Analysis

**1 Introduction**

With the current situation with the outbreak of Covid-19, many schools and other educational facilities have moved to online education. This comes with several problems, the main one being homework, and the ability to track students that decide to, or cannot complete it. There are several solutions to this issue on the market, two of them being by Apple, with the now discontinued iTunesU, and the more modern Apple Schoolwork. Unfortunately, the latter solution sometimes does not work well from my experience, which leads to confusion and unnecessary delays. I have come up with a more reliable solution, which mimics the main functionality of Apple’s Schoolwork app, named Classwork. Classwork is a web application, which allows a headmaster to manage multiple schools, a teacher to manage subjects and assignments, and students to submit their work to the teacher.

**Complete source code can be found at** [**https://github.com/frogstair/classwork**](https://github.com/frogstair/classwork)

**1.1 Research**

Some research was made before I started working on Classwork. Mainly, I asked students from my school what their complaints for Schoolwork were, and other suggestions. The most popular suggestions were to make it more reliable and faster, so I focused on that. To increase reliability, I looked into ORM (Object-Relational Mapper) systems for my programming language of choice, Go. I picked it because it is considered modern, fast, reliable and concurrent, so its features fully address the problems of Schoolwork. The ORM I picked is called GORM, and it greatly simplifies working with a database, allowing for a more secure and stable operation. The final part of my research consisted of looking for a suitable database management system. In the end I settled for PostgreSQL because it is efficient, concurrent and supports standard SQL queries.

***1.1.1 The Go Programming Language***

*Go is a statically typed, compiled programming language designed at Google by Robert Griesemer, Rob Pike, and Ken Thompson. Go is syntactically similar to C, but with memory safety, garbage collection, structural typing, and CSP-style concurrency. The language is often referred to as Golang because of its domain name, golang.org, but the proper name is Go.*

*From Wikipedia*

I picked Go for my project because I was familiar with it before, so it was a first choice, and because it has many other advantages, such as it being compiled, so it has very high performance, statically typed, so it prevents type errors from occurring, and memory safe, which means it won’t create any unexpected errors. A snippet of Go code looks like this

|  |
| --- |
| func main() {  err := godotenv.Load() // Load values from .env  if err != nil { // Check if an error occurred  log.Fatalln("Could not find .env file!")  }  rand.Seed(time.Now().UnixNano()) // Generate a random seed  wg := sync.WaitGroup{} // Create a waitgroup to synchronize goroutines  wg.Add(2)  go run(&wg) // Run function on a new goroutine  wg.Wait() // Wait for the waitgroup to exit } |

***1.1.2 GORM***

*The fantastic ORM library for Go that aims to be developer friendly.*

*From the GORM documentation*

GORM is an Object-Relational Mapper that was created for Go. It automatically builds a safe to use SQL query and tracks relationships between tables. It integrates well with Go as well. A typical use case looks like this

|  |
| --- |
| // Retrieve a user by ID and check for errors err := database.Where("id = ?", userID).First(user).Error if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Invalid user ID"  return 404, resp  }  return 500, "Database error" } |

**1.2 Objectives**

The objectives for the project were pretty clear: create a web application that can host three types of users, each with their own permissions

**Headmaster:**

* Create schools which host students and teachers (one headmaster can manage multiple schools)
* Add teachers and students to the school

**Teacher:**

* Create subjects and add students to said subjects (each teacher can manage multiple subjects)
* Create assignments for students to complete

**Student:**

* Upload a file to an assignment

And with all of that must come a web-based user interface written in JavaScript and HTML

**1.3 How Go works**

Before I begin talking about Classwork, I will talk about how Go works, and describe some functionality that might not exist in other programming languages

**1.3.1 Structs**

Structs are Go’s way of implementing Object Oriented Programming. Structs can have properties, known as fields, attached functions, some form of inheritance, and a feature unique to Go known as tags. They are very useful for grouping data together in a meaningful way, and due to Go’s powerful reflection system, they can also be used to encode data to JSON and other types. A struct is declared as a type in Go, with its properties specified in brackets. Fields and functions that begin with a capital letter are the equivalent to public properties in OOP, and any other character are private and cannot be accessed by external functions

|  |
| --- |
| type customer struct {  ID int  Name string    email string  password string } |

The ID and Name fields are public, and email and password fields are private. A function attached to a struct looks like this

|  |
| --- |
| func (c customer) BuyProduct() {  // Code.... } |

Here, the function is accessible to any other parts of the code, because it starts with a capital letter, and the customer is referred to as c in this case. To call a struct function, this syntax is used

|  |
| --- |
| type customer struct {  ID int  Name string   email string  password string }  func (c customer) BuyProduct() {  // Code.... }  func main() {  cust := customer{}    cust.BuyProduct() } |

Structs also have tags which are used extensively in the code. Tags are indicated by `` for each field, and they describe what a function should do if it encounters that tag. For example, if I wanted to encode the customer struct as JSON

|  |
| --- |
| type customer struct {  ID int `json:"id"`  Name string `json:"name"`  Email string `json:"email"`  Password string `json:"-"` } |

json:”id” indicates that the JSON representation should represent this field as id, and not ID. A dash means that the field should be removed from the JSON representation.

GORM also has special tags for structs, and many tags can be combined together, and not all fields need to have the same amount of tags.

|  |
| --- |
| type customer struct {  ID int `gorm:"primaryKey" json:"id"`  Name string `json:"name"`  Email string `json:"email"`  Password string `json:"-"`  New bool `gorm:"-"` } |

Hare, the ID field will be set as primary key in GORM, and set as id in JSON, if no tag is present, then the default settings are used, such as the new field, which will be encoded as New by the JSON generator. It will also be ignored by GORM, and not appear in any tables.

**1.3.2 Make and New**

There are two types of data structures in Go: static and dynamic. Static structures are similar in declaration to other programming languages, for example

|  |  |
| --- | --- |
| arr := []int{1, 2, 3, 4, 5} |  |

declares an array of fixed size 5.

If I want a dynamic array structure, then I would use the make function, which allocates memory for that type. It takes two arguments, the first one being the type, and the second one being the size of the structure. The function returns a pointer to the newly created data

For example, if I want to create a dynamic array of size 5, then I would use

|  |
| --- |
| arr := make([]int, 5) // type is \*[]int |

If I want to create a struct, then I will use the new function which creates a new instance of a struct, and returns a pointer to it. It acts in the same way as the make function, but without the size argument, because only one instance is created

|  |
| --- |
| type customer struct {  id int  name string } cus := new(customer) |

**1.3.3 Goroutines**

Go is built to be concurrent, and implements a very easy to use system of coroutines, known as goroutines. They are very lightweight, up to 500 000 different goroutines can run at once without any slowdown.

A goroutine is called using the go keyword

|  |
| --- |
| go SomeTestFunction(arg1, arg2) |

which will make it run independently of the parent goroutine, until it or the parent goroutines stop.

It can also be used with anonymous functions

|  |
| --- |
| go func(a, b) {  doSomeOperation(a)  doAnotherOperartion(b)  }(1, 2) |

A downside of goroutines is that they cannot return anything, but through the use of channels or closure they can change some value outside of it.

**1.3.4 Channels**

Channels are built-in data types in Go that resemble queues, and are safe for concurrency. They can be declared with a chan keyword, followed by the type the channel can transmit

|  |
| --- |
| signals := make(chan bool, 10) |

This declares a channel named signals that can only transmit booleans

Channels are a dynamic data structure so they must be declared with the make function.

To interact with a channel, a left pointing arrow is used. For example this code will add a new element to the queue, and then place it in a new variable

|  |
| --- |
| signals := make(chan bool, 1) signals <- true sig := <-signals |

In the end the sig variable will have the value true

Since channels are queues, the first element to be put into them, will be the first one to be removed

For example

|  |
| --- |
| numbers := make(chan int, 2) numbers <- 10 numbers <- 15 num1 := <-numbers // 10 num2 := <-numbers // 15 |

**1.3.5 Select**

Select is a way to interact with channels. Select is similar to a switch case, but it runs certain sections of code, depending on which channel has any data available in it. Select runs only once and returns immediately, that is why it is common to find select in an infinite for loop

|  |
| --- |
| chan1 := make(chan int, 10) chan2 := make(chan int, 10) quit := make(chan bool, 1)  for {  select {  case x <- chan1:  // do something with a value from chan1  case x <- chan2:  // do something with a value from chan2  case \_ <- quit  break  // we don't care what we got from the quit channel,  // we only care we have to exit when there is something in it  } } |

**1.3.6 The go.mod file**

Go uses packages to make code modular, and classwork uses many libraries that come in the form of packages. At compilation time, the Go compiler needs to collect code from those packages, and install any missing ones. The purpose of go.mod is to track which packages are used, which ones are installed, and which one need to be downloaded or updated.

The go.mod file for classwork looks like this:

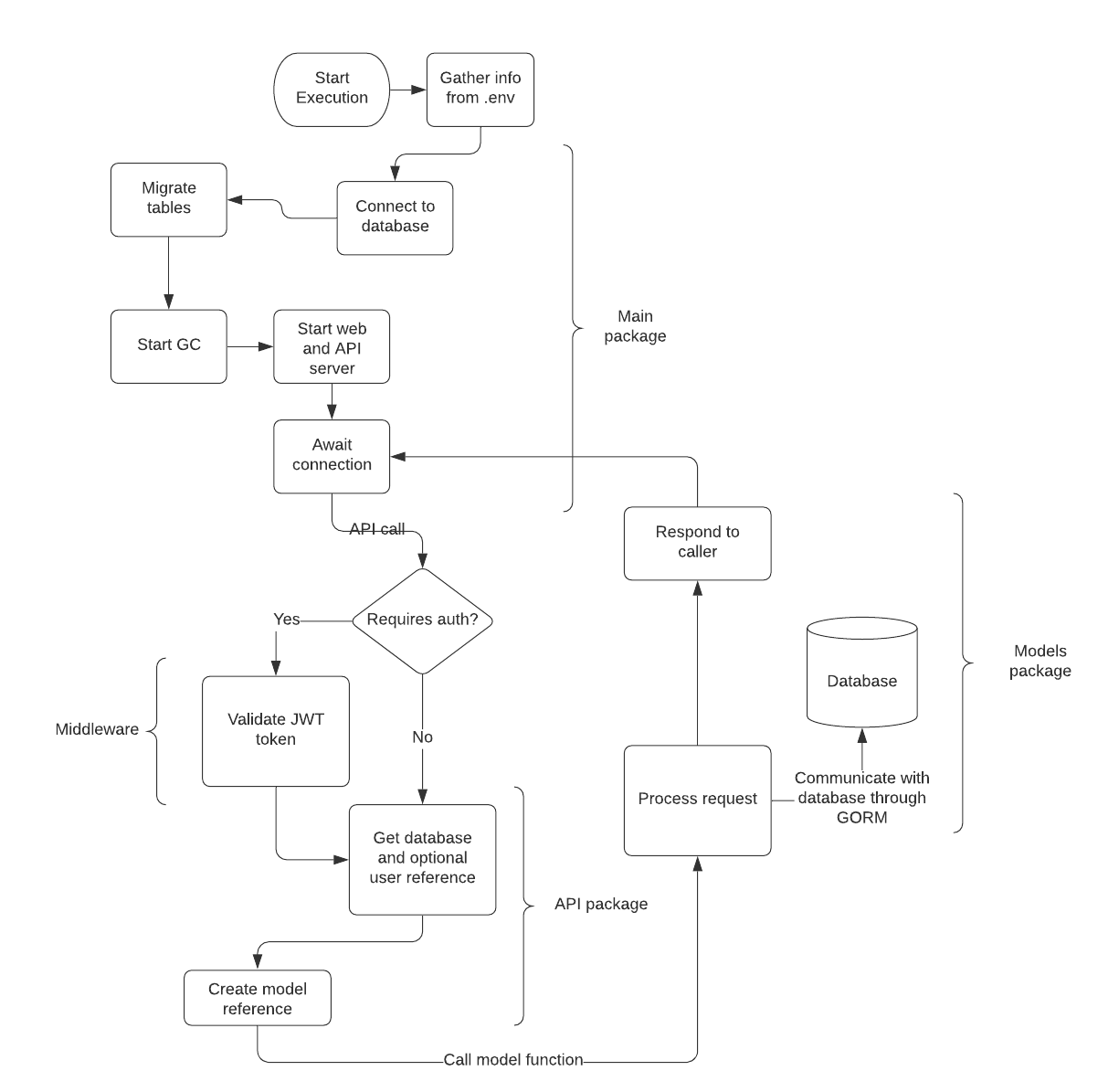
|  |
| --- |
| module classwork  go 1.15  require (  github.com/dgrijalva/jwt-go v3.2.0+incompatible  github.com/fatih/structs v1.1.0  github.com/gin-gonic/gin v1.6.3  github.com/jinzhu/gorm v1.9.16  github.com/jinzhu/now v1.1.1 // indirect  github.com/joho/godotenv v1.3.0  github.com/kr/pretty v0.1.0 // indirect  github.com/lib/pq v1.3.0  github.com/segmentio/ksuid v1.0.3  github.com/stretchr/testify v1.5.1 // indirect  golang.org/x/crypto v0.0.0-20200622213623-75b288015ac9  gopkg.in/check.v1 v1.0.0-20180628173108-788fd7840127 // indirect ) |

The syntax is not valid Go syntax, because it is not real Go code, just a way to track dependencies.

When Go finishes installing all dependencies, a go.sum file is produced, which contains the checksums for all the packages and files. The file is pretty large and information dense, and is not very useful to the programmer, so it will not be replicated here.

**Note: The code presented in later sections is not runnable as-is, as the code in this document is combined from multiple files that the program is split into, and all import statements and boilerplate code have been omitted. For a runnable working version please visit** [**https://github.com/frogstair/classwork**](https://github.com/frogstair/classwork)

**2. General Operation**

[****](https://lucid.app/documents/edit/924f1894-871b-4ffb-883b-45fdeccf2e19/0?callback=close&name=docs&callback_type=back&v=976&s=612)

*Figure 1.*

Figure 1 demonstrates the basic working principle behind every API call, excluding serving HTML and uploading/downloading files, and omitting details like error handling. As a means of communication I will use JSON, which is incredibly common, and Go has a built-in library that helps parse and create JSON from and to Go structs.

**2.1 The Server**

**2.1.1 HTTP/S**

The HTTP/S engine that I decided to use is a framework called Gin, which is considered one of the fastest and most flexible frameworks due to it being written entirely in Go, and it using Go’s native low-level APIs. It also supports a variety of useful features such as middleware, which I will be using to authorize the user and access the database. It’s API is very simple, and can be utilized in many ways, allowing the user to group their endpoints, and handling all the parameters a user might give to the server. I, however, did not implement HTTPS, because it requires owning a domain, to which a certificate should be attached, which I do not have, but it could easily be implemented if such a requirement appears.

The code used to initialize and run the server looks like this:

|  |
| --- |
| func main() {  err := godotenv.Load() // Load all values from the .env file  if err != nil { // If an error occurred then exit  log.Fatalln("Could not find .env file!")  }   rand.Seed(time.Now().UnixNano()) // Seed random number generator   wg := sync.WaitGroup{} // Create a waitgroup to run everything asynchronously  wg.Add(1) // Only one function will be running asynchronously   go run(&wg) // Run the function   wg.Wait() // Wait for the program to exit }  func run(wg \*sync.WaitGroup) {  db := database.GetPostgres() // Get a database connection  defer db.Close() // Close the database connection when the function returns  defer wg.Done() // Mark the function as done when the function returns  defer func() { garbage.Quit <- true }() // Quit the GC once the program exits   gin.SetMode(gin.ReleaseMode) // Set gin's mode to "release" to remove any logs  g := gin.New() // Create a new router  g.Use(gin.Recovery()) // Use the recovery middleware to recover from functions that may have crashed  g.Use(m.Postgres) // Use the postgres middleware to inject the database connection into every function  g.NoRoute(pages.NotFound)  g.NoMethod(pages.NoMethod)   // Create routes to serve all the html pages  // Could have made a smarter system than that  // but it created router conflict  g.GET("/register", pages.ServeRegister)  g.GET("/login", pages.ServeLogin)  g.GET("/login/pass", pages.ServeLoginPassword)  g.GET("/dashboard", pages.ServeDashboard)  g.GET("/school", pages.ServeSchool)  g.GET("/subject", pages.ServeSubject)  g.GET("/assignment", pages.ServeAssignment)  g.Static("/static", "./web/static")   // All the API routes and their handlers  apiGroup := g.Group("/api")   logoutGroup := apiGroup.Group("/logout")  logoutGroup.POST("/", m.ValidateJWT, api.Logout)   logGroup := apiGroup.Group("/login")  logGroup.POST("/", api.Login)  logGroup.POST("/pass", api.GenerateOTC)   regGroup := apiGroup.Group("/register")  regGroup.POST("/", api.Register)  regGroup.GET("/email", api.EmailValid)   dbdGroup := apiGroup.Group("/dashboard")  dbdGroup.GET("/", m.ValidateJWT, api.GetDashboard)   schGroup := apiGroup.Group("/school")  schGroup.POST("/", m.ValidateJWT, api.AddSchool)  schGroup.DELETE("/", m.ValidateJWT, api.DeleteSchool)  schGroup.GET("/", m.ValidateJWT, api.GetSchool)  schGroup.GET("/student", m.ValidateJWT, api.GetStudents)   schGroup.POST("/teacher", m.ValidateJWT, api.AddTeacher)  schGroup.DELETE("/teacher", m.ValidateJWT, api.DeleteTeacher)   schGroup.POST("/student", m.ValidateJWT, api.AddStudent)  schGroup.DELETE("/student", m.ValidateJWT, api.DeleteStudent)   subGroup := schGroup.Group("/subject")  subGroup.POST("/", m.ValidateJWT, api.AddSubject)  subGroup.DELETE("/", m.ValidateJWT, api.DeleteSubject)  subGroup.GET("/", m.ValidateJWT, api.GetSubject)  subGroup.POST("/students", m.ValidateJWT, api.AddStudentSubject)   assgnGroup := subGroup.Group("/assignment")  assgnGroup.POST("/", m.ValidateJWT, api.NewAssignment)  assgnGroup.GET("/", m.ValidateJWT, api.GetAssignment)  assgnGroup.POST("/complete", m.ValidateJWT, api.CompleteAssignment)   g.Use(garbage.AddCollectorToContext)  go garbage.Run()   g.Static("/files", "./files")   fsgroup := g.Group("/files")  fsgroup.POST("/", api.CreateFile)   // Get the address and port from the environment  address, port := os.Getenv("ADDRESS"), os.Getenv("PORT")   // Print a log  log.Printf("Running on %s:%s\n", address, port)   // Run the server  g.Run(address + ":" + port) } |

**2.1.2 The Database**

The database server I decided to use is called PostgreSQL. It has several advantages such as being multithreaded, light on memory usage, fully open source and highly configurable. The database structure is completely decided upon by GORM, so it may not be perfect, and I had to create foreign key relationships myself

The database structure is as follows:

|  |  |
| --- | --- |
| Table name | Description |
| users | User info |
| schools | School info |
| subjects | Subject info |
| assignments | Assignments the teacher has created |
| requests | Upload requests for the assignments |
| request\_uploads | Associates the files that students have uploaded to each request |
| assignment\_files | Files attached to each assignment |
| school\_students | Relate users that are students and schools |
| school\_teachers | Relate users that are teachers and schools |
| subject\_assignments | Relate assignments and subjects |
| subject\_students | Relate subjects and users that are students |
| assignments\_completed | Relates students that completed the assignment to assignments |

When I use GORM, I don’t have to interact with any of these tables, and all the interactions are handled by the library itself.

**Users table**

|  |  |
| --- | --- |
| Column name | Description |
| id | Contains the user’s ID |
| first\_name | User’s first name |
| last\_name | User’s last name |
| email | User’s email |
| password | The hash of the user’s password |
| token | The user’s access token |
| perms | A collection of flags that determine who the user is |
| pass\_set | A flag that determines if the user’s password is set, used to indicate that a user needs to set a password |
| one\_time\_code | A one time code used to set the password for the user |
| school\_id | Indicates to which school a student belongs, otherwise empty. Used to reduce amount of calls to database |

**Schools table**

|  |  |
| --- | --- |
| Column name | Description |
| id | The ID of the school |
| user\_id | The ID of the headmaster that created the school |
| name | School name |

**Subjects table**

|  |  |
| --- | --- |
| Column name | Description |
| id | ID of the subject |
| teacher\_id | ID of the teacher that created the subject |
| school\_id | ID of the school that the subject is in |
| name | Subject name |
| num\_students | Amount of students |

**Assignments**

|  |  |
| --- | --- |
| Column name | Description |
| id | Assignment ID |
| teacher\_id | ID of the teacher that assigned it |
| subject\_id | ID of the subject that the assignment is in |
| name | Name of the assignment |
| text | Assignment description |
| time\_due | Time the assignment is due |
| time\_assigned | Time the assignment was created |

**Connection to the database**

The connection to the database is established at the beginning of the execution of the program, and the function responsible for it will either create a connection to the database, or returns an existing connection.

