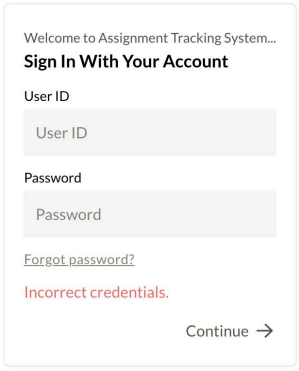
Access control manages who can log into ATS and what they can do once they're in. useActionState() hook from React allows the system to display messages such as an error message returned by a server function. The following shows how the system responds to different authentication scenarios:



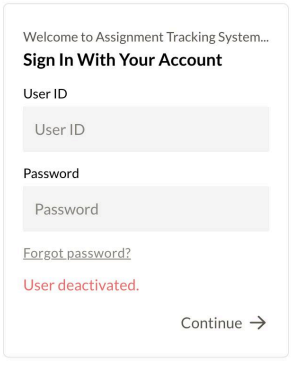
(Error shown when user does not exist)



(Presence check)



(Error shown when login credentials are incorrect)



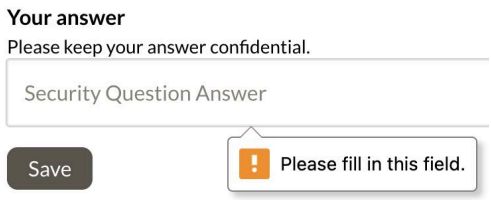
(Error shown when user is deactivated)

Upon calling a database function, the system verifies whether the request was from a valid user session. Certain database functions are reserved for administrators only. requireAdmin() checks the role of the user, returning an error if the user is not authorised to perform the action.

Data Validation

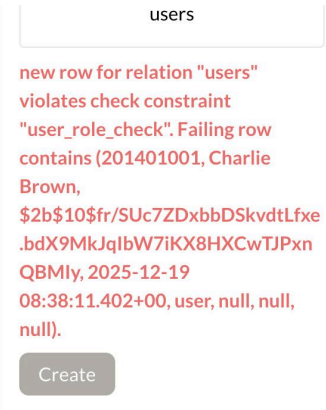
Data validation ensures that the input data is reasonable and matches certain rules. Input fields in ATS go through rigorous client-side and server-side validation. Below are some examples.

Presence check



When a user sets up a security question answer, an error is shown if the input field is empty.

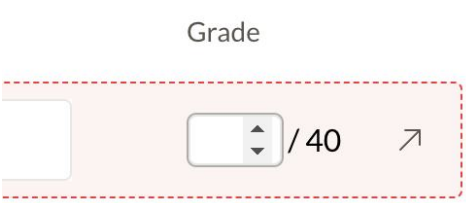
Fixed value check



Roles of users must be either “student”, “teacher” or “admin”.

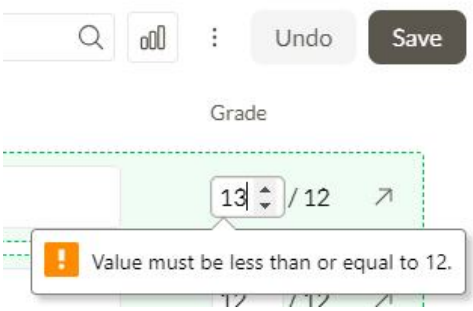
When an administrator attempts bulk-creating users, but one of the users have an invalid role, an error is shown.

Type check



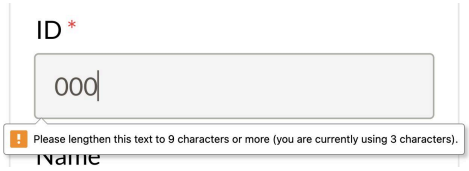
A teacher cannot input characters as score, as the input field is restricted to numbers only.

Range check



When a teacher assigns a grade that is out of range, i.e. negative value or exceeds the maximum grade of the assignment, an error is shown.

