
T3A2 School Yearbook App

Wenxuan Pan, Dieter Schmid, Alicia Han

Website Description

A Digital School Yearbook

Makes a permanent record of all students in attendance for a year level that has graduated a school.

- Displays Image of the Student
- Contact Information on the student
- Final Thoughts

Features/Functionality

- The admin can add students to the app
- The students can update their details
- The admin can upload photos and display them for students
- The students can fill out a questionnaire regarding their final thoughts
- The student can provide contact details
- Admin can invite students to the app via a unique code
- The app will allow admin and student logins



Target Audience

The Application is targeted at all school stakeholders this includes :

- Students of the graduating year level
- Other students at the school
- Alumni members
- Parents of the students
- Teachers and other staff



Tech Stack

This application uses the MERN technology stack:

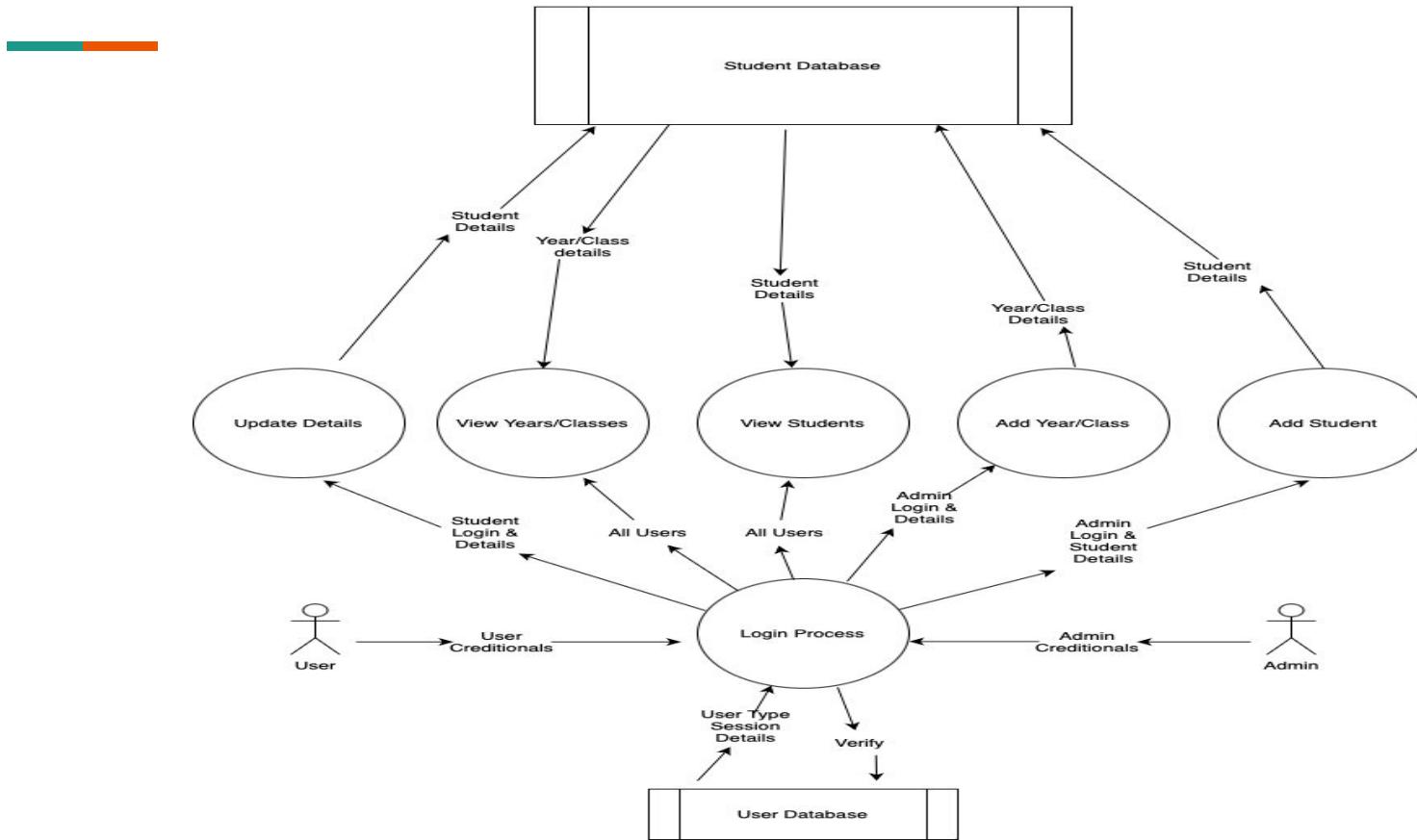
- MongoDB
- Express.js server
- Node.js
- React front end.

Deployment:

- Front end: Vercel
- Back end: Render

Dataflow Diagram

Dataflow Diagram



User Stories



User Stories

Split into three categories:

1. All users (students and admin)
2. Admin only
3. Students only

Throughout the planning process, we found that we needed to add and modify ideas in order for the application to work better. Hence, user stories from the beginning of the planning phase were updated to reflect these changes.