*File: database/postgres.go*

|  |
| --- |
| package database  import (  "fmt"  "log"  "os"   m "classwork/models"   "github.com/jinzhu/gorm" )  var db \*gorm.DB // Global variable that keeps the database connection var connected = false // Flag if the connection is established  // GetPostgres initializes the connection to a postgres database or returns an existing connection func GetPostgres() \*gorm.DB {   if connected { // If the connection is already established then return the connection  return db  }   host := os.Getenv("DB\_ADDR") // Get variables from the environment  port := os.Getenv("DB\_PORT")  role := os.Getenv("DB\_ROLE")  name := os.Getenv("DB\_NAME")  var err error // Variable to contain the error   cstring := fmt.Sprintf("host=%s user=%s dbname=%s sslmode=disable port=%s",  host, role, name, port) // Connection string, formed using a format string   db, err = gorm.Open("postgres", cstring) // Open a "postgres" database connection  // If an error occurred then panic and exit the program  if err != nil {  panic(fmt.Sprintf("\n===========\ncannot establish database connection: \n%s\n===========", err))  }  // Set the flag  connected = true  // Remove unnecessary database logs  db.LogMode(false)   // Print a success log  log.Println("Connected to database")   // Synchronize all the tables in the code with the database  db.AutoMigrate(&m.User{}, &m.School{}, &m.Subject{}, &m.Assignment{}, &m.Request{}, &m.AssignmentFile{}, &m.RequestUpload{})  log.Println("Migrated tables")   // Return the newly established connection  return db }  // Disconnect closes the database connection func Disconnect() {  // Close the connection to the database  db.Close()  // Set the flag to false  connected = false } |

**2.1.4 Authentication**

Authentication is the process of verifying a user’s identity. In Classwork this is done using a JWT (JSON Web Token). JWT is a self-contained key that a user uses to present their claims. For example, a user could claim that he is a headmaster, and request access to a headmaster-only resource. It is the server’s job to validate that key and present the user with information or reject it.

The JWT works the following way:

JWT is represented as an encoded string, separated by fullstops. This is an example of a JWT token that Classwork would generate:

**eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MTcyMDk4NTYsImlkIjoiMW9JcE5IRTdXT0VtZGxDR3VJc1l1NFI3TEdvIn0.hQQaLpD2koqD1\_o3BXSJtRgfa7tGFVmxwJW6CPw3P-x3PM9Ve9Jr80LCnAqNXa9sHdA0\_f4EqfdmiNJSyO4BBw**

It contains three sections, the first one being **eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9** and delimited by a fullstop. This is called the header, and is actually a Base64 encoded JSON string, which contains the token type, always JWT, and the encryption method, which for Classwork is HS512. The decoded string looks like this: {"alg":"HS512","typ":"JWT"}.

The next section is **eyJleHAiOjE2MTcyMDk4NTYsImlkIjoiMW9JcE5IRTdXT0VtZGxDR3VJc1l1NFI3TEdvIn0.** It contains the claims the client has made. It is the information that can identify the client, such as the User ID. In Classwork the claims made by the client are only the User ID, and when the token expires, i.e becomes invalid. The rest of the information is pulled from the database, so the user doesn’t need to provide their claims. The decoded information looks like this: {"exp":1617209856,"id":"1oIpNHE7WOEmdlCGuIsYu4R7LGo"}.

The third and final section is **hQQaLpD2koqD1\_o3BXSJtRgfa7tGFVmxwJW6CPw3P-x3PM9Ve9Jr80LCnAqNXa9sHdA0\_f4EqfdmiNJSyO4BBw**. In a valid key it contains the same data as the second part, except it is encrypted with the algorithm provided in the header, and a key that is stored on the server. When a server validates this information, this section is decoded and the contents are compared.

The function to validate a user’s token looks like this:

*File: models/tokens.go*

|  |
| --- |
| // ParseToken parses a token and returns the associated user func ParseToken(tokstr string, db \*gorm.DB) (int, \*util.Response) {   resp := new(util.Response) // Create a response to the user   tok := &Token{} // Allocate memory for a new token  token, err := jwt.ParseWithClaims(tokstr, tok, // ParseWithClaims function parses the token with a function that returns the key  func(t \*jwt.Token) (interface{}, error) { // Since I only store the key in an environment variable, get the key from there  return []byte(os.Getenv("JWT\_SECRET")), nil // Just return it  })   if !token.Valid { // If the token is invalid (i.e invalid syntax)  if err != nil {  resp.Data = nil  resp.Error = "token is invalid"   return 401, resp // Respond with corresponding message and error code  }  }   user := new(User) // Retreive a user from the database  err = db.Where("id = ?", tok.ID).First(user).Error  if err != nil { // Error handling  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "token does not correspond to user"   return 401, resp  }  log.Printf("Database error: %s\n", err.Error())  return util.DatabaseError(err, resp)  }   if user.Token != tokstr { // In the end, if the tokens dont match  resp.Data = nil  resp.Error = "token does not correspond to user"   return 401, resp  }   resp.Data = user // Return the user variable  resp.Error = ""  return 200, resp // Success } |

And the function to generate such a token is as follows:

*File: models/tokens.go*

|  |
| --- |
| // TokenValidity specifies how long a token is valid var TokenValidity = int64(2592000) // 30 days  // Token is the model for a JWT token with custom claims type Token struct {  jwt.StandardClaims // Standard JWT claims (in my case only the `expires` field)   ID string `json:"id"` // The user's ID } // CreateToken will create a token string for a given ID func CreateToken(id string) string {  token := new(Token) // Create a new token   token.ID = id // Set the ID in the token to the passed ID   token.ExpiresAt = time.Now().Unix() + TokenValidity // Set expiration time (current time + how long a token is valid)   jwtToken := jwt.NewWithClaims(jwt.SigningMethodHS512, token) // Create a new JWT token instance signed with HS512  tokenString, \_ := jwtToken.SignedString([]byte(os.Getenv("JWT\_SECRET"))) // Encrypt the token with the secret from an environment variable   return tokenString // Return the string to be sent to the user } |

**2.1.5 Environment variables**

Classwork makes use of environment variables to securely store and use variables. They are stored in a .env file in the same directory as the executable, and the first thing the program does is search for that file and import it. In a normal use case, the file would be omitted from programs like git, to prevent security breaches. The # symbol represents a comment and its contents are ignored

*File: .env*

|  |
| --- |
| *PORT=5000 ADDRESS= # empty to allow all connections  DB\_ADDR=127.0.0.1 DB\_PORT=5432 DB\_ROLE=frogstair DB\_NAME=classwork  JWT\_SECRET=3af8cdd15bb6009585c025d4f826dfa06a5fe8943e7d9e1a3b529cfc21baacb2ca41288d7a5d31ccdab5648022ae55bf9880bb829a19c48f95 # MUST REPLACE, KEY IS IN THE OPEN* |

In the main function the I use the godotenv package to read the file

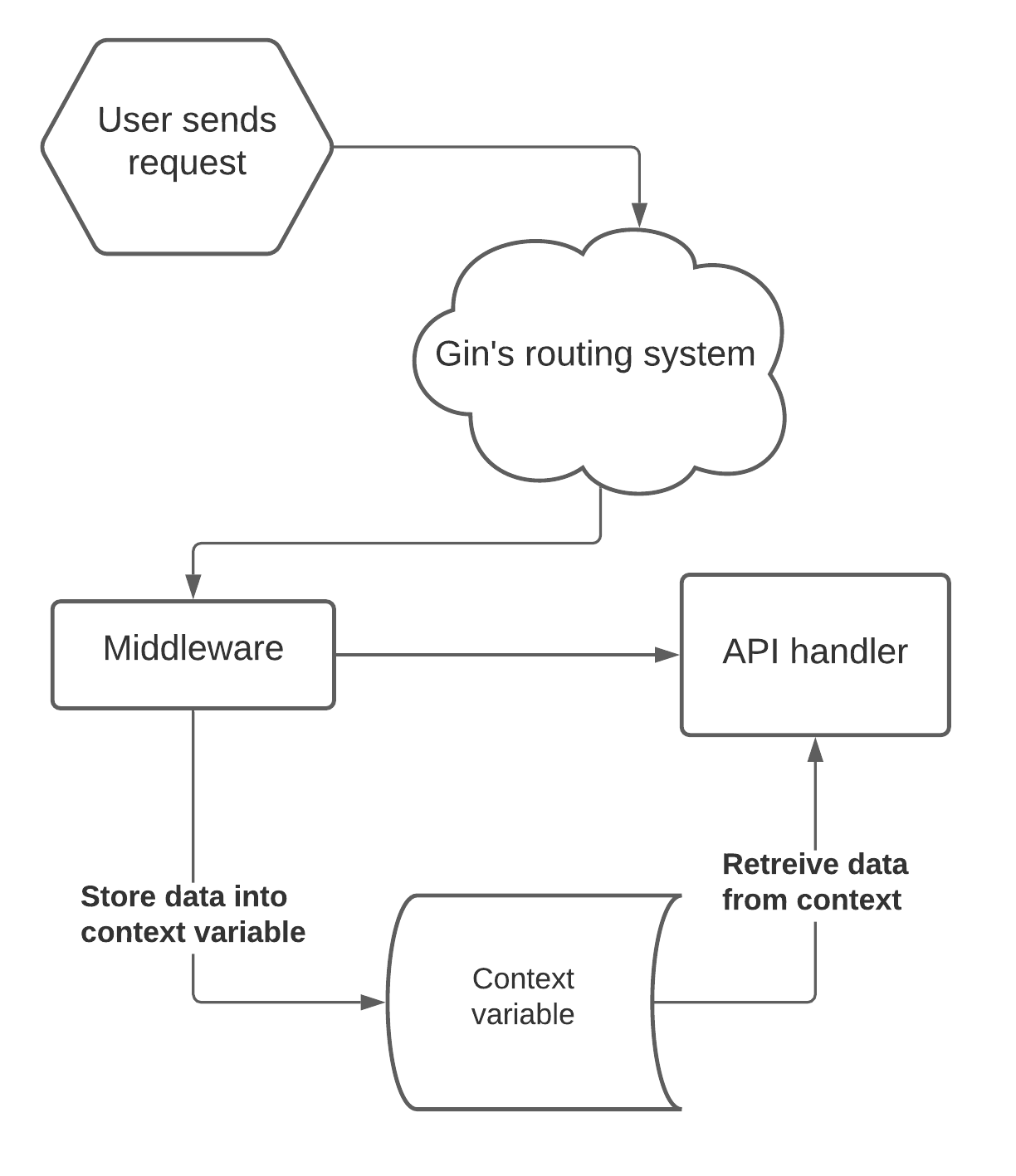
*File: main.go*

|  |
| --- |
| ..... err := godotenv.Load() // Load all values from the .env file if err != nil { // If an error occurred then exit  log.Fatalln("Could not find .env file!")  } ..... |

The variables specified in the .env file are then available to the whole program through os.GetEnv().

**2.1.6 Middleware**

Classwork uses two middleware functions, one to serve a database connection to another function, and another to validate a JWT token. These middleware functions are run before the actual handlers, saving any necessary data into the context variable, which can then be retrieved by later functions. This is the general operation of the functions:

[](https://lucid.app/documents/edit/70e5db97-a799-474f-8774-e3628b27b8c8/0?callback=close&name=docs&callback_type=back&v=373&s=612)

To use middleware across all routes gin has a useful function Use which does exactly that

*File: main.go*

|  |
| --- |
| *..... g.Use(gin.Recovery()) // Use the recovery middleware to recover from functions that may have crashed g.Use(m.Postgres) // Use the postgres middleware to inject the database connection into every function .....* |

Middleware can also be injected into individual routes

|  |
| --- |
| dbdGroup := apiGroup.Group("/dashboard")  // Inject the ValidateJWT middleware into the dashboard route dbdGroup.GET("/", m.ValidateJWT, api.GetDashboard) |

The ValidateJWT middleware uses the JWT parsing function from earlier

|  |
| --- |
| // ValidateJWT validates the JWT token and places the user it belongs to in the context func ValidateJWT(c \*gin.Context) {   db, ok := c.Keys["db"].(\*gorm.DB) // Get database variable from context  if !ok { // If not found  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   tokstr, err := c.Cookie("\_tkn") // Get `\_tkn` from cookie  if err != nil { // If there is an error  c.JSON(400, gin.H{"error": "no token specified"})  c.Abort() // Doesnt run the next function in the chain  return  }   code, data := m.ParseToken(tokstr, db) // Parse token  if code != 200 {  c.JSON(code, gin.H{"error": data.Error})  c.Abort()  return  }   c.Set("usr", data.Data) // Set user context variable  c.Next() } |

And the small database middleware function

|  |
| --- |
| // Postgres attaches a database variable to a given context func Postgres(c \*gin.Context) {  db := database.GetPostgres() // Get existing connection/establish new connection  c.Set("db", db) // Set context variable  c.Next() } |

**3. API and models**

**3.1 /api/login and /api/login/pass**

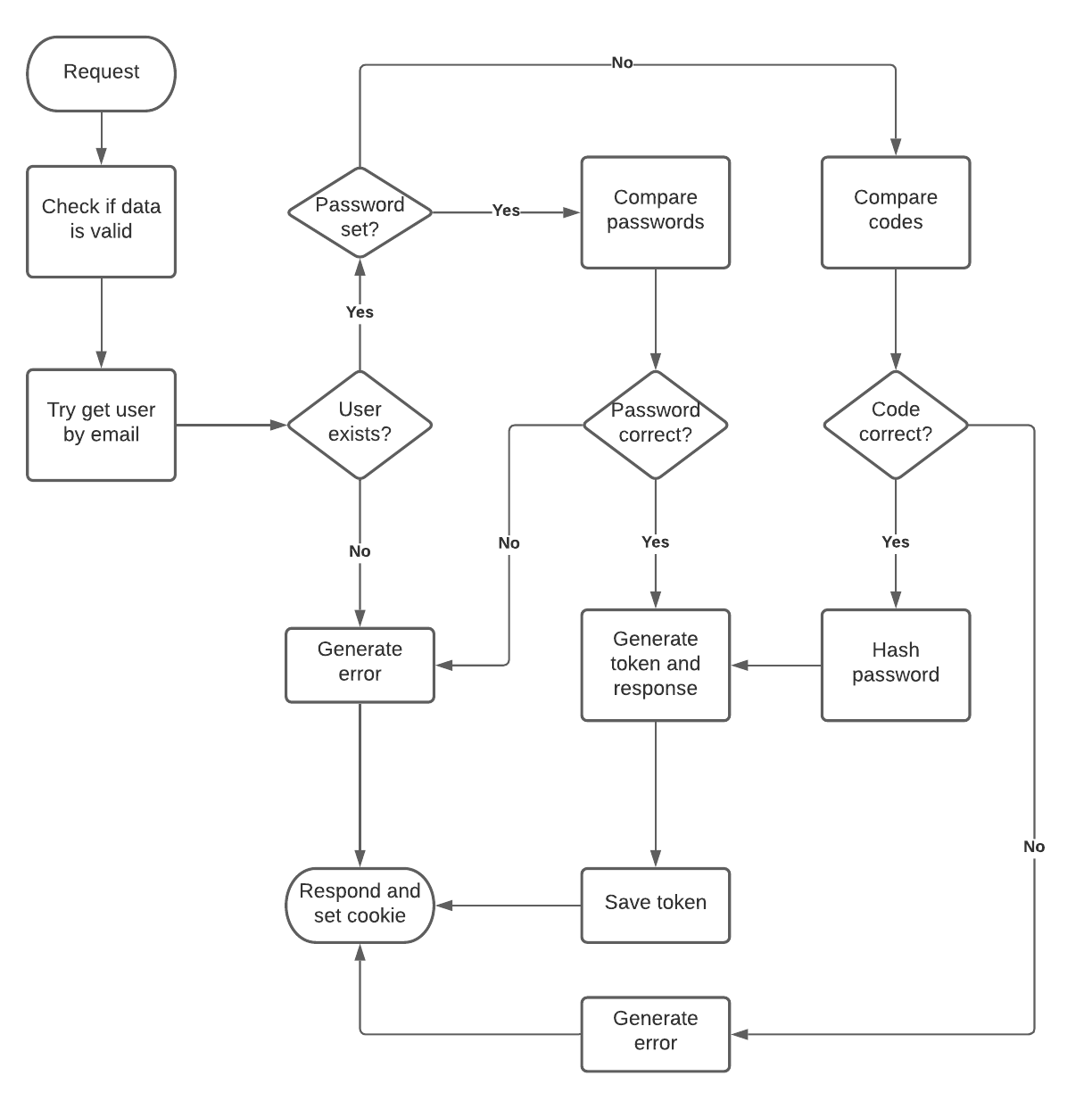
This route is used to login a user. The user must send out the following values in JSON format, which is almost exclusively used in communication with the server

|  |
| --- |
| {  "email": "email@example.com",  "password": "Password123",  "code": ""  } |

The code field is optional, and it’s role will be explained later in this document. Once the user makes a request, the Login function is run, to generate a token

**3.1.1 Login procedure**

The login procedure is described in the image below

[](https://lucid.app/documents/edit/cdee9af1-5875-4efe-8b79-766dbc403d27/0?callback=close&name=docs&callback_type=back&v=1010&s=612)

The code is used to identify users without a password, and is only available once. This only happens when a teacher or a student is added, and it allows them to set their own password the first time they login. This password is then saved and the code is deleted. The code is generated by the server by the /api/login/pass endpoint

The function to generate the code looks like this

|  |
| --- |
| // GenerateOTC creates an OTC for a user if their password is not set func GenerateOTC(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   otc := new(m.OTCCreate)  err := json.NewDecoder(c.Request.Body).Decode(otc)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, resp := otc.Create(db)  c.JSON(code, resp) } |

The OTCCreate model is a Go struct and contains the following field

|  |
| --- |
| // OTCCreate is the struct to check is a user has a password type OTCCreate struct {  Email string } |

And has a function

|  |
| --- |
| // Create creates an OTC for the user func (o \*OTCCreate) Create(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response) // Response placeholder  user := new(User) // User placeholder   err := db.Where("email = ?", o.Email).First(user).Error // Get the user   if err != nil { // Check for errors  if util.IsNotFoundErr(err) { // If user wasn't found  resp.Data = nil  resp.Error = "Invalid user"  return 404, resp  }  return util.DatabaseError(err, resp) // If any other error  }   if user.PassSet { // If the user has already tried to generate the code  resp.Data = nil  resp.Error = "resource gone"  return 410, resp  }   onetimecode := util.RandomCode() // Generate a random code   user.OneTimeCode = onetimecode // Save the code until the user uses it  err = db.Save(user).Error  if err != nil {  return util.DatabaseError(err, resp)  }   resp.Data = onetimecode // Return the code to the user  resp.Error = ""   return 201, resp } |

The login function itself performs the following actions

|  |
| --- |
| // Login creates a token for the user to use for future authentication func Login(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Get database variable from context  if !ok { // If an error occurred  c.JSON(500, gin.H{"error": "internal error"}) // Respond to user  panic("no database variable in context") // Crash with a message  }   loginuser := new(m.LoginUser) // Create an instance of a LoginUser  err := json.NewDecoder(c.Request.Body).Decode(loginuser) // Decode the users request  if err != nil { // If there is an error while decoding  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, data, tok := loginuser.Login(db) // Run the model function  if code == 200 {  c.SetCookie("\_tkn", tok, int(m.TokenValidity), "/api/", "", false, true) // Set a cookie on the user side  }   c.JSON(code, data) // Respond with data } |

The LoginUser model is another Go struct, which contains the following properties

|  |
| --- |
| // LoginUser is the model to create a token for the user type LoginUser struct {  Email string `json:"email"`  Password string `json:"password"`  Code string `json:"code"` } |

And has the following function

|  |
| --- |
| // Login will generate a token for the user func (l \*LoginUser) Login(db \*gorm.DB) (int, \*util.Response, string) {  resp := new(util.Response) // Create response placeholder  l.clean() // Clean the user input (remove trailing spaces and invalid characters)   user := new(User)  err := db.Where("email = ?", l.Email).First(user).Error // Get a user from the database with the entered email   if err != nil { // If an error occurred  if util.IsNotFoundErr(err) { // If no user was found  resp.Data = nil  resp.Error = "Invalid email or password"  return 401, resp, ""  }  \_, resp = util.DatabaseError(err, resp) // Any other errors  return 500, resp, ""  }   if user.PassSet { // If user already has a password  if !util.Compare(user.Password, l.Password) { // Check if password is correct  resp.Data = nil  resp.Error = "Invalid email or password"  return 403, resp, ""  }  } else { // Check the code  if l.Code == user.OneTimeCode { // Create the password if the code is correct  user.PassSet = true  user.Password = util.Hash(l.Password)  user.OneTimeCode = ""  } else { // Otherwise return nothing  resp.Data = nil  resp.Error = "Invatid OTC"  return 401, resp, ""  }  }   user.Token = CreateToken(user.ID) // Create a token for the user  err = db.Save(user).Error // Save the user with the token and password  if err != nil {  \_, resp = util.DatabaseError(err, resp)  return 500, resp, ""  }   resp.Data = true // Place success flag  resp.Error = ""   return 200, resp, user.Token // Respond with the token } |