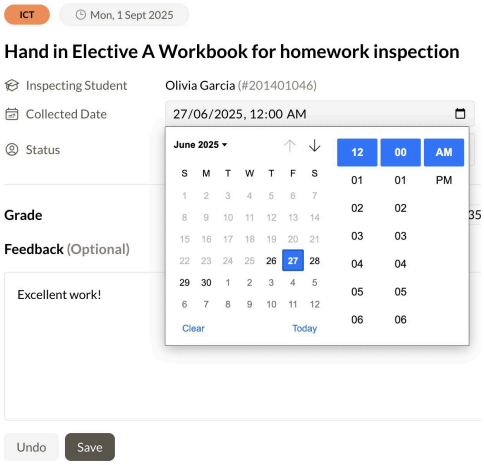
Length check



A user's id must be 9 characters long, following that of student's (e.g. "201401111")

When an administrator attempts creating a user by form with id less than or more than 9 characters, an error is shown.

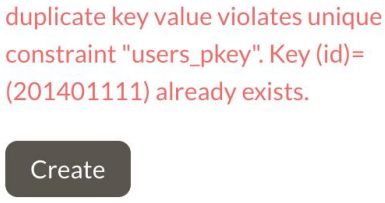
Consistency check



The collected date of an assignment must be later than its assigned date.

The date picker for teachers has a minimum value set to the assignment's assigned date.

Uniqueness check



The id of a user must be unique.

When an administrator attempts creating a user by form with id that already exists, an error is shown.

Other examples

Below are test cases demonstrating the application's response during erroneous input situations. The user input is sometimes restricted by UI components (e.g. date picker, radio button, select menu), so certain input scenarios are impossible, and are not discussed here.

Log in

Type of data validation	Situation	Expected Output	Reason	Actual Output
Presence check	- User id is empty - Password is empty	Please fill in this field.	/	Please fill in this field.

Forgot Password / Change Password

Type of data validation	Situation	Expected Output	Reason	Actual Output
Presence check	- Forgot Password > User Id is empty - Change Password > Answer to security question is empty - Change Password > Set up security question > Security Question is empty - Change Password > Set up security question > Your answer is empty	Please fill in this field.	/	Please fill in this field.
	- Collect Assignment > Student Id > Grade is negative	Value must be greater than or equal to 0.	/	Value must be greater than or equal to 0.
Length check	- Change Password > Set up security question > Your answer is less than 5 characters long	Please lengthen this text to 5 characters or more (you are currently using 4 characters).	Answers to security questions have to be long enough to ensure account safety.	Please lengthen this text to 5 characters or more (you are currently using 4 characters).
Consistency check	- Change Password > New password and confirm new password are not the same.	Passwords do not match.	/	Passwords do not match.

Teacher > Assignments

Type of data validation	Situation	Expected Output	Reason	Actual Output
Presence check	- Create Assignment > Title is empty - Create Assignment > Assigned to no students - Create Assignment > Grade is empty (if grade is not disabled)	[Submit button is disabled.]	A valid assignment must have a title and assigned to at least one student.	[Submit button is disabled.]
Type check	- Collect Assignment > Student Id > Grade is “--” or “-5-” or similar	Please enter a number.	/	Please enter a number.
Range check	- Collect Assignment > Score is greater than maximum grade	Value must be less than or equal to [maximum grade].	/	Value must be less than or equal to [maximum grade].
	- Collect Assignment > Score is negative	Value must be greater than or equal to 0.	/	Value must be greater than or equal to 0.
	- Collect Assignment > Student Id > Grade is greater than maximum grade	Value must be less than or equal to [maximum grade].	/	Value must be less than or equal to [maximum grade].
	- Collect Assignment > Student Id > Grade is negative	Value must be greater than or equal to 0.	/	Value must be greater than or equal to 0.