Initial User Stories

All users:

1. As a user, I would like to be able to add a student profile into different classes/cohorts and year levels, so that student data is organised into the correct categories.
2. As a user, I would like to be able to login and view a list of students in each class and year level/cohort at my school.
3. As a user, I would like to be able to click into individual students and see their relevant information, so that I can stay in touch with classmates and connect with acquaintances.
4. As a user, I would like to have a navigation bar so that I can easily access different parts of the website.

Initial User Stories

Admin only:

1. As an admin, I want to be able to remove any profile from the database to ensure data integrity.

Students only:

1. As a student, I would like to be able to update my information, so that others can get in touch with me via my current contact details if needed, however I would like to ensure that only admin and myself can manipulate my account, for better security.
2. As a student, I want to be able to delete any information on my account, or my entire profile, from the database so that I have control over my information.

Revised User Stories with Updated Features

All users:

1. As a user, I would like to be able to login and view a list of classes in my school, and the students in each class.
2. As a user, I would like to be able to click into individual students and see their relevant information, so that I can stay in touch with my classmates and connect with acquaintances.
3. As a user, I would like to have a navigation bar so that I can easily navigate around the application.

Revised User Stories with Updated Features

Admin only:

1. As an admin, I would like to see a list of all yearbooks and classes at my school, and be able to modify or delete them as appropriate, in case of error or duplication.
2. As an admin, I would like to be able to input a class of students along with their email and photos, and modify and delete student data as needed in case of error or duplication.
3. As an admin, I would like to be able to have a unique code to give to students, so they can utilise it to sign up and create their own accounts linked to the student database, this will ensure that only students who are actually in the class will be able to register to that class.



Revised User Stories with Updated Features

Students only:

1. As a student, I would like to have a unique code from my school admin, to be able to register as an account that is linked to a yearbook in the database.
2. As a student, I would like to be able to add, modify and delete any of my data from my account, and ensure that only the admin or myself can manipulate my account, for better security.

Application Architecture Diagram



User

Admin/student interacts with browser on device

Front-end / Client-side

React

- JavaScript framework
- Makes HTTP request to Express.js
- Receives HTTP response from Express.js
- React Router defines routes
- Renders components on a responsive web page for user, based on HTTP response



Back-end / Server-side

Node.js

- Back-end JavaScript runtime environment

Express.js

- Receives HTTP requests from React and sends HTTP responses back
- Defines API endpoints
- Controller functions
- Authentication and authorisation
- Sends instructions to Mongoose

Mongoose

- Defines schema and models
- Makes queries to the database with CRUD operations
- Sends results back to Express.js

Database



MongoDB

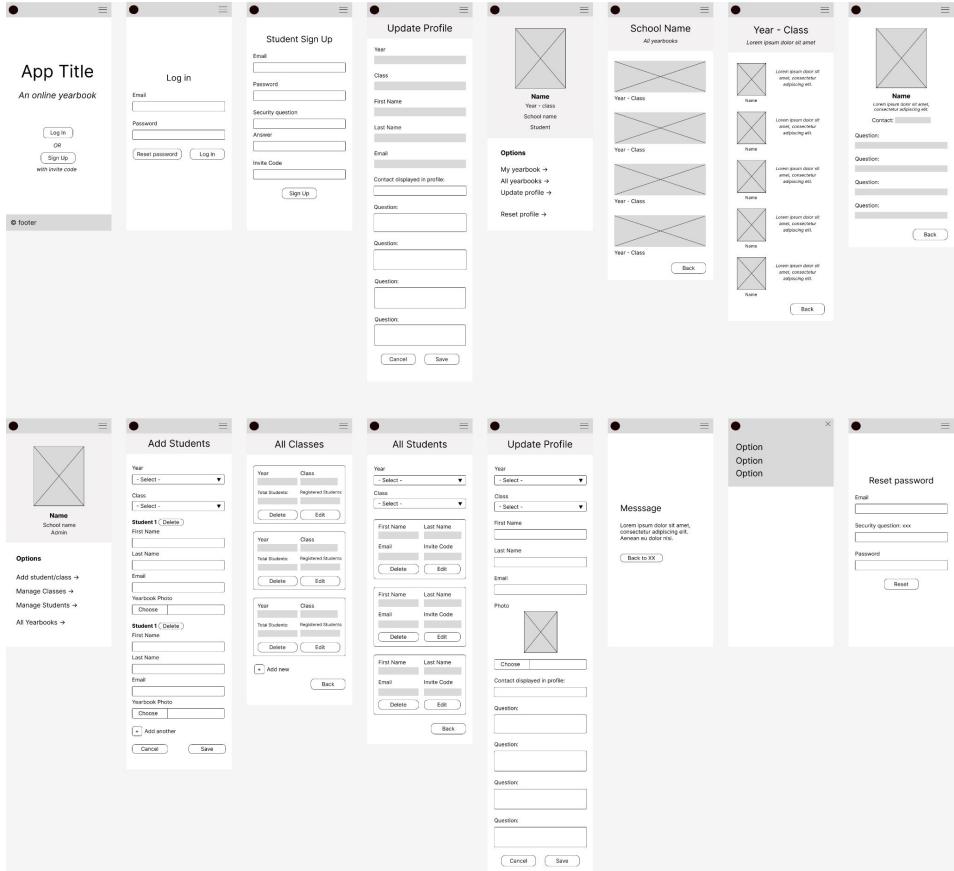
- Hosted on Atlas
- Stores data in collections and documents
- Receives CRUD queries from Mongoose
- Results are interpreted by Mongoose