**3.2 /api/register and /api/register/email**

When a user registers an account, they need to provide the following information

|  |
| --- |
| {  "email": "email@example.com",  "password": "Password123",  "first\_name": "Danila",  "last\_name": "Dudkin" } |

And the register function looks like this

|  |
| --- |
| // Register is used to register a new user func Register(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   reguser := new(m.RegisterUser)  err := json.NewDecoder(c.Request.Body).Decode(reguser)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, data := reguser.Register(db)  c.JSON(code, data) } |

The RegisterUser model is again a struct

|  |
| --- |
| // RegisterUser is the model to register a new user type RegisterUser struct {  Email string `json:"email"`  Password string `json:"password"`  FirstName string `json:"first\_name"`  LastName string `json:"last\_name"` } |

And performs the registration procedure

|  |
| --- |
| // Register registers the user into the database func (r \*RegisterUser) Register(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response) // Response placeholder   r.clean() // Remove trailing whitespace and invalid characters  valid, reason := r.validate() // Validate the input, and if invalid get reason  if !valid {  resp.Data = nil  resp.Error = reason  return 400, resp  }   hashed := util.Hash(r.Password) // Hash the user's password  user := new(User) // Create a user placeholder   user.ID = ksuid.New().String() // Create a GUID and fill in user information  user.Email = r.Email  user.FirstName = r.FirstName  user.LastName = r.LastName  user.Password = hashed  user.Perms = Headmaster  user.PassSet = true   err := db.Create(user).Error // Create the user in the database  if err != nil { // If an error occurred  if util.IsDuplicateErr(err) {  resp.Data = nil  resp.Error = "Email is taken"  return 409, resp  }  return util.DatabaseError(err, resp)  }   userResponse := struct { // Create a response for the user  FirstName string `json:"first\_name"`  LastName string `json:"last\_name"`  ID string `json:"id"`  }{user.FirstName, user.LastName, user.ID}   resp.Data = userResponse  resp.Error = ""   return 201, resp // Respond } |

During the registration process, the client sends out multiple requests to check for the email address, and checks if the email is taken. It sends a simple request to the server as a query parameter.

|  |
| --- |
| // EmailValid checks if the supplied Email is valid func EmailValid(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Get database variable from context   if !ok { // If an error occurred  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   email := new(m.Email)  email.Email = c.Query("email") // Get email from query  code, data := email.Valid(db) // Check if its valid  c.JSON(code, data) // Respond } |

|  |
| --- |
| // Valid returns if the email is valid func (e \*Email) Valid(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response  e.clean()  user := new(User)   valid, reason := e.validate() // Validate the email   if !valid { // If not valid  resp.Data = nil  resp.Error = reason  return 400, resp  }   err := db.Where("email = ?", e.Email).First(user).Error // Get a user by email   if err != nil { // If error occured  if util.IsNotFoundErr(err) {  resp.Data = true // We are actually looking for a not found error  resp.Error = ""  return 200, resp  }  return util.DatabaseError(err, resp)  }   resp.Data = false // If found then return error  resp.Error = ""   return 200, resp } |

**3.3 /api/dashboard**

This endpoint is used to retrieve the dashboard for all the users. It returns all available data for each user, no matter if they are a student, teacher, a headmaster or any combination of the 3

|  |
| --- |
| // GetDashboard gets the dashboard for a user func GetDashboard(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Get database from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // Get the user from the context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   code, resp := user.GetDashboard(db) // Get the dashboard for the user  c.JSON(code, resp) } |

The user model will be described later.

This is the models relating to the dashboard

|  |
| --- |
| // Dashboard is the user's dashboard type Dashboard struct {  User \*User `json:"user"`  Headmaster \*HeadmasterDashboard `json:"headmaster,omitempty"`  Teacher \*TeacherDashboard `json:"teacher,omitempty"`  Student \*StudentDashboard `json:"student,omitempty"` }  // HeadmasterDashboard contains headmaster information type HeadmasterDashboard struct {  Schools []\*School `json:"schools"` }  // TeacherDashboard contains headmaster information type TeacherDashboard struct {  SchoolID string `json:"school\_id"`  Subjects []\*Subject `json:"subjects"` }  // StudentDashboard contains headmaster information type StudentDashboard struct {  Subject []\*Subject `json:"subjects"` } |

The GetDashboard function is pretty large

|  |
| --- |
| // GetDashboard gets the users dashboard func (u \*User) GetDashboard(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response   dashboard := new(Dashboard) // Dashboard placeholder   if u.Has(Headmaster) { // If the user is a headmaster  hmDashboard := new(HeadmasterDashboard) // Placeholder   err := db.Where("user\_id = ?", u.ID).Find(&hmDashboard.Schools).Error // Get all the schools the headmaster owns  if err != nil {  return util.DatabaseError(err, resp)  }   dashboard.Headmaster = hmDashboard // Set the dashboard  }  if u.Has(Teacher) { // If the user is a teacher  tchDashboard := new(TeacherDashboard)   result := struct { // Result placeholder  SchoolID string  TeacherID string  }{}  db.Raw("select \* from school\_teachers where user\_id = ?", u.ID).Scan(&result) // Get all the schools the teacher is in   tchDashboard.SchoolID = result.SchoolID   err := db.Where("teacher\_id = ?", u.ID).Find(&tchDashboard.Subjects).Error // Get all the teacher subjects  if err != nil {  return util.DatabaseError(err, resp)  }   dashboard.Teacher = tchDashboard // Set the dashboard  }  if u.Has(Student) { // Is user is a student  stuDashboard := new(StudentDashboard)   db.Model(u).Association("Subjects").Find(&u.Subjects) // Get for each subject  for s, subject := range u.Subjects {  db.Where("subject\_id = ?", subject.ID).Order("time\_assigned desc").Limit(10).Find(&subject.Assignments)  u.Subjects[s] = subject  for a, assignment := range subject.Assignments { // Each assignment  db.Model(assignment).Association("Requests").Find(&assignment.Requests)  assignment.CompletedBy = []\*User{}  u.Subjects[s].Assignments[a] = assignment  for r, req := range assignment.Requests { // Each upload request  upl := make([]\*RequestUpload, 0)  req.Uploads = upl  db.Model(req).Association("Uploads").Find(&req.Uploads)  found := false  for \_, upload := range req.Uploads {  if upload.UserID == u.ID {  found = true  break  }  }   req.Complete = &found  req.Uploads = nil  u.Subjects[s].Assignments[a].Requests[r] = req  }  }  }   stuDashboard.Subject = u.Subjects   dashboard.Student = stuDashboard  }   dashboard.User = u // Respond with dashboard   resp.Data = dashboard  resp.Error = ""   return 200, resp } |

The user model looks like this, the above function is attached to it

|  |
| --- |
| // User is the internal representation of a user type User struct {  ID string `gorm:"primaryKey" json:"id"`  FirstName string `gorm:"not null" json:"first\_name"`  LastName string `gorm:"not null" json:"last\_name"`  Email string `gorm:"not null;unique" json:"email"`  Password string `json:"-"`  Token string `json:"-"`  Perms Role `gorm:"not null" json:"-"`  PassSet bool `json:"-"`  OneTimeCode string `json:"-"`  Subjects []\*Subject `gorm:"many2many:subject\_students" json:"students,omitempty"` } |

The user model also has this function

|  |
| --- |
| // Has returns if a user has a role func (u \*User) Has(r Role) bool {  role := u.Perms & r // Perform AND logical operation  return role == r // If the AND operation did not return 0, return true } |

The user can also be deleted, using this function

|  |
| --- |
| // Delete deletes a user func (u \*User) Delete(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response   err := db.Delete(u).Error // Delete user from database  if err != nil {  resp.Error = "internal error"  resp.Data = nil  log.Printf("Database error: %s\n", err.Error())  return 500, resp  }   resp.Data = true // success  resp.Error = ""   return 202, resp // Deleted } |

**3.4 /api/school**

This endpoint has various functions associated with it, with different request types.

The endpoint is described like this in the code

|  |
| --- |
| schGroup := apiGroup.Group("/school") schGroup.POST("/", m.ValidateJWT, api.AddSchool) schGroup.DELETE("/", m.ValidateJWT, api.DeleteSchool) schGroup.GET("/", m.ValidateJWT, api.GetSchool) schGroup.GET("/student", m.ValidateJWT, api.GetStudents) |

The school model looks like this

|  |
| --- |
| // School is the internal representation of the schools type School struct {  ID string `gorm:"primaryKey" json:"id"`  UserID string `gorm:"not null" json:"-"`  Name string `gorm:"not null" json:"name"`  Students []\*User `gorm:"many2many:school\_students" json:"students,omitempty"`  Teachers []\*User `gorm:"many2many:school\_teachers" json:"teachers,omitempty"`  Subjects []\*Subject `json:"subjects,omitempty"` } |

A POST request calls the add school function

|  |
| --- |
| // AddSchool adds a school func AddSchool(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Get database variable from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // Get the user to whom the school belongs  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Headmaster) { // Teachers and students can't add schools  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   newSchool := new(m.NewSchool) // Placeholder  err := json.NewDecoder(c.Request.Body).Decode(newSchool)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, resp := newSchool.Add(db, user) // Call function  c.JSON(code, resp) }  // NewSchool is the model to add a new school type NewSchool struct {  Name string `json:"name"` } func (n \*NewSchool) validate() (bool, string) {  if !util.ValidateName(n.Name) {  return false, "Name should be at least 4 characters"  }  return true, "" }  // Add adds a new school to the database func (n \*NewSchool) Add(db \*gorm.DB, user \*User) (int, \*util.Response) {  resp := new(util.Response) // Response placeholder   n.clean() // Remove trailing whitespace and etc   if valid, reason := n.validate(); !valid { // Check if valid  resp.Error = reason  resp.Data = nil  return 400, resp  }   school := new(School) // Create placeholder   school.Name = n.Name // Set all fields  school.UserID = user.ID  school.ID = ksuid.New().String()   err := db.Save(school).Error // Save the school  if err != nil {  return util.DatabaseError(err, resp)  }   schoolResp := struct { // Response struct  Name string `json:"name"`  ID string `json:"id"`  }{n.Name, school.ID}   resp.Data = schoolResp  resp.Error = ""  return 201, resp } |

A DELETE request calls this function

|  |
| --- |
| // DeleteSchool will delete a school from the database func DeleteSchool(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Database from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // User from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Headmaster) { // Teachers and students cant delete schools  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   deleteSchool := new(m.DeleteSchool) // Placeholder  deleteSchool.ID = c.Query("id") // Get school ID from query   code, resp := deleteSchool.Delete(db, user) // Call function  c.JSON(code, resp) }  // DeleteSchool deletes a school type DeleteSchool struct {  ID string `json:"id"` }  // Delete will delete a school func (d \*DeleteSchool) Delete(db \*gorm.DB, user \*User) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response  d.clean() // Remove trailing whitespace  school := new(School) // Placeholder   err := db.Where("id = ?", d.ID).First(school).Error // Get school by ID  if err != nil {  if util.IsNotFoundErr(err) { // if not found  resp.Data = nil  resp.Error = "Invalid school ID"  return 404, resp  }  return util.DatabaseError(err, resp) // If other error  }   if school.UserID != user.ID { // If the school doesn't belong to the headmaster  resp.Data = nil  resp.Error = "user does not own school"  return 403, resp  }   err = db.Delete(school).Error // Delete the record  if err != nil {  return util.DatabaseError(err, resp)  }   subjects := make([]\*Subject, 0) // Get all subjects  err = db.Where("school\_id = ?", school.ID).Find(&subjects).Error  if err != nil {  return util.DatabaseError(err, resp)  }   for \_, subject := range subjects { // Delete each subject  go subject.Delete(db)  }   resp.Data = true // Return success  resp.Error = ""   return 200, resp } |

There are also two GET requests associated with the endpoint. One is used to retrieve the information about the school

|  |
| --- |
| // GetSchool gets info about the school func GetSchool(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   schoolGetInfo := new(m.GetSchoolInfo)  schoolGetInfo.ID = c.Query("id") // Get variable from query   code, resp := schoolGetInfo.GetInfo(db, user)  c.JSON(code, resp) } // GetSchoolInfo is the model to get school info type GetSchoolInfo struct {  ID string `json:"id"` } // GetInfo gets the info for a school func (g \*GetSchoolInfo) GetInfo(db \*gorm.DB, user \*User) (int, \*util.Response) {   resp := new(util.Response) // Placeholder response   school := new(School)  err := db.Where("id = ?", g.ID).First(school).Error // Get school from database  if err != nil { // Handle errors  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Invalid school ID"  return 400, resp  }  return util.DatabaseError(err, resp)  }   // If the user isnt a headmaster and that the school doesnt belong to headmaster  // This check is needed because this endpoint works for all user types  if school.UserID != user.ID && user.Has(Headmaster) {  resp.Data = nil  resp.Error = "forbidden"  return 403, resp  }   school.Teachers = make([]\*User, 0) // Placeholders  school.Students = make([]\*User, 0)  school.Subjects = make([]\*Subject, 0)   db.Model(school).Association("Teachers").Find(&school.Teachers) // Fill out arrays with data  db.Model(school).Association("Students").Find(&school.Students)  db.Model(school).Association("Subjects").Find(&school.Subjects)   for i, subj := range school.Subjects { // For each subject find the assignments  db.Where("subject\_id = ?", subj.ID).Find(&school.Subjects[i].Assignments)  for j, assignment := range school.Subjects[i].Assignments {  db.Model(assignment).Association("Requests").Find(&school.Subjects[i].Assignments[j].Requests)  db.Model(assignment).Association("Files").Find(&school.Subjects[i].Assignments[j].Files)  db.Model(assignment).Association("CompletedBy").Find(&school.Subjects[i].Assignments[j].CompletedBy)  }  }   for i, subject := range school.Subjects { // For each subject get the teacher  usr := new(User)  err := db.Where("id = ?", subject.TeacherID).First(usr).Error  if err != nil {  return util.DatabaseError(err, resp)  }  school.Subjects[i].Teacher = usr  }   // If the user isnt a headmaster, need to verify the user  // is a teacher or a student at this school  if !user.Has(Headmaster) {  found := false  for \_, teacher := range school.Teachers {  if teacher.ID == user.ID {  found = true  break  }  }  for \_, student := range school.Students {  if found {  break  }  if student.ID == user.ID {  found = true  break  }  }   if !found {  resp.Data = nil  resp.Error = "forbidden"  return 403, resp  }  }   // If the user is a teacher  if user.Has(Teacher) {  school.Teachers = make([]\*User, 0) // remove info about  school.Students = make([]\*User, 0)   subj := make([]\*Subject, 0) // Get all the subjects  for \_, subject := range school.Subjects {  if subject.TeacherID == user.ID {  subj = append(subj, subject)  }  }   school.Subjects = subj  } else if user.Has(Student) { // If the user is a student  school.Teachers = make([]\*User, 0) // Remove all info other than the assignments  school.Students = make([]\*User, 0)  school.Subjects = make([]\*Subject, 0)  }   resp.Data = school // respond   return 200, resp } |

The other endpoint is used to retrieve all the students in the school, and is used by the teacher to add students to a subject they will be teaching

|  |
| --- |
| // GetStudents will get the students from a school func GetStudents(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if user.Has(m.Student) { // Students can't use this endpoint  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   getStudents := new(m.GetStudents)  getStudents.ID = c.Query("id")   code, res := getStudents.Get(db)  c.JSON(code, res) } // GetStudents is the model to get students from the school type GetStudents struct {  ID string } // Get gets the students from the school func (g \*GetStudents) Get(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response) // Placeholder   g.clean()   school := new(School)  err := db.Where("id = ?", g.ID).First(school).Error // Get school from database  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Invalid school ID"  return 404, resp  }  return util.DatabaseError(err, resp)  }   db.Model(school).Association("Students").Find(&school.Students) // Find all students  // If no students were found GORM sets the array to NULL,  // which will crash the JSON generator  if len(school.Students) == 0 {  resp.Data = []\*User{}  } else {  resp.Data = school.Students  }   resp.Error = ""   return 200, resp } |

**3.5 /api/teacher**

This endpoint is used to add and remove teachers from a school.

The endpoint definition is as follows

|  |
| --- |
| schGroup.POST("/teacher", m.ValidateJWT, api.AddTeacher) schGroup.DELETE("/teacher", m.ValidateJWT, api.DeleteTeacher) |

The POST request will add a teacher

|  |
| --- |
| // AddTeacher adds a teacher to the school func AddTeacher(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Headmaster) { // Only a headmaster can add teachers  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   newTeacher := new(m.NewTeacher)  err := json.NewDecoder(c.Request.Body).Decode(newTeacher)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, resp := newTeacher.Add(db)  c.JSON(code, resp) } // NewTeacher is the model to add a new teacher type NewTeacher struct {  FirstName string `json:"first\_name"`  LastName string `json:"last\_name"`  Email string `json:"email"`  SchoolID string `json:"school\_id"` }  // Add adds a new teacher to the database func (n \*NewTeacher) Add(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response) // Placeholder   n.clean()   if valid, reason := n.validate(); !valid { // Validate  resp.Data = nil  resp.Error = reason  return 400, resp  }   user := new(User)  err := db.Where("email = ?", n.Email).First(user).Error // Check if a user exists  found := !util.IsNotFoundErr(err)  if err != nil && found {  return util.DatabaseError(err, resp)  }   school := new(School)  err = db.Where("id = ?", n.SchoolID).First(school).Error // Get the school to which to add  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "School not found"  return 404, resp  }  return util.DatabaseError(err, resp)  }   if found { // If the user exists already   if user.Has(Teacher) {  resp.Data = nil  resp.Error = "User already a teacher"  return 409, resp  }   user.Perms |= Teacher // Add teacher to the users perms  school.Teachers = append(school.Teachers, user)   err = db.Save(user).Error // Save the user instance  if err != nil {  resp.Data = nil  resp.Error = "Internal error"  log.Printf("Database error: %s\n", err.Error())  return 500, resp  }   err = db.Save(school).Error // Save the school instance  if err != nil {  resp.Data = nil  resp.Error = "Internal error"  log.Printf("Database error: %s\n", err.Error())  return 500, resp  }  } else { // If the user isnt found (New user)  user = new(User)   user.ID = ksuid.New().String() // Fill out all info  user.Email = n.Email  user.FirstName = n.FirstName  user.LastName = n.LastName  user.Perms = Teacher  user.PassSet = false // There is no password  // User creates the password themselves on first login   err = db.Create(user).Error // Create the user  if err != nil {  resp.Data = nil  resp.Error = "Internal error"  log.Printf("Database error: %s\n", err.Error())  return 500, resp  }   school.Teachers = append(school.Teachers, user) // Add teacher to school   err = db.Save(school).Error // Save the school  if err != nil {  resp.Data = nil  resp.Error = "Internal error"  log.Printf("Database error: %s\n", err.Error())  return 500, resp  }  }   newTeacherResponse := struct { // Respond with teacher info  FirstName string `json:"first\_name"`  LastName string `json:"last\_name"`  ID string `json:"id"`  }{user.FirstName, user.LastName, user.ID}   resp.Data = newTeacherResponse  resp.Error = ""  return 201, resp } |

The DELETE request deletes the teacher

|  |
| --- |
| // DeleteTeacher deletes a teacher func DeleteTeacher(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Headmaster) {  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   deleteTeacher := new(m.DeleteTeacher)  deleteTeacher.UserID = c.Query("uid")  deleteTeacher.SchoolID = c.Query("sid")   code, resp := deleteTeacher.Delete(db)  c.JSON(code, resp) } // DeleteTeacher is a model to delete a teacher from a database type DeleteTeacher struct {  UserID string `json:"id"`  SchoolID string `json:"school\_id"` } // Delete deletes a teacher func (d \*DeleteTeacher) Delete(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response)   d.clean()   user := new(User)  err := db.Where("id = ?", d.UserID).First(user).Error // Get the user to delete  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Teacher not found"  return 404, resp  }  return util.DatabaseError(err, resp)  }   if !user.Has(Teacher) { // If the user isnt a teacher  resp.Data = nil  resp.Error = "Teacher not found"  return 404, resp  }   school := new(School) // Get the school to delete from  err = db.Where("id = ?", d.SchoolID).First(school).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Teacher not found"  return 404, resp  }  return util.DatabaseError(err, resp)  }   db.Model(school).Association("Teachers").Delete(user) // Delete the teacher from the association  user.Perms &^= Teacher // Remove the teacher flag   if user.Perms == 0 { // If no perms left then delete user  return user.Delete(db)  }   err = db.Save(user).Error // Save the user  if err != nil {  return util.DatabaseError(err, resp)  }   resp.Data = true  resp.Error = ""   return 202, resp } |

**3.6 /api/student**

This is a similar set of endpoints, with some properties changed

|  |
| --- |
| schGroup.POST("/student", m.ValidateJWT, api.AddStudent) schGroup.DELETE("/student", m.ValidateJWT, api.DeleteStudent) |