Administrator > User Management

Type of data validation	Situation	Expected Output	Reason	Actual Output
Presence check	- Create User > ID is empty - Create User > Name is empty - Create User > Password is empty - All Users > Edit User > Name is empty	Please fill in this field.	/	Please fill in this field.
	- Create Users by CSV > ID is empty - Create Users by CSV > Name is empty - Create Users by CSV > Password is empty - Create Users by CSV > Role is empty	No valid users provided. Please check that all users have their ids, names, passwords, and roles specified in the correct format.	/	No valid users provided. Please check that all users have their ids, names, passwords, and roles specified in the correct format.
Fixed value check	- Create Users by CSV > Role is “user”	new row for relation “users” violates check constraint “user_role_check”. Failing row contains (201401001, Charlie Brown, `hashed password`, `date`, user, null, null, null).	Role must be “student”, “teacher” or “admin”	new row for relation “users” violates check constraint “user_role_check”. Failing row contains (201401001, Charlie Brown, `hashed password`, `date`, user, null, null, null).
Length check	- Create User > ID is less than 9 characters long	Please lengthen this text to 9 characters or more (you are currently using 1 character).	Id must be exactly 9 characters.	Please lengthen this text to 9 characters or more (you are currently using 1 character).
Uniqueness check	- Create User > ID is 201401111 (which already exists in the database)	duplicate key value violates unique constraint “users_pkey”. Key (id)=(201401111) already exists.	User id must be unique.	duplicate key value violates unique constraint “users_pkey”. Key (id)=(201401111) already exists.

Feedback

Type of data validation	Situation	Expected Output	Reason	Actual Output
Presence check	- Feedback is empty - Feedback is comprised of the space character only	Empty feedback!	/	Empty feedback!

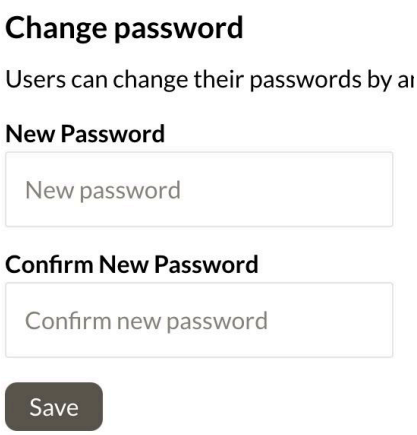
Administrator > Subject Management

Type of data validation	Situation	Expected Output	Reason	Actual Output
Presence check	- Create Subject > Grade is empty - Create Subject > Block is empty	Please fill in this field.	/	Please fill in this field.
	- Create Subject > Subject is empty - Create Subject > Subject Teacher is empty	Subject name missing. / Subject teacher missing.	/	Subject name missing. / Subject teacher missing.
	- All Users > Edit Subject > Subject Teacher is empty	Subject teacher missing.	/	Subject teacher missing.
Fixed value check	- Create Users by CSV > Role is "user"	new row for relation "users" violates check constraint "user_role_check". Failing row contains (201401001, Charlie Brown, `hashed password`, `date`, user, null, null, null).	Role must be "student", "teacher" or "admin".	new row for relation "users" violates check constraint "user_role_check". Failing row contains (201401001, Charlie Brown, `hashed password`, `date`, user, null, null, null).
Type check				
Range check	- Create Subject > Grade is negative - Create Subject > Grade is greater than 12	Value must be greater than or equal to 1. / Value must be less than or equal to 12	Grade must be between 1 and 12.	Value must be greater than or equal to 1. / Value must be less than or equal to 12
	- Create Subject > Block is negative - Create Subject > Block is greater than 12	Value must be greater than or equal to 1.	Block must be a positive integer.	Value must be greater than or equal to 1.

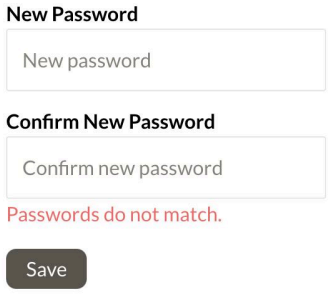
Data Verification

Data verification determines whether the input data is accurate and consistent when being transferred between media.

Inputting data twice



Users must enter their new password twice. The two copies are compared with each other to see whether they are the same. The system returns an error if the passwords do not match.