Add student:

|  |
| --- |
| // AddStudent adds a new student func AddStudent(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Headmaster) {  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   newStudent := new(m.NewStudent)  err := json.NewDecoder(c.Request.Body).Decode(newStudent)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, resp := newStudent.Add(db)  c.JSON(code, resp) } // NewStudent is a model to add a new student type NewStudent struct {  FirstName string `json:"first\_name"`  LastName string `json:"last\_name"`  Email string `json:"email"`  SchoolID string `json:"school\_id"` } // Add adds a new student to the database func (n \*NewStudent) Add(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response)   n.clean()   if valid, reason := n.validate(); !valid {  resp.Data = nil  resp.Error = reason  return 400, resp  }   user := new(User)  err := db.Where("email = ?", n.Email).First(user).Error  found := !util.IsNotFoundErr(err)  if err != nil && found {  return util.DatabaseError(err, resp)  }   school := new(School)  err = db.Where("id = ?", n.SchoolID).First(school).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "School not found"  return 404, resp  }  return util.DatabaseError(err, resp)  }   if found {   if user.Has(Student) {  resp.Data = nil  resp.Error = "User already a student"  return 409, resp  }   user.Perms |= Student  school.Students = append(school.Students, user)   err = db.Save(user).Error  if err != nil {  return util.DatabaseError(err, resp)  }   err = db.Save(school).Error  if err != nil {  return util.DatabaseError(err, resp)  }  } else {  user = new(User)   user.ID = ksuid.New().String()  user.Email = n.Email  user.FirstName = n.FirstName  user.LastName = n.LastName  user.Perms = Student  user.PassSet = false   err = db.Create(user).Error  if err != nil {  return util.DatabaseError(err, resp)  }   school.Students = append(school.Students, user)   err = db.Save(school).Error  if err != nil {  return util.DatabaseError(err, resp)  }  }   newStudentResponse := struct {  FirstName string `json:"first\_name"`  LastName string `json:"last\_name"`  ID string `json:"id"`  }{user.FirstName, user.LastName, user.ID}   resp.Data = newStudentResponse  resp.Error = ""  return 201, resp } |

Delete student

|  |
| --- |
| // DeleteStudent adds a new student func DeleteStudent(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Headmaster) {  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   deleteStudent := new(m.DeleteStudent)  deleteStudent.UserID = c.Query("uid")  deleteStudent.SchoolID = c.Query("sid")   code, resp := deleteStudent.Delete(db)  c.JSON(code, resp) }  // DeleteStudent is a model to delete a student from a database type DeleteStudent struct {  UserID string `json:"id"`  SchoolID string `json:"school\_id"` } // Delete deletes a teacher func (d \*DeleteStudent) Delete(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response)   d.clean()   user := new(User)  err := db.Where("id = ?", d.UserID).First(user).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Teacher not found"  return 404, resp  }  return util.DatabaseError(err, resp)  }   school := new(School)  err = db.Where("id = ?", d.SchoolID).First(school).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Teacher not found"  return 404, resp  }  return util.DatabaseError(err, resp)  }   db.Model(school).Association("Students").Delete(user)  user.Perms &^= Student   if user.Perms == 0 {  return user.Delete(db)  }   err = db.Save(user).Error  if err != nil {  return util.DatabaseError(err, resp)  }   resp.Data = true  resp.Error = ""   return 202, resp } |

**3.7 /api/logout**

This api endpoint logs a user out, by deleting the user’s token

|  |
| --- |
| logoutGroup := apiGroup.Group("/logout")  logoutGroup.POST("/", m.ValidateJWT, api.Logout) |

|  |
| --- |
| // Logout removes a token from a user func Logout(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   code, resp := user.Logout(db)  c.JSON(code, resp) }   // Logout removes a token from a user func (u \*User) Logout(db \*gorm.DB) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response   u.Token = "" // Set user token to empty  err := db.Save(u).Error // Save  if err != nil {  resp.Error = "internal error"  resp.Data = nil  log.Printf("Database error: %s\n", err.Error())  return 500, resp  }   resp.Data = true // Success  resp.Error = ""   return 202, resp } |

**3.8 /api/school/subject**

The definitions for the endpoint are as follows

|  |
| --- |
| subGroup := schGroup.Group("/subject") subGroup.POST("/", m.ValidateJWT, api.AddSubject) subGroup.DELETE("/", m.ValidateJWT, api.DeleteSubject) subGroup.GET("/", m.ValidateJWT, api.GetSubject) subGroup.POST("/students", m.ValidateJWT, api.AddStudentSubject) |

A teacher can add or remove subjects using the corresponding GET and POST endpoints, and the teacher will be responsible for the subjects that were added or deleted.

Below is the code to add a new subject

|  |
| --- |
| // AddSubject adds a new subject func AddSubject(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Database variable from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // Get the user from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Teacher) { // Only the teacher can add subjects  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   newSubject := new(m.NewSubject) // Create a model  err := json.NewDecoder(c.Request.Body).Decode(newSubject)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, resp := newSubject.Add(db, user) // Run function  c.JSON(code, resp) }  // NewSubject is a new subject type NewSubject struct {  Name string `json:"name"`  SchoolID string `json:"school\_id"` }  // Add adds a subject to the database func (n \*NewSubject) Add(db \*gorm.DB, u \*User) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response   n.clean() // Remove trailing whitespace   if valid, reason := n.validate(); !valid { // Valiate the input  resp.Data = nil  resp.Error = reason  return 400, resp  }   school := new(School) // Get the school from database by ID provided  err := db.Where("id = ?", n.SchoolID).First(school).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Invalid school id"  return 400, resp  }  return util.DatabaseError(err, resp)  }   db.Model(school).Association("Teachers").Find(&school.Teachers) // Get all teachers from the school   found := false // Check if the teacher is in the school  for \_, t := range school.Teachers {  if u.ID == t.ID {  found = true  break  }  }   if !found { // If not in school  resp.Data = nil  resp.Error = "Teacher not in school"  return 403, resp  }   subj := new(Subject) // Create new subject  subj.ID = ksuid.New().String()  subj.TeacherID = u.ID  subj.Teacher = u  subj.Name = n.Name  subj.NumStudents = 0  subj.SchoolID = n.SchoolID   err = db.Create(subj).Error // Save it to the database  if err != nil {  return util.DatabaseError(err, resp)  }   school.Subjects = append(school.Subjects, subj) // Add the subject to the school  err = db.Save(school).Error // Save the school  if err != nil {  return util.DatabaseError(err, resp)  }   subjResponse := struct { // Create a response  ID string `json:"id"`  Name string `json:"name"`  }{subj.ID, subj.Name}   resp.Data = subjResponse // Respond to the user  resp.Error = ""  return 201, resp } |

The delete procedure is as follows:

|  |
| --- |
| // DeleteSubject deletes a subject func DeleteSubject(c \*gin.Context) { // Database variable from context  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // User variable from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Teacher) { // Only the teacher can add subjects  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   deleteSubject := new(m.DeleteSubject) // Create model  deleteSubject.ID = c.Query("id")   code, resp := deleteSubject.Delete(db, user) // Call function  c.JSON(code, resp) }  // DeleteSubject deletes a subject type DeleteSubject struct {  ID string `json:"id"` }  // Delete deletes a subject func (d \*DeleteSubject) Delete(db \*gorm.DB, user \*User) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response   d.clean()   subject := new(Subject) // Get subject from ID  err := db.Where("id = ?", d.ID).First(subject).Error  if err != nil {  if util.IsDuplicateErr(err) {  resp.Data = nil  resp.Error = "Invalid subject id"  return 400, resp  }  return util.DatabaseError(err, resp)  }   subject.Teacher = new(User) // Get the subject teacher  db.Model(subject).Association("Teacher").Find(&subject.Teacher)   if subject.Teacher.ID != user.ID { // If the teacher doesnt own the subject  resp.Data = nil  resp.Error = "forbidden"  return 403, resp  }   // Headmaster can also delete the subject, so get the headmaster from the school  school := new(School)  err = db.Where("id = ?", subject.SchoolID).First(school).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Check if the user is a headmaster but doesn't own the school  if school.UserID != user.ID && user.Has(Headmaster) {  resp.Data = nil  resp.Error = "forbidden"  return 403, resp  }   db.Delete(subject) // Delete the subject from the database   resp.Data = true  resp.Error = ""  return 202, resp } |

When the teacher is trying to get the subject, this code will be run

|  |
| --- |
| // GetSubject gets information about a subject func GetSubject(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User)  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   getSubject := new(m.GetSubjectInfo)  getSubject.ID = c.Query("id") // Get the subject ID from query  getSubject.SID = c.Query("sid") // Get the student ID from query  code, resp := getSubject.Get(db, user)  c.JSON(code, resp) }  // GetSubjectInfo is the internal model to retreive a subject's info type GetSubjectInfo struct {  ID string }   // Get gets info about the subject func (g \*GetSubjectInfo) Get(db \*gorm.DB, user \*User) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response   subject := new(Subject) // Get the subject from the database  err := db.Where("id = ?", g.ID).First(subject).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Unknown subject ID"  return 404, resp  }  return util.DatabaseError(err, resp)  }   if user.Has(Teacher) { // If the user is a teacher  if subject.TeacherID != user.ID { // If the teacher doesnt own the subjet  resp.Data = nil  resp.Error = "forbidden"  return 403, resp  }   db.Model(subject).Association("Students").Find(&subject.Students) // Get all the students for the subject   // Get ten latest assignments for the subject  db.Where("subject\_id = ?", g.ID).Order("time\_assigned desc").Limit(10).Find(&subject.Assignments)  for a, assignment := range subject.Assignments {  // For each assignment get the requests  db.Model(assignment).Association("Requests").Find(&assignment.Requests)  // For each request  for \_, req := range assignment.Requests {  // Get all student uploads  upl := make([]\*RequestUpload, 0)  req.Uploads = upl  db.Model(req).Association("Uploads").Find(&req.Uploads)  req.Complete = nil  }    // Get the list of students that completed it  db.Model(assignment).Association("CompletedBy").Find(&assignment.CompletedBy)   // If the assignment has a due time  if assignment.TimeDue != nil {  // Get list of completed users  completed := make(map[string]bool)  // For each user that completed the assignment make a record of that  for \_, compl := range assignment.CompletedBy {  completed[compl.ID] = true  }   // make an array of users that didn't complete it  assignment.NotCompletedBy = make([]\*User, 0)  // Fill out the list of students that did not complete it  for \_, student := range subject.Students {  if \_, ok := completed[student.ID]; !ok {  assignment.NotCompletedBy = append(assignment.NotCompletedBy, student)  }  }  }   subject.Assignments[a] = assignment  }   // Get all students from the subject  getStudents := new(GetStudents)  getStudents.ID = g.SID  code, students := getStudents.Get(db)   // If an error occured while getting the user,  // respond with its error code instead  if code != 200 {  return code, students  }   // Respond  response := struct {  S \*Subject `json:"subject"`  St []\*User `json:"students"`  }{  subject,  students.Data.([]\*User),  }   resp.Data = response  resp.Error = ""   } else { // If the user isn't a teacher then forbid access  resp.Data = nil  resp.Error = "forbidden"  return 403, resp  }   return 200, resp } |

To add students these functions are called

|  |
| --- |
| // AddStudentSubject adds a new student to a subject func AddStudentSubject(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Get database from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // Get user from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Teacher) && !user.Has(m.Headmaster) { // Students cant add others to a subject  c.JSON(403, gin.H{"error": "fobidden"})  return  }   newStudentSubject := new(m.NewSubjectStudent) // Create model  err := json.NewDecoder(c.Request.Body).Decode(newStudentSubject)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, resp := newStudentSubject.Add(db, user) // Add the student  c.JSON(code, resp) }  // NewSubjectStudent is the model to add a new student to the subject type NewSubjectStudent struct {  ID string `json:"user\_id"`  Subject string `json:"subject\_id"` }  // Add adds a new student to a subject func (n \*NewSubjectStudent) Add(db \*gorm.DB, user \*User) (int, \*util.Response) {  resp := new(util.Response) // Response placeholder  n.clean() // Remove trailing whitespace   usr := new(User) // Get student that needs to be added  err := db.Where("id = ?", n.ID).First(usr).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Invalid user id"  return 400, resp  }  return util.DatabaseError(err, resp)  }   subject := new(Subject) // Get the subject to which to add the student  err = db.Where("id = ?", n.Subject).First(subject).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Invalid subject id"  return 400, resp  }  return util.DatabaseError(err, resp)  }    // If the teacher doesn't own the subject then restrict access  if subject.TeacherID != user.ID {  resp.Data = nil  resp.Error = "forbidden"  return 403, resp  }   school := new(School) // Get the school in which the subject is  err = db.Where("id = ?", subject.SchoolID).First(school).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Get a list of all students in the school  err = db.Model(school).Association("Students").Find(&school.Students).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Get a list of all students in a subject  err = db.Model(subject).Association("Students").Find(&subject.Students).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Get all subjects the user is taking  err = db.Model(user).Association("Subjects").Find(&user.Subjects).Error  if err != nil {  return util.DatabaseError(err, resp)  }   if !usr.Has(Student) {  resp.Data = nil  resp.Error = "user not a student"  return 400, resp  }   // Check if the student is in the school  found := false  for \_, student := range school.Students {  if student.ID == usr.ID {  found = true  break  }  }  if !found {  resp.Data = nil  resp.Error = "user not in school"  return 400, resp  }   // Check if the student is already added to the subject  found = false  for \_, student := range subject.Students {  if student.ID == usr.ID {  found = true  break  }  }  if found {  resp.Data = nil  resp.Error = "Student already added"  return 409, resp  }   // Add the student to the subjects list  subject.Students = append(subject.Students, usr)  subject.NumStudents++  err = db.Save(subject).Error // Save the subject  if err != nil {  return util.DatabaseError(err, resp)  }   // Add the subject to the user  usr.Subjects = append(usr.Subjects, subject)  err = db.Save(user).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Create a response  studentResponse := struct {  ID string `json:"id"`  FirstName string `json:"first\_name"`  LastName string `json:"last\_name"`  }{usr.ID, usr.FirstName, usr.LastName}   resp.Data = studentResponse  resp.Error = ""   return 201, resp } |

**3.6 /api/school/subject/assignment**

When teachers want to assign homework to students, they create an assignment. Assignments contain files and requests. Students can download the files, and then upload them into the requests, which the teacher can then view.

The teacher can also set a due date, and the student is notified when a request is overdue.

The requests are accessed via the /api/school/subject/assignment endpoint, which is defined like this:

|  |
| --- |
| assgnGroup := subGroup.Group("/assignment") assgnGroup.POST("/", m.ValidateJWT, api.NewAssignment) assgnGroup.GET("/", m.ValidateJWT, api.GetAssignment) assgnGroup.POST("/complete", m.ValidateJWT, api.CompleteAssignment) |

The assignment struct looks like this:

|  |
| --- |
| // Assignment is the internal model for assignments type Assignment struct {  ID string `gorm:"primaryKey" json:"id"`  TeacherID string `gorm:"not null" json:"teacher\_id"`  Teacher \*User `gorm:"not null" json:"teacher,omitempty"`  SubjectID string `gorm:"not null" json:"subject\_id,omitempty"`  Name string `gorm:"not null" json:"name"`  Text string `json:"text"`  Files []\*AssignmentFile `json:"files,omitempty"`  TimeDue \*time.Time `json:"time\_due,omitempty"`  TimeAssigned \*time.Time `json:"time\_assigned"`  Requests []\*Request `json:"requests,omitempty"`  CompletedBy []\*User `gorm:"many2many:assignments\_completed" json:"comleted\_by,omitempty"`  NotCompletedBy []\*User `gorm:"-" json:"not\_completed\_by,omitempty"` } |

There are some additional models that are used to track uploads and upload requests

|  |
| --- |
| // AssignmentFile is the internal structure of a file relating to an assignment type AssignmentFile struct {  AssignmentID string `json:"-"`  Path string `gorm:"primaryKey" json:"path"`  Name string `json:"name"` }  // Request is the model to request an upload for the students type Request struct {  ID string `gorm:"primaryKey" json:"id"`  AssignmentID string `json:"-"`  Name string `gorm:"not null" json:"name"`  Complete \*bool `gorm:"-" json:"complete,omitempty"`  Uploads []\*RequestUpload `json:"uploads,omitempty"` }  // RequestUpload is the model to tracks who uploaded what type RequestUpload struct {  RequestID string `json:"-"`  UserID string `json:"-"`  User \*User `gorm:"-" json:"user"`  Filepath string `json:"path"`  Filename string `json:"name"` } |

When a teacher is adding an assignment, these functions are called

|  |
| --- |
| // NewAssignment creates a new assignment func NewAssignment(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Get database from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // Get user from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Teacher) { // Only teachers can create assignments  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   newAssignment := new(m.NewAssignment) // Create a model  err := json.NewDecoder(c.Request.Body).Decode(newAssignment)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, resp := newAssignment.Create(db, user) // Call model function  c.JSON(code, resp) }  // NewAssignment is a model to create a new assignment type NewAssignment struct {  Name string `json:"name"`  Text string `json:"text"`  SubjectID string `json:"subject\_id"`  // TimeDue field is a pointer because it is optional  // if the pointer is nil, we can ignore it and say  // there is no time due for the request  TimeDue \*time.Time `json:"time\_due"`  Files []string `json:"files"`  UploadRequest bool `json:"-"`  UploadRequests []string `json:"uploads"` }  // Create creates a new assignment func (n \*NewAssignment) Create(db \*gorm.DB, user \*User) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response  n.clean()   if valid, reason := n.validate(); !valid {  resp.Data = nil  resp.Error = reason  return 400, resp  }   // Check if the time due is in the past, and return an error   if n.UploadRequest && (n.TimeDue == nil || n.TimeDue.Before(time.Now())) {  resp.Data = nil  resp.Error = "Cannot set time due in the past"  return 400, resp  }   // Get the subject by ID to which the assignment is added  subject := new(Subject)  err := db.Where("id = ?", n.SubjectID).Find(subject).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "Invalid subject ID"  return 403, resp  }  return util.DatabaseError(err, resp)  }   // Check if the user owns the subject  if subject.TeacherID != user.ID {  resp.Data = nil  resp.Error = "forbidden"  return 403, resp  }   // Generate filenames for the assignments   // Create an empty array  names := make([]string, len(n.Files))  for i, file := range n.Files {  // Get the file on disk  file = util.ToGlobalPath(file)  // If it doesnt exist then throw an error  if \_, err := os.Stat(file); os.IsNotExist(err) {  resp.Data = nil  resp.Error = "Internal error"  return 500, resp  }   // Split the filename and extension  name, ext := util.SplitName(file)  // Set file as verified  name = name[:len(name)-2] + "\_1"  // Rename the file to validate it  os.Rename(file, name+ext)  // Place the name in the array  names[i] = name + ext  }   now := time.Now()   // Set all the fields in the assignment  assignment := new(Assignment)  assignment.ID = ksuid.New().String()  assignment.Name = n.Name  assignment.Text = n.Text  assignment.TeacherID = user.ID  assignment.SubjectID = subject.ID  assignment.TimeDue = n.TimeDue  assignment.TimeAssigned = &now   // Save the assignment in the database  err = db.Save(assignment).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Create all the necessary upload requests  requests := make([]\*Request, len(n.UploadRequests))   // Generate all the necessary info for all the requests  for i, uploadReq := range n.UploadRequests {  request := new(Request)  request.ID = ksuid.New().String()  request.AssignmentID = assignment.ID  request.Name = uploadReq   requests[i] = request  db.Save(request)  }   // Create all links to the files that are attached to the assignment  files := make([]\*AssignmentFile, len(names))  for i, name := range names {  file := new(AssignmentFile)  file.AssignmentID = assignment.ID  file.Path = name  file.Name = util.ToLocalPath(name)   files[i] = file  db.Save(file)  }   // Make a response  assgn := struct {  ID string `json:"id"`  Name string `json:"name"`  TeacherID string `json:"teacher\_id"`  SubjectID string `json:"subject\_id"`  AssignedAt time.Time `json:"time\_assigned"`  Files []\*AssignmentFile `json:"files"`  Uploads []\*Request `json:"uploads"`  }{assignment.ID, assignment.Name, assignment.TeacherID, assignment.SubjectID, now, files, requests}   resp.Data = assgn  resp.Error = ""   return 200, resp } |

When a student is trying to get information about the assignment, these functions are run

|  |
| --- |
| // GetAssignment gets information about the assignment func GetAssignment(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Get database from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // Get user from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   gasn := new(m.GetAssignment) // Create models and run functions  gasn.ID = c.Query("id")  code, resp := gasn.Get(db, user)  c.JSON(code, resp) }  // GetAssignment is the model to get the assignment type GetAssignment struct {  ID string }  // Get gets the assignment information func (g \*GetAssignment) Get(db \*gorm.DB, user \*User) (int, \*util.Response) {  resp := new(util.Response) // Response placeholder   assignment := new(Assignment) // Get the assignment by ID  err := db.Where("id = ?", g.ID).First(assignment).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "not found"  return 400, resp  }  return util.DatabaseError(err, resp)  }  db.Model(assignment).Association("Files").Find(&assignment.Files) // Get all the files for the assignment   // Get the subject the assignment is in  subject := new(Subject)  err = db.Where("id = ?", assignment.SubjectID).First(subject).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "not found"  return 400, resp  }  return util.DatabaseError(err, resp)  }  // Get all students from the subject  db.Model(subject).Association("Students").Find(&subject.Students)   // If the user is a teacher then  if user.Has(Teacher) {  // Get all the requests, for each request get the requests,  // and for each request get all the files that students uploaded  db.Model(assignment).Association("Requests").Find(&assignment.Requests)  for \_, req := range assignment.Requests {  upl := make([]\*RequestUpload, 0)  req.Uploads = upl  uplBuf := make([]\*RequestUpload, 0)  db.Model(req).Association("Uploads").Find(&req.Uploads)  req.Complete = nil   for \_, uploads := range req.Uploads {  u := new(User)  db.Where("id = ?", uploads.UserID).First(u)  uploads.User = u  uplBuf = append(uplBuf, uploads)  }   req.Uploads = uplBuf  }  // Get who has completed each assignment  db.Model(assignment).Association("CompletedBy").Find(&assignment.CompletedBy)   // If the assignment has a time due  if assignment.TimeDue != nil {  // Check who completed the assignment and who didnt  completed := make(map[string]bool)  for \_, compl := range assignment.CompletedBy {  completed[compl.ID] = true  }   assignment.NotCompletedBy = make([]\*User, 0)  for \_, student := range subject.Students {  if \_, ok := completed[student.ID]; !ok {  assignment.NotCompletedBy = append(assignment.NotCompletedBy, student)  }  }  }  } else {  // Get all the requests for a student  db.Model(assignment).Association("Requests").Find(&assignment.Requests)  // For each request get what needs to be uploaded  for r, req := range assignment.Requests {  db.Model(req).Association("Uploads").Find(&assignment.Requests[r].Uploads)  found := false  for \_, upload := range assignment.Requests[r].Uploads {  if upload.UserID == user.ID {  found = true  break  }  }  req.Complete = &found  req.Uploads = nil  }  }   resp.Data = assignment  resp.Error = ""   return 200, resp } |

And finally, students who want to complete their assignment by uploading a file use the POST method

|  |
| --- |
| // CompleteAssignment completes an assignment func CompleteAssignment(c \*gin.Context) {  db, ok := c.Keys["db"].(\*gorm.DB) // Database variable from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no database variable in context")  }   user, ok := c.Keys["usr"].(\*m.User) // Get user from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no user variable in context")  }   if !user.Has(m.Student) { // Only students can complete their assignments  c.JSON(403, gin.H{"error": "insufficient permissions"})  return  }   newComplete := new(m.NewRequestComplete) // Create a model and run the method  err := json.NewDecoder(c.Request.Body).Decode(newComplete)  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   code, resp := newComplete.Complete(db, user)  c.JSON(code, resp) } // NewRequestComplete is a model to complete an upload request type NewRequestComplete struct {  RequestID string `json:"request\_id"`  Filename string `json:"filepath"` }  // Complete completes the upload request func (n \*NewRequestComplete) Complete(db \*gorm.DB, user \*User) (int, \*util.Response) {  resp := new(util.Response) // Placeholder response   n.clean() // Remove trailing whitespace   request := new(Request) // Get the request to complete from the database  err := db.Where("id = ?", n.RequestID).First(request).Error  if err != nil {  if util.IsNotFoundErr(err) {  resp.Data = nil  resp.Error = "request not found"  return 400, resp  }  return util.DatabaseError(err, resp)  }   assgn := new(Assignment) // Get the assignment from which the request is  err = db.Where("id = ?", request.AssignmentID).First(assgn).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Fill in all the requests  db.Model(assgn).Association("Requests").Find(&assgn.Requests)   subj := new(Subject) // Get the subject from which the assignment is  err = db.Where("id = ?", assgn.SubjectID).First(subj).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Get all the students from the subject  db.Model(subj).Association("Students").Find(&subj.Students)  // Check if the student takes the subject  found := false  for \_, student := range subj.Students {  if student.ID == user.ID {  found = true  break  }  }  if !found {  resp.Data = nil  resp.Error = "user not in subject"  return 403, resp  }   // Get all the uplopads from the request to check if a file was already uploaded  db.Model(request).Association("Uploads").Find(&request.Uploads)  for \_, upl := range request.Uploads {  if upl.UserID == user.ID {  resp.Data = nil  resp.Error = "file already uploaded"  return 409, resp  }  }   // Check if the file that the user uploaded exists  if \_, err := os.Stat(util.ToGlobalPath(n.Filename)); os.IsNotExist(err) {  resp.Data = nil  resp.Error = "file doesnt exist"  return 400, resp  }   // If all checks passed  // Create a request upload  reqUpl := new(RequestUpload)  reqUpl.Filepath = util.ToGlobalPath(n.Filename)  reqUpl.Filename = n.Filename  reqUpl.RequestID = request.ID  reqUpl.UserID = user.ID   // Save into database  err = db.Create(reqUpl).Error  if err != nil {  return util.DatabaseError(err, resp)  }   // Check if the student has any remaining requests unfilled  completed := true  db.Model(assgn).Association("Requests").Find(&assgn.Requests)  db.Model(assgn).Association("CompletedBy").Find(&assgn.CompletedBy)  for \_, req := range assgn.Requests {  db.Model(req).Association("Uploads").Find(&req.Uploads)  found := false  for \_, upload := range req.Uploads {  if upload.UserID == user.ID {  found = true  break  }  }  if !found {  completed = false  break  }  }   // if all the requests were completed  if completed {  // Here I use a raw database query because  // I need to create an association with existing data  // which GORM doesnt yet support  err = db.Exec("insert into assignments\_completed(assignment\_id, user\_id) values (?, ?);", assgn.ID, user.ID).Error  if err != nil {  return util.DatabaseError(err, resp)  }  }   resp.Data = true  resp.Error = ""   return 202, resp } |

**3.7 /api/files**

Students and teachers can upload files to the server. The actual mechanism of how they work will be discussed later

The definition of the endpoint where files are uploaded is as follows

|  |
| --- |
| g.Static("/files", "./files") fsgroup := g.Group("/files") fsgroup.POST("/", api.CreateFile) |

The g.Static line allows any connection to access any file in a given directory

The first parameter shows to which endpoint the GET request should be sent, and the second indicates the directory on the hard drive itself, where the file is located. For example, a request to /api/files/test\_file.pdf searches the ./files directory, and returns the file if it is found, and a 404 error if it is missing

The function that manages and creates files that users upload

|  |
| --- |
| // CreateFile creates a new file for upload func CreateFile(c \*gin.Context) {   gcChan, ok := c.Keys["collector"].(\*chan []string) // Get the garbage file collector from context  if !ok {  c.JSON(500, gin.H{"error": "internal error"})  panic("no channel variable in context")  }   // The user sends the file as a multipart form with the files field  form, err := c.MultipartForm() // Get the data that user sends  if err != nil {  c.JSON(400, gin.H{"error": "invalid data"})  return  }   // Get all files from the form  files := form.File["files"]  names := make([]string, len(files))   // If no files were sent, then dont do anything  if len(files) == 0 {  return  }   // For each file check the size, if its more than 100mb then exit  for \_, file := range files {  if file.Size > 100000000 {  c.JSON(400, gin.H{"error": fmt.Sprintf("File %s is too large, limit is 100MB", file.Filename)})  return  }  }   // For each file  for i, file := range files {  // Remove extension  \_, ext := util.SplitName(file.Filename)  // Generate a random name for a file  name := util.GenerateName()   // Add \_0 to the end and the extension, to mark that the file isnt  // verified  name = name + "\_0" + ext  fname := name   // Get the global path of the file  name = util.ToGlobalPath(name)   // Save the uploaded file into the disk  err = c.SaveUploadedFile(file, name)  if err != nil {  c.JSON(500, gin.H{"error": "internal error"})  log.Printf("File upload error %s\n", err.Error())  return  }   names[i] = fname  }  // Send the files to the garbage collector  \*gcChan <- names   // respond to the user with the uploaded files and ways to access them  resp := struct {  Files []string `json:"files"`  }{names}   c.JSON(200, resp) } |

This is all the functions that are run by the API endpoints. Now I will describe the utility functions, file management and finally the web frontend.

**4. File storage and garbage file collection**

Files are stored in the server in the files directory, and are accessible from any client with a GET request to the /files endpoint.

To store the files a system is used to verify that a file is actually used. Since anyone can upload a file to the server, any unused files have to be deleted after 10 minutes. When a file is uploaded, it is sent to the garbage collector via a channel, which then checks for 10 minutes if the file is still unverified, and if in that time the file was verified, it stops running and the file remains on the server permanently.

An unverified file is marked with \_0 after its name, and with \_1 after it has been verified. When a file is uploaded, a random 30-character name is generated to make it unique.

An example filename stored on the server will be zi4shrwuBhUIrD2r93CmfyE77cRXK3\_0.jpg, which means it is still unverified.

The garbage collector creates separate goroutines for each file upload, so performance is not affected, and a parent goroutine manages all the existing goroutines, and stops them when they have detected a file has been verified.

This is the code of the garbage collector

|  |
| --- |
| // The channel which is used to transfer files to the garbage collector var fileChannel chan []string  // Quit is the channel to quit the execution of the garbage collector var Quit chan bool  // Init is run before the main() function func init() {  fileChannel = make(chan []string) }  // GetChannel gets the pointer to the garbage collector file channel func GetChannel() \*chan []string {  return &fileChannel }  // AddCollectorToContext adds the garbage collector channel to the context func AddCollectorToContext(c \*gin.Context) {  c.Set("collector", GetChannel())  c.Next() }  // Run runs the garbage collector func Run() {  log.Println("Started garbage collector") // Write out a log  for { // Check which channel has incoming information  select {  case f := <-fileChannel: // If there is an incoming file  go func(files []string) { // Run a function on a new goroutine  defer log.Printf("Cleaned %v", files) // When it returns, write out the log with all the cleaned files  for i := 0; i < 10; i++ { // Check each minute if the files are still unverified  time.Sleep(1 \* time.Minute)  for \_, file := range files { // For each file check if it exists  if \_, err := os.Stat(util.ToGlobalPath(file)); os.IsNotExist(err) {  return // If a file doesnt exist that means it was verified and the goroutine can be stopped  }  }  }  // After ten minutes delete all the files  for \_, file := range files {  // Check if the file exists  file = util.ToGlobalPath(file)  if \_, err := os.Stat(file); os.IsNotExist(err) {  continue  }  // Remove the file  err := os.Remove(file)  if err != nil {  panic(err) // If an unexpected error happened then panic  }  }  return  }(f)  // When a quit signal is received then stop the garbage collector  case <-Quit:  log.Printf("Stopping garbage collection")  return  }  } } |

**5. Utility functions**

Some small functions in the util package are used a lot throughout many different packages

These are the error functions

|  |
| --- |
| // IsDuplicateErr checks if the error generated comes from inserting a duplicate record func IsDuplicateErr(err error) bool {  pqerr, ok := err.(\*pq.Error) // Get the PostgreSQL error code  return ok && pqerr.Code.Name() == "unique\_violation" // Return true if the  }  // IsNotFoundErr checks if the returned error signifies a missing record func IsNotFoundErr(err error) bool {  return err != nil && errors.Is(err, gorm.ErrRecordNotFound) }  // Response is the response struct, that will be sent back to the user type Response struct {  Error string `json:"error,omitempty"`  Data interface{} `json:"data,omitempty"` }  // DatabaseError notifies of a database error func DatabaseError(err error, resp \*Response) (int, \*Response) {  resp.Data = nil // Set the response data to null which will make it missing from the response JSON  resp.Error = "Internal error" // Set the error  log.Printf("Database error: %s\n", err.Error())  return 500, resp // Return a 500 error } |

The are utility functions for the files

|  |
| --- |
| // SplitName splits the filename info extension and name func SplitName(filename string) (string, string) {  ext := filepath.Ext(filename) // Get the extension of the file  name := strings.TrimSuffix(filename, ext) // Remove the extension from the original name  return name, ext // Return both name and extension }  // GenerateName generates a random name for a file func GenerateName() string {  r := make([]byte, 30) // Make a 30 lette array  // The list of the letters to pick from  c := "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"  // For each letter pick a random one from the list  for i := range r {  r[i] = c[rand.Intn(len(c))]  }  // Return as a string  return string(r) }  // ToGlobalPath converts filename to file path in file server func ToGlobalPath(p string) string {  return "files/" + p }  // ToLocalPath converts filename to file path in file server func ToLocalPath(p string) string {  return p[6:] } |

These are all the utility functions to work with numbers

|  |
| --- |
| // Hash will hash a string and return it func Hash(hash string) string {  hashed, \_ := bcrypt.GenerateFromPassword([]byte(hash), 10)  return string(hashed) }  // Compare compares a hashed password to a plaintext password func Compare(hash string, plaintext string) bool {  err := bcrypt.CompareHashAndPassword([]byte(hash), []byte(plaintext))  return err == nil }  // RandomCode generates a random string func RandomCode() string {  r := make([]byte, 15) // Make a 15 letter random code  c := "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789$%{}#!@\_"  // Fill in an array with random letters  for i := range r {  r[i] = c[rand.Intn(len(c))]  }  // Return the hash of that for more randomness  return Hash(string(r)) } |

These all functions are used to validate and clean strings

|  |
| --- |
| // ValidateName tells if a name is valid or not func ValidateName(name string) bool {  return len(name) > 3 }  // ValidateEmail checks if the email is valid func ValidateEmail(email string) bool {  // A really long regex to check if the email is right  // Source: https://emailregex.com/  reg := regexp.MustCompile(`(?:[a-z0-9!#$%&'\*+/=?^\_\x60{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_\x60{|}~-]+)\*|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])\*")@(?:(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]\*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])`)  return reg.Match([]byte(email)) }  // Clean removes duplicate whitespace and illegal characters func Clean(text \*string) {  \*text = strings.TrimSpace(\*text)  \*text = html.EscapeString(\*text)  reg := regexp.MustCompile(`([^a-zA-Z0-9\s])|(\s{2,})`)  \*text = reg.ReplaceAllString(\*text, " ") }  // RemoveSpaces removes n duplicates and leading/trailing spaces from a string func RemoveSpaces(password \*string) {  \*password = html.EscapeString(\*password)  \*password = strings.TrimSpace(\*password)  reg := regexp.MustCompile(`\s`)  \*password = reg.ReplaceAllString(\*password, "") } |

**6. Frontend**

Since classwork is a web app, users interact with it using a webpage, and connecting to a server. The page was written in HTML, with the Bootstrap 5 CSS library, JQuery, Axios JS for network requests and FontAwesome for some icons.

**6.1 Registration page**

The registration page contains some fields for the user to register a new account. The only account type that can be created is a Headmaster account. The headmaster account can later create Teacher and Student accounts for each school.

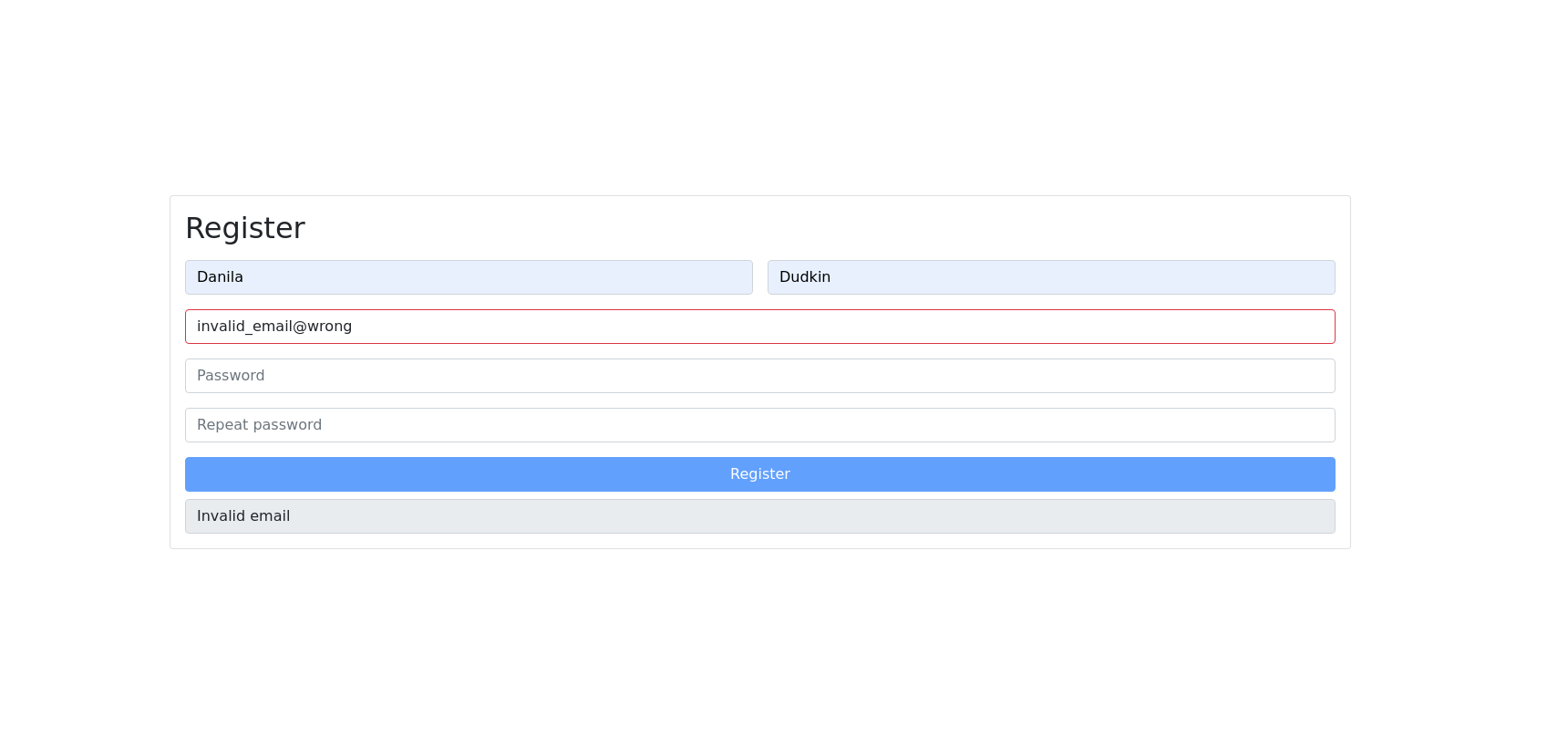
This is the HTML code for the registration page.

|  |
| --- |
| <!DOCTYPE html> <html lang="en">  <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1" />   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.slim.min.js"></script>  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>  <script src="/static/js/register.js"></script>   <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"  rel="stylesheet"  integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"  crossorigin="anonymous"  />  <link rel="stylesheet" href="/static/css/main.css" />   <title>Register</title>  </head>  <body>  <div class="container-md mx-auto vertical-center">  <div class="col">  <form class="card">  <div class="card-body">  <h2 class="mb-3">Register</h2>  <div class="row g-3 mb-3">  <div class="col">  <input  id="fname"  type="text"  class="form-control"  placeholder="First name"  aria-label="First name"  onblur="validateName(this)"  />  </div>  <div class="col">  <input  id="lname"  type="text"  class="form-control"  placeholder="Last name"  aria-label="Last name"  onblur="validateName(this)"  />  </div>  </div>  <div class="mb-3">  <input  type="email"  class="form-control"  id="email-input"  placeholder="Email"  onblur="validateEmail(this)"  />  </div>  <div class="mb-3">  <input  type="password"  class="form-control"  id="pass-input"  placeholder="Password"  onblur="validatePassword()"  />  </div>  <div class="mb-3">  <input  type="password"  class="form-control"  id="pass-repeat"  placeholder="Repeat password"  onblur="validatePassword()"  />  </div>  <div class="d-grid gap-2">  <button  onclick="register()"  type="button"  id="reg-button"  class="btn btn-primary disabled"  >  Register  </button>  <input  id="errors"  class="form-control"  type="text"  readonly  />  </div>  </div>  </form>  </div>  </div>  <script  src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle.min.js"  integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroafW"  crossorigin="anonymous"  ></script>  </body> </html> |

And the JavaScript code for the web page

|  |
| --- |
| function validateName(element) {  el = $(element) // Perform the same check as the server  var name = el.val() // but in real time  setValid(name.length > 3, "Name must be at least 4 characters long", el) }  function validateEmail(element) {  el = $(element) // Get the value of the input field   var params = { // Prepare the data to be sent to the server  email: el.val()  }   axios // Use axios to form a GET request  .get("/api/register/email", {  params: params // Set the parameters of the request to the above value  })  .then((res) => { // After a 2xx response was received  var valid = res.data.data  setValid(valid, "Email is taken", el)  })  .catch((err) => { // If a non-2xx response was received  var error = err.response.data.error  setValid(false, error, el)  }) }  function setValid(valid, text, ...elements) { // Enable or disable the button  if(!valid) { // Code to disable the button  elements.forEach((el) => { // Mark all the elements with an error  el.addClass("border-danger")  })  $("#errors").val(text)  $("#reg-button").addClass("disabled")  } else { // Code to enable the button  elements.forEach((el) => { // Unmark all the elements with an error  el.removeClass("border-danger")  })  $("#reg-button").removeClass("disabled")  $("#errors").val("")  }  $("#waiter").remove() }  function validatePassword() { // Check if the two passwords match  var pass = $("#pass-input")  var rpass = $("#pass-repeat")  setValid(pass.val() === rpass.val(), "Passwords do not match", pass, rpass) }  function register() {  // Create a spinner inside the button element and disable it while waiting  $("#reg-button").addClass("disabled")  $("#reg-button").prepend($("<span id=\"waiter\" class=\"spinner-grow spinner-grow-sm\" role=\"status\" aria-hidden=\"true\"></span>"))   // The data being sent to the server  var data = {  first\_name: $("#fname").val(),  last\_name: $("#lname").val(),  email: $("#email-input").val(),  password: $("#pass-input").val(),  }   // Make a request  axios  .post("/api/register", data) // Set above data as the data for the post request  .then((res) => { // If no error then redirect to the login page  window.location.replace("/login")  })  .catch((err) => { // If an error occurred then set all error fields  setValid(false, err.response.data.error)  }) } |

In the end, the registration page with an example of an invalid email is shown below



**6.2 The login and password page**

As discussed below, the login process occurs in two stages, one in which the user enters their email, and then a password, where a decision is made to create a new password for the user, or use an existing one, based on whether a flag is set in the database.