(Error shown when passwords do not match)

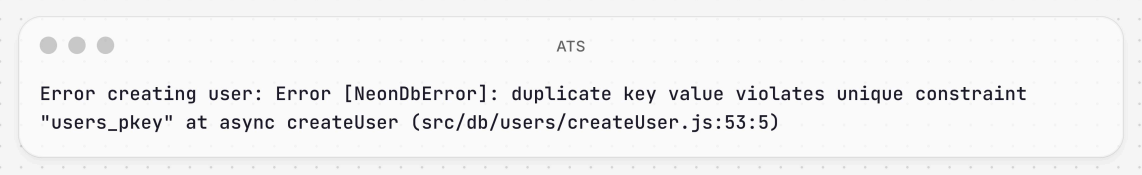
Database Integrity Constraints

The system must safeguard database integrity, namely in three areas: **entity integrity, referential integrity** and **domain integrity**.

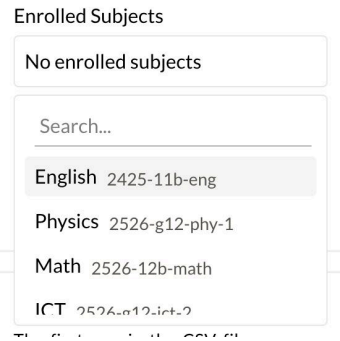
Entity integrity

Entity integrity ensures that each row in a table is uniquely identified by a primary key.

For example, when an administrator attempts to create a user with an already existing id of “201401111”, the system returns an error showing that duplicate key value violates unique constraint.



Referential integrity



Referential integrity ensures that the references of foreign keys in a table are valid so that its relationship with other tables remains intact.

Referential integrity is guaranteed as most forms in ATS use selection menus to restrict options to valid references. For example, administrators can only enrol students into existing subjects.

Domain integrity

Domain integrity ensures that the values stored in a database column adhere to a specific format or type.

For example, the grade of assignments must stored as an integer. On client-side, teachers can only input numbers but not text in the input field.

Making a Major Design Change

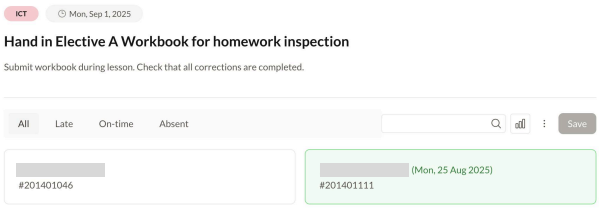
During ATS's development, some issues with the web page design and database schema were identified. Changes were made accordingly.

1. Grid vs Table for assignment status

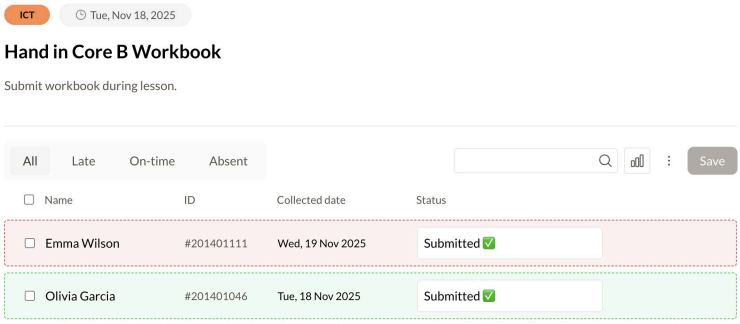
In the early versions of ATS, the student's assignment statuses were displayed in a grid format. Subject monitors click on the cards to toggle the status between submitted and not submitted.

During testing, the grid design proved to be inflexible (cards have to be redesigned when incorporating new information) and the toggle mechanism was difficult to understand. More importantly, it was not apparent how subject monitors can mark a student as absent.

Following the evaluation, a table design was implemented, similar to the "All Users" section in the Administrator View. It became much easier to adjust the view for monitors and teachers alike.



(Before)



(After)

2. Include year in subject id

Previously, the subject ID did not include the academic year. It followed the format grade/class-name-block (e.g. "g11-ict-2"). However, I later realized that the same ID would be used for different academic years. This would violate entity integrity, i.e. the primary key must uniquely identify each record.

The subject ID now includes the academic year: year-grade/class-name-block (e.g. "2526-g11-ict-2").