This is the html for the login page

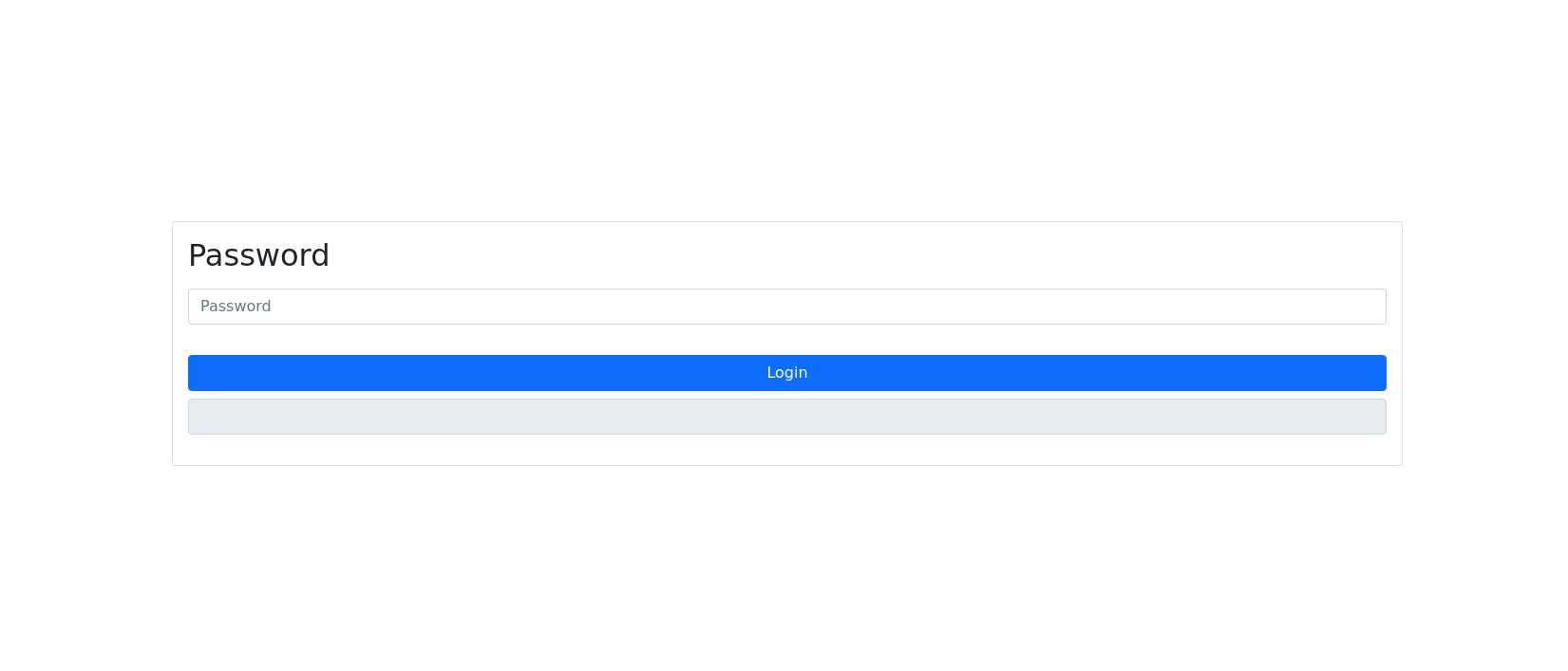
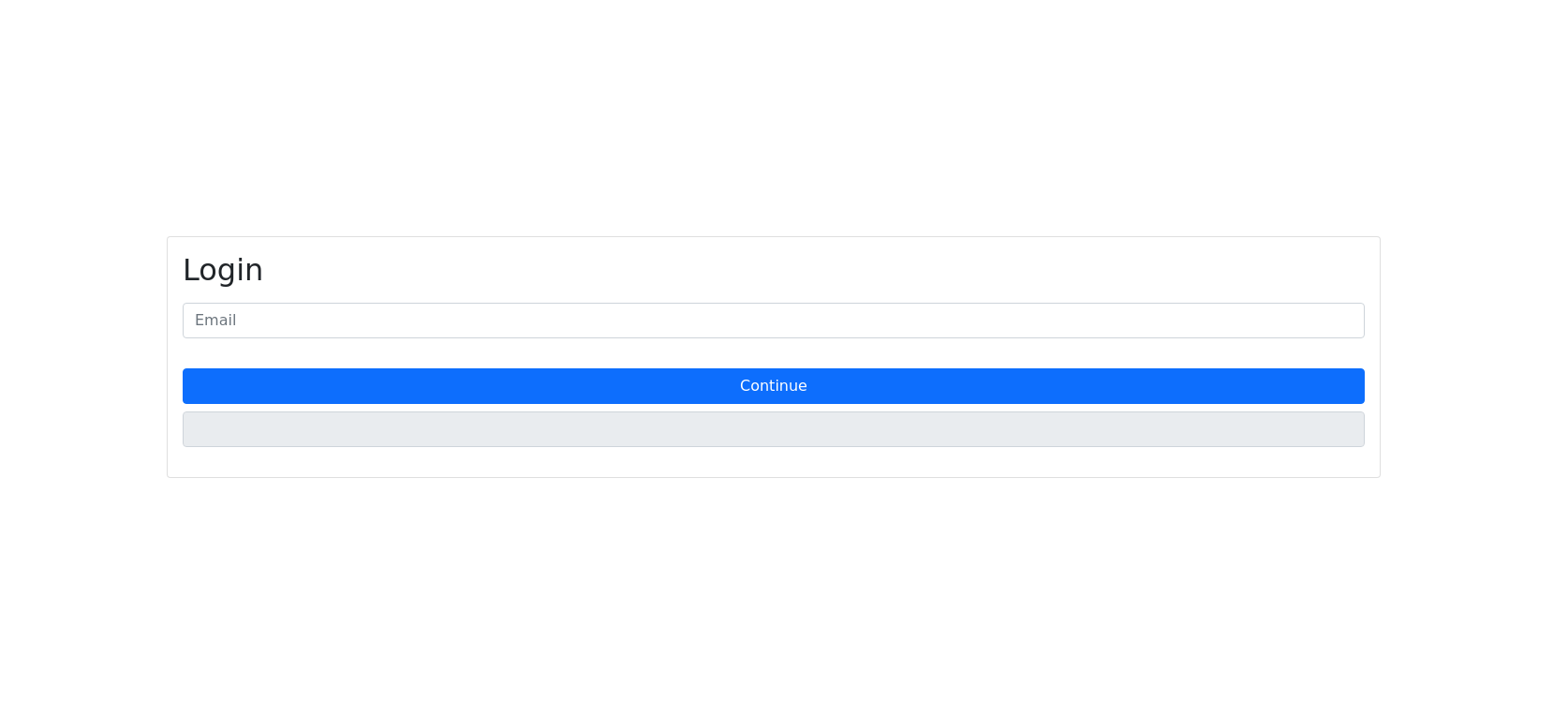
|  |
| --- |
| <!DOCTYPE html> <html lang="en">  <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1" />   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.slim.min.js"></script>  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>  <script src="/static/js/login.js"></script>   <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"  rel="stylesheet"  integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"  crossorigin="anonymous"  />  <link rel="stylesheet" href="/static/css/main.css" />   <title>Login</title>  </head>  <body>  <div class="container-md mx-auto vertical-center">  <div class="col">  <form class="card">  <div class="card-body">  <h2 class="mb-3">Login</h2>  <div class="row g-3 mb-3">  <div class="mb-3">  <input  type="email"  class="form-control"  id="email-input"  placeholder="Email"  />  </div>  <div class="d-grid gap-2">  <button  onclick="checkEmail(this)"  type="button"  id="continue-button"  class="btn btn-primary"  >  Continue  </button>  <input  id="errors"  class="form-control"  type="text"  readonly  />  </div>  </div>  </form>  </div>  </div>  <script  src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle.min.js"  integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroafW"  crossorigin="anonymous"  ></script>  </body> </html> |

And the html for the password page

|  |
| --- |
| <!DOCTYPE html> <html lang="en">  <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1" />   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.slim.min.js"></script>  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>  <script src="/static/js/login.js"></script>   <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"  rel="stylesheet"  integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"  crossorigin="anonymous"  />  <link rel="stylesheet" href="/static/css/main.css" />   <title>Login</title>  </head>  <body>  <div class="container-md mx-auto vertical-center">  <div class="col">  <form class="card">  <div class="card-body">  <h2 class="mb-3">Password</h2>  <p id="password-label"></p>  <div class="row g-3 mb-3">  <div class="mb-3">  <input  type="password"  class="form-control"  id="pass-input"  placeholder="Password"  />  </div>  <div class="d-grid gap-2">  <button  onclick="login(this)"  type="button"  id="login-button"  class="btn btn-primary"  >  Login  </button>  <input  id="errors"  class="form-control"  type="text"  readonly  />  </div>  </div>  </form>  </div>  </div>  <script  src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle.min.js"  integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroafW"  crossorigin="anonymous"  ></script>  </body> </html> |

And the JavaScript that is used for both pages

|  |
| --- |
| var storage; // Two global variables for the storage variables var code; // and the code if a password is unset  // A shorthand for window.onload in jquery $(() => {  storage = window.sessionStorage; // set the storage variable  code = storage.getItem("\_cde") // get the code from storage  if(code !== "\_") { // check if the code already exists, an underscore means the user already has a password  $("#password-label").text("Create a new password for your account")  } else { // No underscore means the account doesnt have a password  code = undefined  } })  function checkEmail(el) {  data = { // Check the email to get a code or confirm that a password is already set  email: $("#email-input").val()  }   // Make a request to check  axios  .post("/api/login/pass", data)  .then((resp) => { // Set all the appropriate storage slots if success  var code = resp.data.data  storage.setItem("\_eml", data.email)  storage.setItem("\_cde", code)  window.location.replace("/login/pass")  })  .catch((err) => { // Check for error  var status = err.response.status  if(status == 410) { // The server returns 410 GONE if the password is already set  storage.setItem("\_eml", data.email)  storage.setItem("\_cde", "\_") // Set the code as \_  window.location.replace("/login/pass")  } else { // If some other error occurred then notify user  $(el).addClass("border-danger")  $("#errors").val(err.response.data.error)  }  }) }  function login() {  // Disable button and make a spinner  $("#login-button").addClass("disabled")  $("#login-button").prepend($("<span id=\"waiter\" class=\"spinner-grow spinner-grow-sm\" role=\"status\" aria-hidden=\"true\"></span>"))   // all the data the user sends  var data = {  email: storage.getItem("\_eml"),  password: $("#pass-input").val(),  code: code // If undefined it wont be included  }   // Make a request  axios  .post("/api/login", data)  .then(() => { // On success remove the code from the user for security reasons  storage.removeItem("\_eml")  storage.removeItem("\_cde")  window.location.replace("/dashboard") // Move to dashboard  })  .catch((err) => { // Notify of error  $("#pass-input").addClass("border-danger")  $("#errors").val(err.response.data.error)  $("#waiter").remove()  $("#login-button").removeClass("disabled")  }) } |

This is what the login and password pages look like

**6.3 Dashboard**

The dashboard is the main way that the user interacts with the app. It shows all the immediate information, such as the student’s assignments, the teachers subjects or the headmaster’s schools.

It is mostly based on templating, and has very little html code, and the code is split into four files

The code for all interactions

|  |
| --- |
| var dashboard;  const PERMS = { // A map of permissions and their corresponding number  Headmaster: 1,  Teacher: 2,  Student: 3, };  // Currently selected tab var selected = 0;  // On window load $(() => {  $("#waiter").show(); // Show a waiter until all the information is retrieved from the server  $("#content").hide();   // Create an event when the content is loaded  // When the event is triggered the loadWorkspace is run  $("#content").on("complete", loadWorkspace);   // Clear everything in the local storage  window.localStorage.clear();   // Get the information for the dashboard  axios  .get("/api/dashboard")  .then((res) => {  dashboard = res.data.data;  try {  // Show the content  $("#content").trigger("complete");  } catch (err) {  // If an error occurs in the loadWorkspace function  // it is caught here  console.error(err);  }  })  // If an error occurred then send the user to the login page  .catch((err) => {  console.error(err);  window.location.replace("/login");  }); });  function loadWorkspace() {  // Remove all waiters and show the content  $("#waiter").remove();  $("#content").show();  // Get the last selected tab from a cookie  selected = Cookies.get("\_lsel");   // Get the user's name  $("#name").text(dashboard.user.first\_name);   // Set the currently selected role and set it as active  // in the navbar  if (dashboard.headmaster) {  var template = `<li class="nav-item">  <b onclick="selectRole(1)" class="nav-link clickable active">Headmaster</b>  </li>`;  $("#navbar").prepend($(template));  }  if (dashboard.teacher) {  var template = `<li class="nav-item">  <b onclick="selectRole(2)" class="nav-link clickable active">Teacher</b>  </li>`;  $("#navbar").prepend($(template));  }  if (dashboard.student) {  var template = `<li class="nav-item">  <b onclick="selectRole(3)" class="nav-link clickable active">Student</b>  </li>`;  $("#navbar").prepend($(template));  }   // Update selection will load in all the necessary info for the selected role  updateSelection(); }  function updateSelection() {   // Set the cookie for the last selected role  if (!selected || selected == 0 || selected > 3) {  if (dashboard.headmaster) selected = PERMS.Headmaster;  else if (dashboard.teacher) selected = PERMS.Teacher;  else if (dashboard.student) selected = PERMS.Student;  Cookies.set("\_lsel", selected, { path: "/dashboard" });  }   // Clear the screen  $("#data").empty();    // Insert the correct data depending on selected role  if (selected == PERMS.Headmaster) {  // Title  $("#data").append($("<h1 id='title'>Headmaster</h1>"));   // insert all the schools the headmaster owns  dashboard.headmaster.schools.forEach((school) => {  template = schoolTemplate(school.id, school.name);  $("#data").append($(template));  1;  });   // Append the form to add a new school  $("#data").append(  $(`<div class="row">  <div class="col-10">  <input  type="text"  class="form-control"  id="school-name"  placeholder="School name"  />   <input  id="errors"  type="text"  class="form-control"  readonly  />  </div>  <div class="col-2 d-grid">  <button id="schooladder" onclick="addSchool()" class="btn btn-primary">  <i id="plus" class="fas fa-plus"></i>  </button>  </div>  </div>`)  );  }  if (selected == PERMS.Teacher) {  $("#data").append($("<h1 id='title'>Teacher</h1>"));   // Insert all the subjects  dashboard.teacher.subjects.forEach((subject) => {  template = subjectTemplate(subject.id, subject.name);  $("#data").append($(template));  1;  });   // Insert the form to add a new subject  $("#data").append(  $(`<div class="row">  <div class="col-10">  <input  type="text"  id="subject-name"  class="form-control mb-1"  placeholder="Subject name"  />   <input  id="errors"  class="form-control"  type="text"  readonly  />  </div>  <div class="col-2 d-grid">  <button id="subjectadder" onclick="addSubject()" class="btn btn-primary">  <i id="plus" class="fas fa-plus"></i>  </button>  </div>  </div>`)  );  }  if (selected == PERMS.Student) {  // Title  $("#data").append($("<h1 id='title'>Student</h1>"));   // For each subject  dashboard.student.subjects.forEach((subject) => {  $("#data").append($(subjTemplate(subject.id, subject.name)));  // For each assignment in the subject  subject.assignments.forEach((assgn) => {  // Insert all the requests and count the ones that are missing  var remaining = 0;  assgn.requests.forEach((req) => {  if (!req.complete) remaining++;  });  $("#" + subject.id).append(  // Append them all to the subject  assgnTemplate(assgn.id, assgn.name, assgn.text, subject.id, remaining)  );  });  });  } }  function logout() {  // send a request to the server  axios  .post("/api/logout")  .then() // empty then because it will only run in case of a success  .then(() => { // Second then always runs  window.location.replace("/login");  }); }  function selectRole(role) {  selected = role; // set the role  Cookies.set("\_lsel", role, { path: "/dashboard" }); // update the cookie  updateSelection(); // update the selection } |

The javascript for the headmaster role

|  |
| --- |
| function addSchool() {  data = { // Get the info to send to the server  name: $("#school-name").val(),  };   // Disable the button while the request is running  $("#schooladder").addClass("disabled");  // Create a waiter  $("#schooladder").prepend(  $(  '<span id="waiter" class="spinner-grow spinner-grow-sm" role="status" aria-hidden="true"></span>'  )  );  // Remove the plus icon  $("#plus").remove();   // Send the request  axios  .post("/api/school", data)  .then((res) => {  var school = res.data.data;  // Add the school at the top of the page  template = schoolTemplate(school.id, school.name);  $("#title").after($(template));  // Remove all errors and borders  $("#errors").val("");  $("#school-name").val("");  $("#schooladder").removeClass("disabled");  $("#school-name").removeClass("border-danger");  })  .catch((err) => { // In case of error  // Add a danger border and show error  $("#school-name").addClass("border-danger");  $("#errors").val(err.response.data.error);  $("#schooladder").removeClass("disabled");  })  .then(() => { // Add the plus and remove the waiter  $("#schooladder").append(`<i id="plus" class="fas fa-plus"></i>`);  $("#waiter").remove();  }); }  // School html template function schoolTemplate(id, name) {  return `<div id="${id}" class="card mb-3">  <div class="row g-0">  <div class="col-sm-8">  <div class="card-body">  <h5 class="card-title mt-4">${name}</h5>  </div>  </div>  <div class="col-sm-4 mt-4 mb-4">  <div class="row mb-2">  <div class="col">  <button onclick="manageSchool('${id}')" class="btn btn-primary">Manage</button>  </div>  </div>  <div class="row">  <div class="col">  <button onclick="deleteSchool('${id}')" class="btn btn-primary">Delete</button>  </div>  </div>  </div>  </div>  </div> </div>`; }  function deleteSchool(id) {  // Send a request to delete the school  axios  .delete("/api/school/subject?id=" + encodeURI(id))  .then(() => {  $("#" + id).remove();  })  .catch((err) => {  $("#errors").val(err.response.data.error);  }); }  function manageSchool(id) {  // Set the storage item to the school ID for reference  window.localStorage.setItem("\_sch", id);  // Move to the /school page  window.location.href = "/school"; } |

The javascript for the teacher functionality

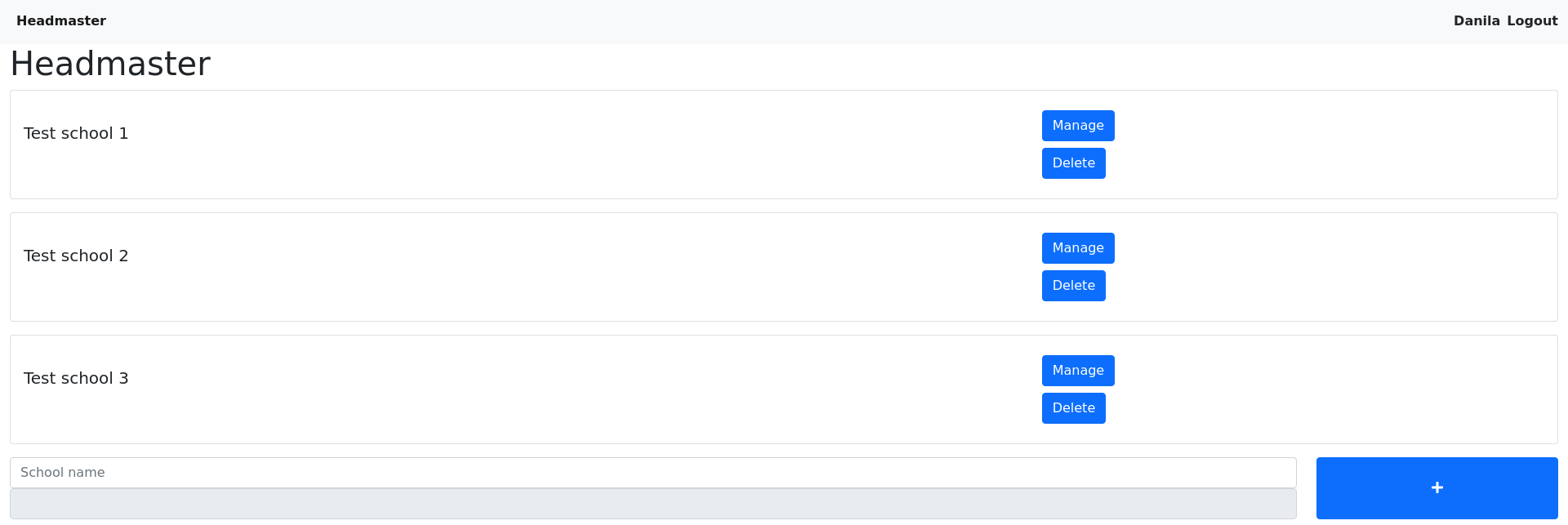
|  |
| --- |
| function addSubject() {  data = { // Get the data for the subject  name: $("#subject-name").val(),  school\_id: dashboard.teacher.school\_id, // Get the school ID from the teacher  };   // Disable the button  $("#subjectadder").addClass("disabled");  // Add a waiter  $("#subjectadder").prepend(  $(  '<span id="waiter" class="spinner-grow spinner-grow-sm" role="status" aria-hidden="true"></span>'  )  );  // Remove the plus icon  $("#plus").remove();   // Send a request to create a new subject  axios  .post("/api/school/subject", data)  .then((res) => {  var subject = res.data.data;  // Add the subject at the top  template = subjectTemplate(subject.id, subject.name);  $("#title").after($(template));   // Clear all errors and borders  $("#errors").val("");  $("#subject-name").val("");  $("#subjectadder").removeClass("disabled");  $("#subject-name").removeClass("border-danger");  })  .catch((err) => {  // Set errors  $("#subject-name").addClass("border-danger");  $("#errors").val(err.response.data.error);  $("#subjectadder").removeClass("disabled");  })  .then(() => {  // remove the waiter and add the plus icon  $("#subjectadder").append(`<i id="plus" class="fas fa-plus"></i>`);  $("#waiter").remove();  }); }  // A template for a subject card function subjectTemplate(id, name) {  return `<div id="${id}" class="card mb-3">  <div class="row g-0">  <div class="col-sm-8">  <div class="card-body">  <h5 class="card-title mt-4">${name}</h5>  </div>  </div>  <div class="col-sm-4 mt-4 mb-4">  <div class="row mb-2">  <div class="col">  <button onclick="manageSubject('${id}')" class="btn btn-primary">Manage</button>  </div>  </div>  <div class="row">  <div class="col">  <button onclick="deleteSubject('${id}')" class="btn btn-primary">Delete</button>  </div>  </div>  </div>  </div>  </div> </div>`; }  function deleteSubject(id) {  // Send a request to delete a subject  axios  .delete("/api/school/subject?id=" + encodeURI(id))  .then(() => {  $("#" + id).remove();  })  .catch((err) => {  $("#errors").val(err.response.data.error);  }); }  function manageSubject(id) {  // Set the subject and school ID for future reference  window.localStorage.setItem("\_sbj", id)  window.localStorage.setItem("\_sch", dashboard.teacher.school\_id)  // Redirect user  window.location.href = "/subject"; } |

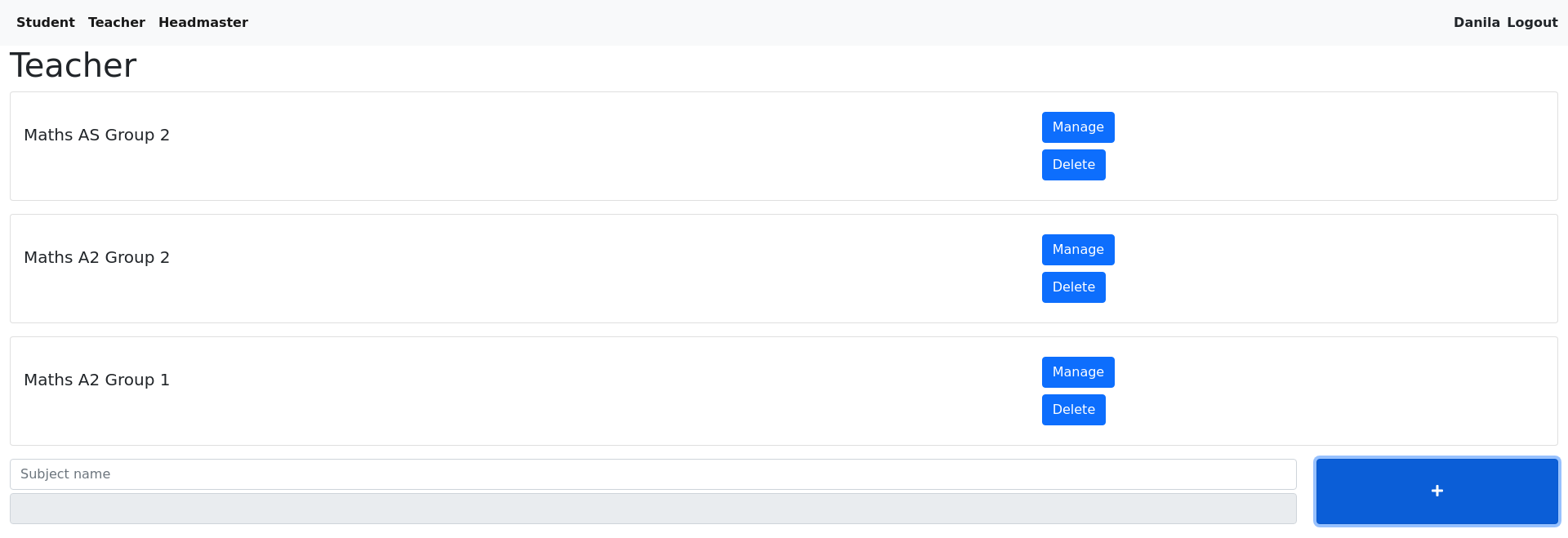
The functions for the student dashboard

|  |
| --- |
| // Template for a subject function subjTemplate(subjid, subjname) {  return `<div id="${subjid}">  <h3 class="mt-3">${subjname}</h3>    </div>`; }  // Template for an assignment function assgnTemplate(id, name, text, subjid, remaining) {  return `<div class="card mb-3">  <div class="card-body">  <h4>${name}</h4>  <p>${text}</p>  <small class="text-muted">${remaining} parts remaining</small  ><br /><br />  <button onclick="viewAssgn('${id}', '${subjid}')" class="btn btn-primary">  View  </button>  </div>  </div>`; }  // View assignment function viewAssgn(id, sid) {  // Set all appropritate fields for future reference  window.localStorage.setItem("\_sub", sid);  window.localStorage.setItem("\_asn", id);  // Indicate that a student is viewing the assignment  window.localStorage.setItem("\_rol", "cNFDBScBBJ8=");  // Redirect  window.location.href = "/assignment"; } |

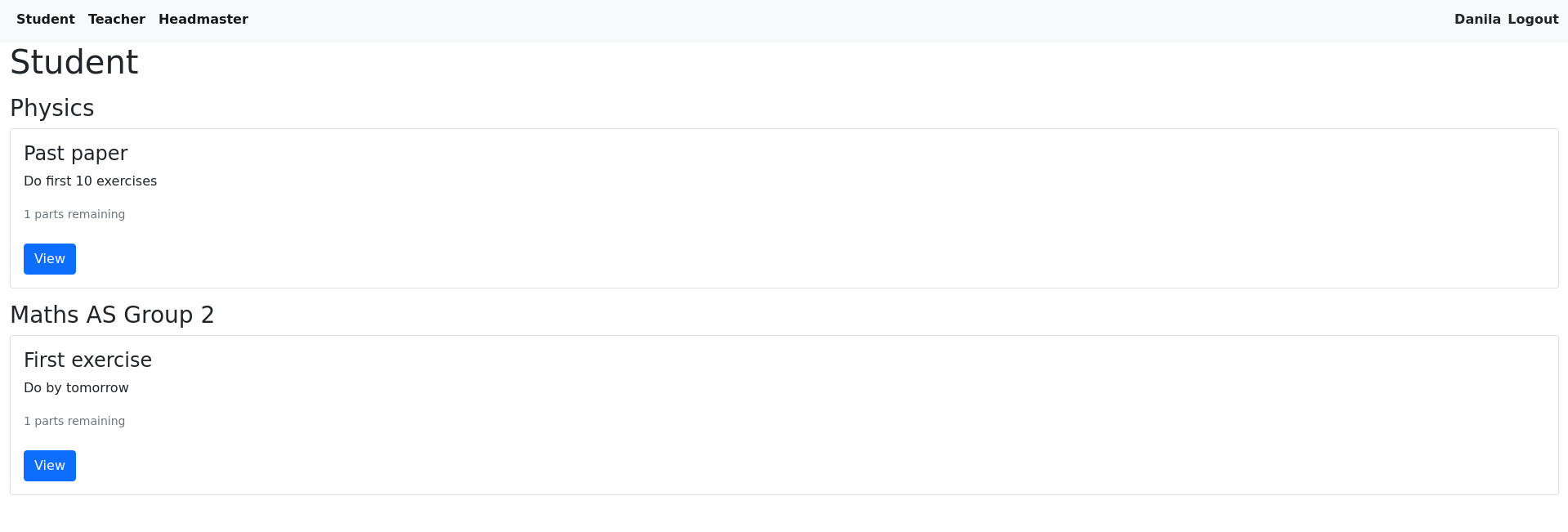
This is what a dashboard may look like

For a headmaster



For a teacher

For a student



**6.4 The school page**

This page is only accessible to the headmaster, where he can add or remove people from the school, specifically teachers and students

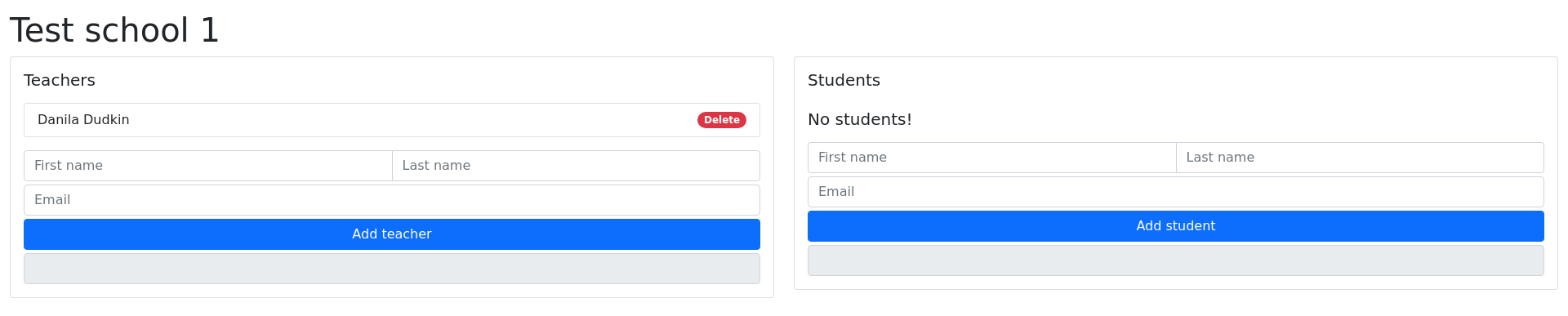
The HTML for the page looks like this

|  |
| --- |
| <!DOCTYPE html> <html>  <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1" />   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.slim.min.js"></script>  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>  <script src="/static/js/school.js"></script>  <script src="https://cdn.jsdelivr.net/npm/js-cookie@rc/dist/js.cookie.min.js"></script>   <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"  rel="stylesheet"  integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"  crossorigin="anonymous"  />  <link  rel="stylesheet"  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/all.min.css"  integrity="sha512-HK5fgLBL+xu6dm/Ii3z4xhlSUyZgTT9tuc/hSrtw6uzJOvgRr2a9jyxxT1ely+B+xFAmJKVSTbpM/CuL7qxO8w=="  crossorigin="anonymous"  />  <link rel="stylesheet" href="/static/css/main.css" />  </head>  <body>  <div id="waiter" class="position-relative vertical-center">  <div class="position-absolute top-50 start-50 translate-middle">  <div class="spinner-grow" role="status">  <span class="visually-hidden">Loading...</span>  </div>  </div>  </div>   <div class="container-fluid mt-3" id="content">  <h1 id="school\_name"></h1>   <div class="row">  <div class="col-sm-6 mb-3">  <div class="card">  <div class="card-body" id="teachers">  <h5 class="card-title mb-3">Teachers</h5>  <div id="teacher-form" class="input-group mb-1 mt-3">  <input  id="teacher-fname"  class="form-control"  placeholder="First name"  />  <input  id="teacher-lname"  class="form-control"  placeholder="Last name"  />  </div>  <input  id="teacher-email"  type="email"  class="form-control mb-1"  placeholder="Email"  />  <button onclick="add('teacher')" id="teacher-add" class="btn btn-primary col-12 mb-1">  Add teacher  </button>  <input id="teacher-errors" class="form-control" type="text" readonly />  </div>  </div>  </div>  <div class="col-sm-6">  <div class="card">  <div class="card-body" id="students">  <h5 class="card-title mb-3">Students</h5>  <div id="student-form" class="input-group mb-1 mt-3">  <input  id="student-fname"  class="form-control"  placeholder="First name"  />  <input  id="student-lname"  class="form-control"  placeholder="Last name"  />  </div>  <input  id="student-email"  type="email"  class="form-control mb-1"  placeholder="Email"  />  <button onclick="add('student')" id="student-add" class="btn btn-primary col-12 mb-1">  Add student  </button>  <input id="student-errors" class="form-control" type="text" readonly />  </div>  </div>  </div>  </div>  </div>  </body> </html> |

And the JavaScript that runs on the page

|  |
| --- |
| var school\_id = window.localStorage.getItem("\_sch"); // Get the school ID  var schoolData;  // When the page loads $(() => {  // If the school ID doesnt exist then return to dashboard  if (!school\_id) {  window.location.href = "/dashboard";  }   // Show a waiter  $("#waiter").show();  $("#content").hide();   // Add an event when loading is complete  $("#content").on("complete", loadWorkspace);   // The data required for a GET request  var data = {  id: school\_id  }   // Make the network request  axios  .get("/api/school", {  params: data  })  .then((res) => {  schoolData = res.data.data;  try {  $("#content").trigger("complete");  } catch (err) {  console.error(err);  }  })  .catch((err) => {  console.error(err);  window.localStorage.removeItem("\_sch");  window.location.replace("/dashboard");  }); });  // Load the workspace function loadWorkspace() {  $("#school\_name").text(schoolData.name);   // If there are no teachers then show a message  if (!schoolData.teachers) {  $("#teacher-form").before($("<h5 id='teacher-heading'>No teachers!</h5>"));  } else {  // Otherwise list all the teachers  $("#teacher-form").before(  $(`<ul id="teacher-list" class="list-group mb-2"></ul>`)  );  schoolData.teachers.forEach((t) => {  $("#teacher-list").append(  $(template("teacher", t.id, t.first\_name + " " + t.last\_name))  );  });  }   // Same for students  if (!schoolData.students) {  $("#student-form").before($("<h5 id='student-heading'>No students!</h5>"));  } else {  $("#student-form").before(  $(`<ul id="student-list" class="list-group mb-2"></ul>`)  );  schoolData.students.forEach((s) => {  $("#student-list").append(  $(template("student", s.id, s.first\_name + " " + s.last\_name))  );  });  }   // Show all the content  $("#waiter").remove();  $("#content").show(); }  function template(who, id, name) {  // Template for one the list elements of students and teachers  t = `  <li  class="list-group-item d-flex justify-content-between align-items-center"  id="${who + '\_' + id}"  >  ${name}  <span onclick="remove('${who}', '${id}', this)" class="badge bg-danger clickable rounded-pill">Delete</span>  </li>`;  return t; }  function add(who) {  // who acts like selector of whether you want to add  // a student or a teacher  $("#" + who + "-errors").val("");  $("#" + who + "-add").addClass("disabled")  schoolData = {  first\_name: $("#" + who + "-fname").val(),  last\_name: $("#" + who + "-lname").val(),  email: $("#" + who + "-email").val(),  school\_id: school\_id,  };   // If there were no teachers in the beginning, then remove the header  // that says there are no teachers  if ($("#" + who + "-list").length == 0) {  $("#" + who + "-heading").remove();  $("#" + who + "-form").before(  $(`<ul id="${who}-list" class="list-group mb-2"></ul>`)  );  }   // Make the request  axios  .post("/api/school/" + who, schoolData)  .then((res) => {  // Add the new teacher to the list  t = template(  who,  res.data.data.id,  res.data.data.first\_name + " " + res.data.data.last\_name  );  $("#" + who + "-list").append($(t));  // Clear the form and enable the button  $("#" + who + "-fname").val("");  $("#" + who + "-lname").val("");  $("#" + who + "-email").val("");  $("#" + who + "-add").removeClass("disabled")  })  .catch((err) => {  // Show the error to the user  $("#" + who + "-errors").val(err.response.data.error);  $("#" + who + "-add").removeClass("disabled")  }); }  function remove(who, id, button) {  // Remove the student or the teacher  $("#" + who + "-errors").val("");  el = $(button)  el.removeClass("bg-danger")  el.addClass("bg-secondary")  // Make a delete request  axios  .delete("/api/school/" + who + "?sid=" + school\_id + "&uid=" + id)  .then(() => {  // remove the list element with the teacher  $("#" + who + "\_" + id).remove();  let amount = $("#" + who + "-list > li").length;  // Show if there is 0 elements of the list left  if (amount == 0) {  $("#" + who + "-form").before($(`<h5 id='${who}-heading'>No ${who}s!<h5>`));  }  })  .catch((err) => {  // Show all errors  el.addClass("bg-danger")  el.removeClass("bg-secondary")  $("#" + who + "-errors").val(err.response.data.error);  }); } |

The page looks like this



**6.5 The subject page**

The subject page is viewed by the teacher, to create new assignments and check old ones