GitHub

Every change to the source code was documented in a GitHub repository.

https://github.com/CatMCGT/official-assignment-tracking-system

Potential Improvements to the System

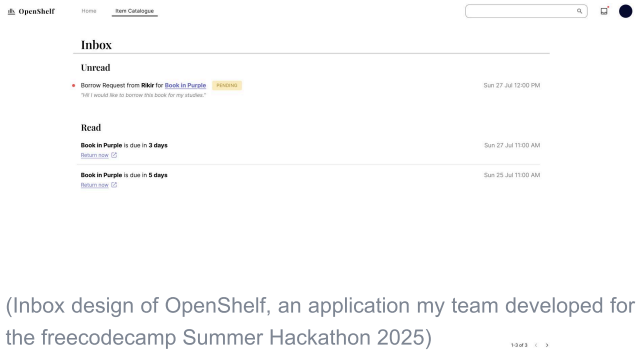
The Assignment Tracking System has its strengths and limitations. The following is a list of possible extensions to the web app's scope.

1. Notification System

Students may be notified about upcoming deadlines, when a teacher has returned a graded assignment, or even about scheduled system maintenance.

The notification system can follow this flow: trigger event (user action/ automatic) → backend processing → queueing (handled by a message queue) → delivery → UI display. A separate table would be needed to store read/unread status.

The challenge lies in constructing a scheduler service that pushes deadline notifications to the message queue 3 days before or a day before an assignment is due. Delayed notifications are the key reason for implementing a notification system in the first place. However, the recommended approaches seem to involve external dependencies that are well beyond my current capabilities. Given time, I may be able to come up with a viable plan and build a notification system of my own!



(Inbox design of OpenShelf, an application my team developed for the freecodecamp Summer Hackathon 2025)

2. Export Reports in PDF Format

Teachers may export tables and graphs about student performance in PDF format. At the moment, users may print out any web pages with the built-in command + P, or simply take a screenshot of the graphs. But it is, of course, best if the application can re-format the tables for printing.

On a similar note, ATS can also provide more data analytic tools beyond what is currently provided.

3. Light/ Dark Mode

Users may switch between light and dark mode. TailwindCSS includes a "dark" variant, which uses the "prefers-color-scheme" CSS media feature. Alternatively, the dark theme can also be driven by a CSS selector, supporting manual dark mode toggling.

4. User Settings

Users may customise their profile picture, display name, email, notification preferences, and colour scheme (light/ dark mode).

5. Improved Accessibility Support

ATS should include more accessibility features. For example, users less adept at using mouse should be able to use the keyboard exclusively to access all features and contents. When the tab key is pressed, the cursor should move to all of the menus, hyperlinks and input boxes in a logical order. Additionally, ATS should be compatible with common screen readers and screen magnifying software by leveraging aria labels.

I have made considerable efforts to comply to basic accessibility rules, including but not limited to the fact that all interactive elements should be focusable and in an logical order. However, I have not tested ATS with a screen reader.

Conclusion

Three years ago, when I first learned how to program websites, I envisioned to build products that would serve an audience and better their lives. With HKDSE SBA and ATS, my vision was realised.

I began confident in my capabilities: industry standard was what I was measuring ATS against. But during its development I met numerous roadblocks that exposed gaps within my knowledge, revealing just how naive I was to chase that goal! Powerful frameworks have their benefits, but the superfluous functionalities can hinder complete understanding of the architectures under the hood. Still, I am more than proud of what I have achieved in ATS.

With decades worth of web computing history and knowledge to digest, there is still a long journey ahead. But the road forward is clear: be creative, be brave, be willing to learn and develop, one website at a time.

References

1. OWASP Cheat Sheet Series. *Choosing and Using Security Questions Cheat Sheet*. https://cheatsheetseries.owasp.org/cheatsheets/Choosing_and_Using_Security_Questions_Cheat_Sheet.html
2. Developer Rants. *How to insert a json array into a postgresql table with a single query*. Retrieved October 17, 2025, from https://medium.com/developer-rants/how-to-insert-a-json-array-into-a-postgresql-table-with-a-single-query-9afd10ac4e88
3. W3Schools. *MYSQL CROSS JOIN Keyword.* https://www.w3schools.com/mysql/mysql_join_cross.asp