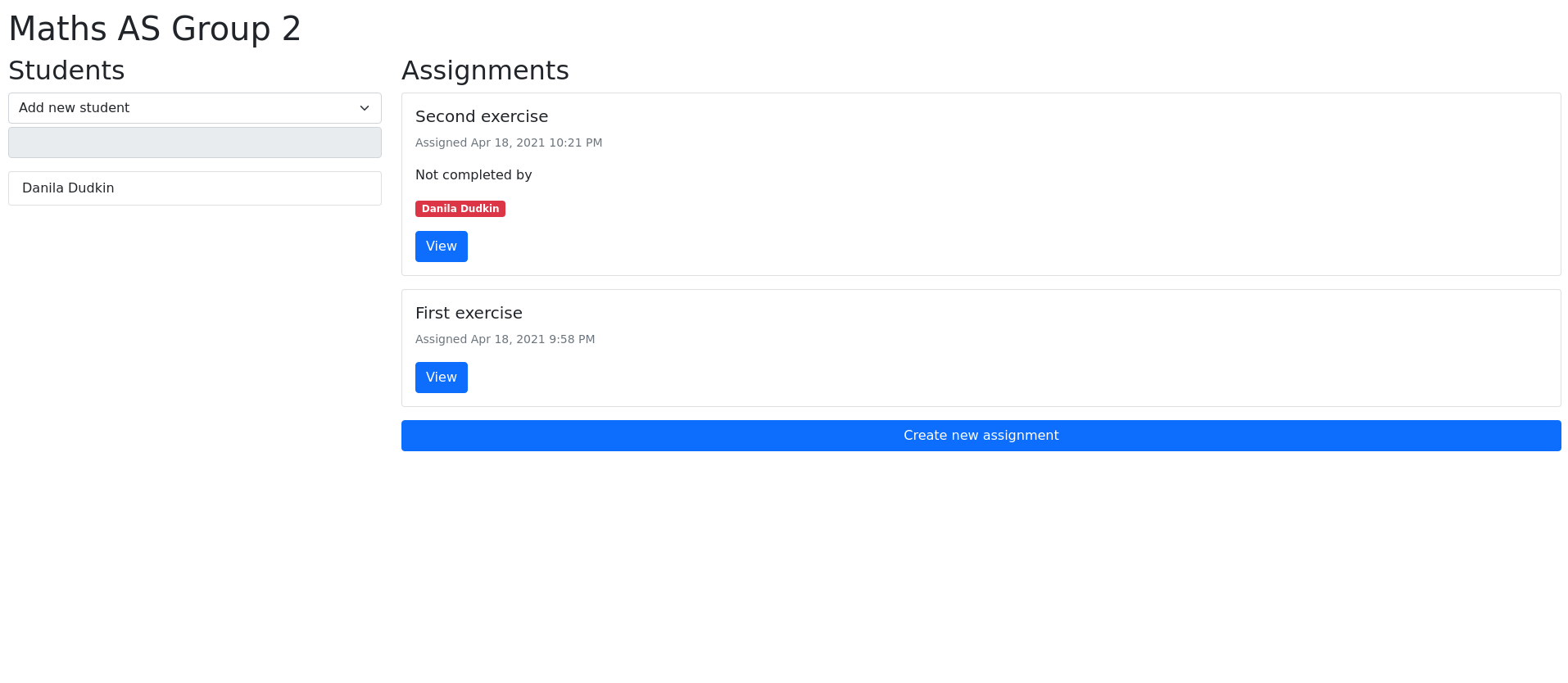
HTML for the page

|  |
| --- |
| <!DOCTYPE html> <html>  <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1" />   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.slim.min.js"></script>  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/js-cookie@rc/dist/js.cookie.min.js"></script>  <script  src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle.min.js"  integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroafW"  crossorigin="anonymous"  ></script>  <script  src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.1/moment.min.js"  integrity="sha512-qTXRIMyZIFb8iQcfjXWCO8+M5Tbc38Qi5WzdPOYZHIlZpzBHG3L3by84BBBOiRGiEb7KKtAOAs5qYdUiZiQNNQ=="  crossorigin="anonymous"  ></script>  <script src="https://momentjs.com/downloads/moment-timezone-with-data.js"></script>  <script src="/static/js/subject.js"></script>   <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"  rel="stylesheet"  integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"  crossorigin="anonymous"  />  <link  rel="stylesheet"  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/all.min.css"  integrity="sha512-HK5fgLBL+xu6dm/Ii3z4xhlSUyZgTT9tuc/hSrtw6uzJOvgRr2a9jyxxT1ely+B+xFAmJKVSTbpM/CuL7qxO8w=="  crossorigin="anonymous"  />  <link rel="stylesheet" href="/static/css/main.css" />  </head>  <body class="mb-3">  <div id="waiter" class="position-relative vertical-center">  <div class="position-absolute top-50 start-50 translate-middle">  <div class="spinner-grow" role="status">  <span class="visually-hidden">Loading...</span>  </div>  </div>  </div>   <div class="container-fluid mt-3" id="content">  <h1 id="name"></h1>  <div class="row">  <div class="col-lg-3 mb-4">  <h2>Students</h2>  <select  id="students\_select"  class="form-select mb-1"  onchange="addStudent(this)"  >  <option value="0">Add new student</option>  </select>  <input id="errors" class="form-control mb-3" type="text" readonly />  <ul id="students" class="list-group"></ul>  </div>  <div class="col-lg-9">  <h2>Assignments</h2>  <div id="assignments" class="mb-3">    </div>  <div class="d-grid">  <button  type="button"  class="btn btn-primary"  onclick = "modal.show()"  >  Create new assignment  </button>  </div>   <div  class="modal fade"  id="assignmentModal"  data-bs-backdrop="static"  data-bs-keyboard="false"  tabindex="-1"  >  <div class="modal-dialog">  <div class="modal-content">  <div class="modal-header">  <h5 class="modal-title" id="staticBackdropLabel">  Create assignment  </h5>  <button  type="button"  class="btn-close"  data-bs-dismiss="modal"  aria-label="Close"  ></button>  </div>  <div class="modal-body" id="subject-form">  <input  id="asn\_name"  class="form-control mb-1"  type="text"  placeholder="Assignment title"  />  <textarea  placeholder="Assignment text..."  class="form-control mb-3"  id="text"  rows="3"  ></textarea>  <input  id="timedue"  class="form-control mb-1"  type="datetime-local"  />  <input  id="files"  class="form-control mb-3"  type="file"  multiple  />   <p>Upload requests</p>  <ul  id="uplRequests"  class="list-group list-group-flush mb-4"  ></ul>  <div class="row row-cols-md-auto g-3">  <div class="col-6">  <button  onclick="addUploadRequest()"  class="btn btn-primary"  >  Add upload request  </button>  </div>  <div class="col-6">  <input  id="requestName"  class="form-control mb-1"  type="text"  placeholder="Name"  />  </div>  </div>  </div>  <div class="modal-footer">  <div class="row">  <div class="col-7">  <input id="errors\_asn" class="form-control-plaintext" />  </div>  <div class="col-5">  <button  type="button"  class="btn btn-primary"  onclick="addAssignment()"  >  Create  </button>  </div>  </div>  </div>  </div>  </div>  </div>  </div>  </div>  </div>  </body> </html> |

The javascript that runs on the page:

|  |
| --- |
| var subject; // Global variables var students; var subject\_id = window.localStorage.getItem("\_sbj"); var school\_id = window.localStorage.getItem("\_sch"); var modal;  var upload\_req = { // Data for an upload request  name: "",  text: "",  subject\_id: "",  time\_due: "",  files: [],  uploads: [], };  $(() => {  // Set the modal variable (the window that allows the user to create a new assignment)  modal = new bootstrap.Modal(document.getElementById("assignmentModal"));  // Show all the waiters  $("#waiter").show();  $("#content").hide();   // Create an event  $("#content").on("complete", loadWorkspace);   window.localStorage.removeItem("\_asn");   // Params to get all assignments  var params = {  id: subject\_id,  sid: school\_id,  };   // Make the request  axios  .get("/api/school/subject/", {  params: params,  })  .then((res) => {  subject = res.data.data.subject;  students = res.data.data.students;   try {  $("#content").trigger("complete");  } catch (err) {  console.error(err);  }  })  .catch((err) => {  console.error(err);  console.error(err.response.data.error);  window.location.replace("/dashboard");  }); });  // Template for the assignment function asnTemplate(id, name, date, needsUpload) {  var text = needsUpload  ? `<p>Not completed by</p><div class="mb-3" id="${id}\_nc"></div>`  : "";   return `<div class="card mb-3" id="${id}">  <div class="card-body">  <h5 class="card-title">${name}</h5>  <p class="card-text"><small class="text-muted">Assigned ${date} </small></p>   ${text}   <a onclick="viewAssignment('${id}')" class="btn btn-primary">View</a>  </div> </div>`; }  function loadWorkspace() {  $("#waiter").remove(); // Show the content  $("#content").show();  $("#name").text(subject.name); // Set the name of the subject   // Place in a try because random errors occur sometimes  try {  // Add each student in the subject to a list  subject.students.forEach((student) => {  $("#students").append(  `<li value="${student.id}" class="list-group-item">${  student.first\_name + " " + student.last\_name  }</li>`  );  });  } catch {}   // Set the locale for the time  var locale = window.navigator.userLanguage || window.navigator.language;  moment.locale(locale);   // If the subject has any assignments  if (subject.assignments) {  // Display each assignment with a template  subject.assignments.forEach((assgn) => {  // Get the time when each was assigned  const date = moment(assgn.time\_assigned).format("lll");  // use a template  var t = asnTemplate(assgn.id, assgn.name, date, assgn.not\_completed\_by);  $("#assignments").append($(t));   // If there are students who didnt complete an assignment  if (assgn.not\_completed\_by != null) {  assgn.not\_completed\_by.forEach((ncstd) => {  $("#" + assgn.id + "\_nc").append(  `<span class="badge bg-danger">${  ncstd.first\_name + " " + ncstd.last\_name  }</span>`  );  });  }  });  }   // Add all students in the school to a selection list  // to add them to the subject  students.forEach((student) => {  $("#students\_select").append(  $(  `<option value="${student.id}">${  student.first\_name + " " + student.last\_name  }</option>`  )  );  }); }  // Add a student to a subject function addStudent(el) {  $("#errors").val("");  if (el.value == "0") return;   data = { // All the necessary data  user\_id: el.value,  subject\_id: subject\_id,  };   // Make the request  axios  .post("/api/school/subject/students", data)  .then((res) => {  student = res.data.data;  // Add the student to the list  $("#students").append(  `<li value="${student.id}" class="list-group-item">${  student.first\_name + " " + student.last\_name  }</li>`  );  })  .catch((err) => {  $("#errors").val(err.response.data.error);  }); }  // Add a new upload request function addUploadRequest() {  // Get the name of the request  var name = $("#requestName").val();  // Name cannot be empty  if (name.length == 0) {  return;  }   // Set all form elements to 0  $("#requestName").val("");  // Create a new list of elements  var el = $("<li></li>");  el.text(name);  el.addClass("list-group-item");  el.addClass("d-flex");  el.addClass("justify-content-between");  el.addClass("align-items-center");  el.attr("id", upload\_req.uploads.length);   // Add the name of the request to the request list  upload\_req.uploads.push(name);   // Add a delete button  el.append(  $(  `<span onclick="removeUploadReq(this)" class="badge bg-danger rounded-pill">Delete</span>`  )  );  // Add the request to the screen  $("#uplRequests").append(el); }  function removeUploadReq(el) {  // Delete the request from the list  var parent = $(el).parent();  upload\_req.uploads.splice(parent.attr("id"), 1);  parent.remove(); }  function addAssignment() {  // First upload the file  var formData = new FormData();  var files = $("#files").prop("files");  $.each(files, (\_, file) => {  formData.append("files", file);  });   // Create a POST request to the files route  axios  .post("/files", formData, {  headers: {  "Content-Type": "multipart/form-data",  },  })  .then((res) => {  // After file was uploaded then set all necessary values  // Path to files on server  upload\_req.files = res.data.files;  // Set the needed values in the upload\_req value  upload\_req.name = $("#asn\_name").val();  upload\_req.subject\_id = subject\_id;  upload\_req.text = $("#text").val();   // Set the time due into a valid format  var time\_due = moment.tz($("#timedue").val(), moment.tz.guess());   upload\_req.time\_due = time\_due;  uploadAssgn(upload\_req);  })  .catch((err) => {  // Notify user of error  console.error(err);  $("#errors\_asn").val(err.response.data.error);  }); }   function uploadAssgn(data) {  // Create a post request with all information from before  axios  .post("/api/school/subject/assignment", data)  .then((res) => {  var assgn = res.data.data;  var time\_created = moment.tz($("#timedue").val(), moment());  $("#assignments").append(asnTemplate(assgn.id, assgn.name, time\_created));  modal.hide();  })  .catch((err) => {  console.error(err);  $("#errors\_asn").val(err.response.data.error);  }); }  // View the assignment function viewAssignment(id) {  window.localStorage.setItem("\_asn", id);  // Set role as teacher  window.localStorage.setItem("\_rol", "nqrOzz0jmxA=");  // Redirect  window.location.href = "/assignment"; } |

This is what the subject page will look like for a teacher



**6.6 Assignment page**

The assignment page can be viewed by both students and teachers, and contains information necessary for either person

The HTML for the page

|  |
| --- |
| <!DOCTYPE html> <html>  <head>  <meta charset="utf-8" />  <meta name="viewport" content="width=device-width, initial-scale=1" />   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.slim.min.js"></script>  <script src="https://unpkg.com/axios/dist/axios.min.js"></script>  <script src="https://cdn.jsdelivr.net/npm/js-cookie@rc/dist/js.cookie.min.js"></script>  <script  src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle.min.js"  integrity="sha384-ygbV9kiqUc6oa4msXn9868pTtWMgiQaeYH7/t7LECLbyPA2x65Kgf80OJFdroafW"  crossorigin="anonymous"  ></script>  <script  src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.1/moment.min.js"  integrity="sha512-qTXRIMyZIFb8iQcfjXWCO8+M5Tbc38Qi5WzdPOYZHIlZpzBHG3L3by84BBBOiRGiEb7KKtAOAs5qYdUiZiQNNQ=="  crossorigin="anonymous"  ></script>  <script src="https://momentjs.com/downloads/moment-timezone-with-data.js"></script>  <script id="t" src="/static/js/assignment-t.js"></script>  <script id="st" src="/static/js/assignment-st.js"></script>   <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css"  rel="stylesheet"  integrity="sha384-giJF6kkoqNQ00vy+HMDP7azOuL0xtbfIcaT9wjKHr8RbDVddVHyTfAAsrekwKmP1"  crossorigin="anonymous"  />  <link  rel="stylesheet"  href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.2/css/all.min.css"  integrity="sha512-HK5fgLBL+xu6dm/Ii3z4xhlSUyZgTT9tuc/hSrtw6uzJOvgRr2a9jyxxT1ely+B+xFAmJKVSTbpM/CuL7qxO8w=="  crossorigin="anonymous"  />  <link rel="stylesheet" href="/static/css/main.css" />  </head>  <body class="mb-3">  <div id="waiter" class="position-relative vertical-center">  <div class="position-absolute top-50 start-50 translate-middle">  <div class="spinner-grow" role="status">  <span class="visually-hidden">Loading...</span>  </div>  </div>  </div>  <div id="content" class="container-fluid">  <div class="row my-3">  <div class="col-12">  <div class="card">  <div class="card-body">  <h1 id="name"></h1>  <p id="text"></p>  <small id="assigned" class="text-muted"></small><br />  <small id="due" class="text-muted"></small>  <div id="files">  </div>  </div>  </div>  </div>  </div>  <div class="row">  <div id="requests" class="col-lg-2 mb-3"></div>  <div id="uploads" class="col-lg-10 mb-3"></div>  </div>  </div>  </body> </html> |

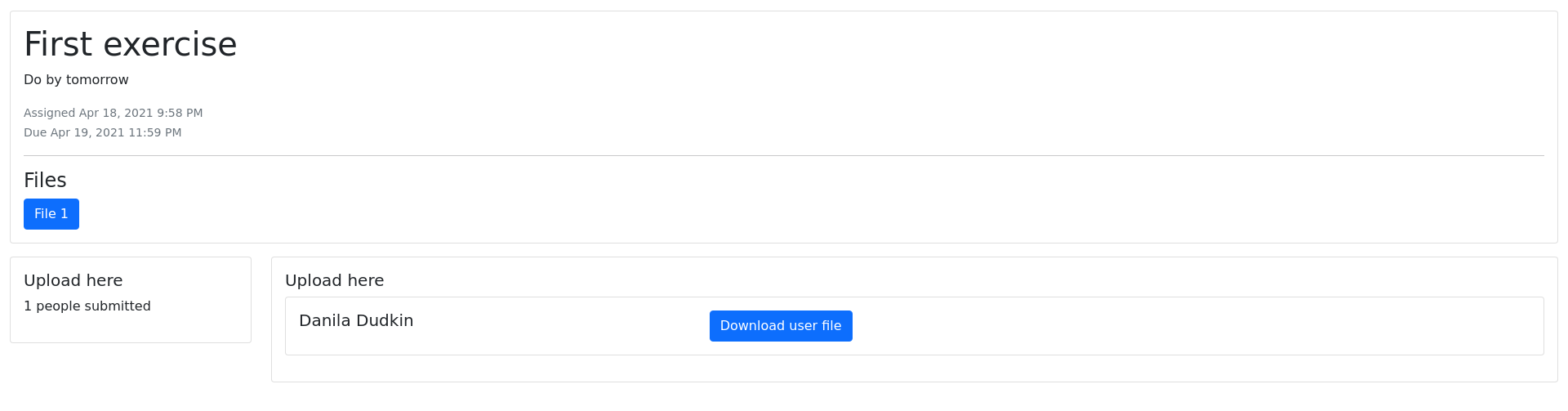
The javascript for the teacher

|  |
| --- |
| var assgn\_id = window.localStorage.getItem("\_asn"); // Get all the global variables var assignment; var selected = 0;  $(() => {  // Get the role of the person  var role = window.localStorage.getItem("\_rol");  // If the role is not a teacher  if (role != "nqrOzz0jmxA=") {  // Remove the script from the page and let the other script run instead  $("#t").remove();  return;  }   // Show a waiter  $("#waiter").show();  $("#content").hide();   $("#content").on("complete", loadWorkspace);   var params = {  id: assgn\_id,  };   // Get all the info for the assignment  axios  .get("/api/school/subject/assignment", {  params: params,  })  .then((res) => {  assignment = res.data.data;  try {  $("#content").trigger("complete");  } catch (e) {  console.error(e);  }  })  .catch((err) => {  console.error(err);  }); });  function loadWorkspace() {  // Set the name of the assignment and the text  $("#name").text(assignment.name);  $("#text").text(assignment.text);   // Parse when it was assigned and when its due  const time\_assigned = moment(assignment.time\_assigned).format("lll");  const time\_due = moment(assignment.time\_due).format("lll");   // Add the text to the page  $("#assigned").text("Assigned " + time\_assigned);  $("#due").text("Due " + time\_due);   // If the assignment has any files attached show that they can be downloaded  if (assignment.files) {  $("#files").append("<hr/><h4>Files</h4>")  assignment.files.forEach((file, index) => {  // Set the name of the download to  // assignment\_name\_filename  $("#files").append(`  <a  style="max-width: 18em"  class="btn btn-primary"  href="${file.path}"  download="${assignment.name.replace(/\s/g, "\_")}\_${file.name}"  >File ${index + 1}</a  >`)  })  }   // If there are any requests show them too  if (assignment.requests) {  var index = -1;  assignment.requests.forEach((request) => {  index++;  var subText =  request.uploads  ? request.uploads.length + " people submitted"  : "Nobody submitted";  $("#requests").append(  $(`  <div onclick="select(${index})" class="card clickable">  <div class="card-body">  <h5 class="card-title">${request.name}</h5>  <p>${subText}</p>  </div>  </div>`)  );  });  }   updateSelection();   $("#waiter").remove();  $("#content").show(); }  function updateSelection() {  // Delete everything from the screen  $("#uploads").empty();   // Get the selected request  var request = assignment.requests[selected];  if (request.uploads) {  // For each upload get the file to download  request.uploads.forEach((upload) => {  var downloadFile = `${  upload.user.first\_name + "\_" + upload.user.last\_name + "\_" + upload.name  }`;   // template for each upload  $("#uploads").append(  $(`<div class="card">  <div class="card-body">  <h5 id="req-name" class="card-title"></h5>  <div class="card mb-3">  <div class="card-body">  <div class="row g-0">  <div class="col-lg-4">  <h5 class="card-title">${  upload.user.first\_name + " " + upload.user.last\_name  }</h5>  </div>  <div class="col-lg-8">  <a class="btn btn-primary" href="${  upload.path  }" download="${downloadFile}">Download user file</a>  </div>  </div>  </div>  </div>  </div>  </div>`)  );  });  } else {  // If nobody uploaded say "no submissions"  $("#uploads").append(  $(`<div class="card">  <div class="card-body">  <h5 id="req-name" class="card-title"></h5>  <div class="card mb-3">  <div class="card-body">  <h2>No submissions</h2>  </div>  </div>  </div>  </div>`)  );  }  $("#req-name").text(request.name); }  // Select each request function select(id) {  selected = id;  updateSelection(); } |

The javascript for the student

|  |
| --- |
| var assgn\_id = window.localStorage.getItem("\_asn"); var assignment; var selected = 0;  $(() => {  var role = window.localStorage.getItem("\_rol");  // If not a student is viewing the page then let the teacher script run  if (role != "cNFDBScBBJ8=") {  $("#st").remove();  return;  }   // Show a waiter  $("#waiter").show();  $("#content").hide();   $("#content").on("complete", loadWorkspaceSt);   var params = {  id: assgn\_id,  };   // Get the information for the assignment  axios  .get("/api/school/subject/assignment", {  params: params,  })  .then((res) => {  assignment = res.data.data;  try {  $("#content").trigger("complete");  } catch (e) {  console.error(e);  }  })  .catch((err) => {  console.error(err);  }); });  function loadWorkspaceSt() {  // Show the name and text of assignment  $("#name").text(assignment.name);  $("#text").text(assignment.text);   // Get when it was assigned and due  const time\_assigned = moment(assignment.time\_assigned).format("lll");  const time\_due = moment(assignment.time\_due).format("lll");   // Add the text to the screen  $("#assigned").text("Assigned " + time\_assigned);  $("#due").text("Due " + time\_due);   // For each file added to the assignment  // Add a download button  if (assignment.files) {  $("#files").append("<hr/><h4>Files</h4>");  assignment.files.forEach((file, index) => {  $("#files").append(`  <a  style="max-width: 18em"  class="btn btn-primary"  href="${file.path}"  download="${assignment.name.replace(/\s/g, "\_")}\_${file.name}"  >File ${index + 1}</a  >`);  });  }   // If there are any upload requests  if (assignment.requests) {  // For each request show a button  assignment.requests.forEach((request) => {  if (request.complete) return;  $("#uploads").append(  $(`  <div id="${request.id}" class="card mb-3">  <div class="card-body">  <div class="row">  <div class="col-lg-3">  <h5 class="card-title">${request.name}</h5>  </div>  <div class="col-lg-9">  <input onchange="completeReq('${request.id}', this)" id="file" class="form-control" type="file" id="uploadFile"/>  </div>  </div>  </div>  </div>`)  );  });  }   $("#waiter").remove();  $("#content").show(); }  function completeReq(id, el) {  // Send the uploaded file to the server  var formData = new FormData();  var files = $(el).prop("files");  $.each(files, (\_, file) => {  formData.append("files", file);  });   // Make the POST request  axios  .post("/files", formData, {  headers: {  "Content-Type": "multipart/form-data",  },  })  .then((res) => {  markComplete(id, res.data.files[0]);  })  .catch((err) => {  console.error(err);  $("#errors\_asn").val(err.response.data.error);  }); }  function markComplete(id, file) {  // Mark assignment as complete  data = {  request\_id: id,  filepath: file,  };   // Create a POST request to complete an assignment  axios.post("/api/school/subject/assignment/complete", data)  .then((res) => {  if (res.data.data) {  // If uploaded successfully then delete the upload request  $("#" + id).remove()  }  })  .catch((err) => {  alert(err.response.data.error);  }); } |

The page looks like this for the teacher



The page looks like this for the student