Wireframes

Mobile

[View in Figma](#)

*Note (2 Sep): Reset password not implemented in production stage due to time constraints



Tablet

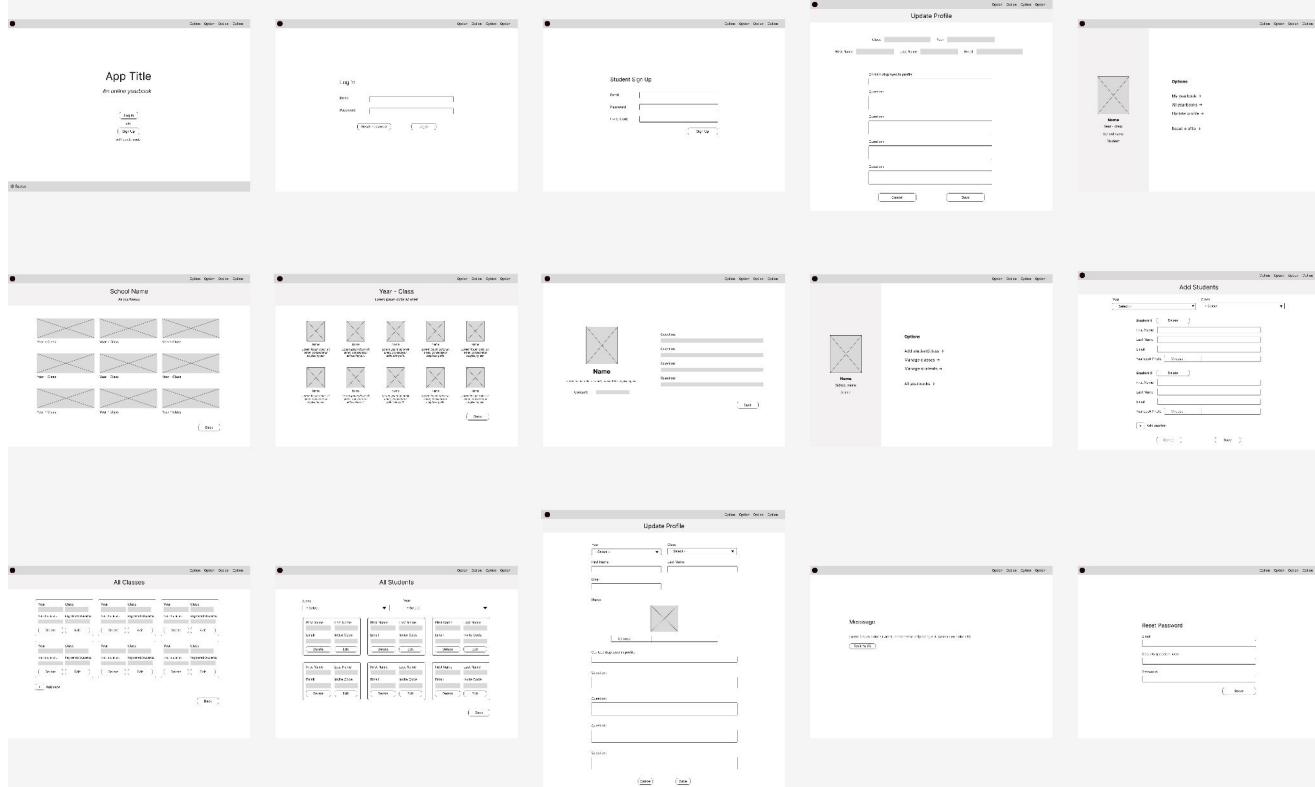
[View in Figma](#)

The wireframe displays a variety of screens for a tablet application:

- App Title:** An online yearbook.
- Log in:** Fields for Email and Password, with options to Reset Password or Log In.
- Student Sign Up:** Fields for First Name, Last Name, Email, Password, Security question, Answer, and Birth Date, with a "Sign Up" button.
- Update Profile:** A large form for updating profile information, including Name, Year, Photo, and several security questions.
- Options:** A menu with links to My yearbook, All yearbooks, Update profile, and Reset profile.
- School Name:** A list of school names with checkboxes.
- Year - Class:** A list of year and class combinations with checkboxes.
- Name:** A form for entering a name, with a placeholder "Leave empty if you don't want to add a student".
- Add Students:** A form for adding students, requiring Year, School name, and Class, and providing fields for Student 1 and Student 2.
- All Classes:** A list of classes with checkboxes.
- All Students:** A list of students with checkboxes.
- Update Profile:** A form for updating student profiles, including Name, Class, Photo, and security questions.
- Message:** A screen for sending messages, with a placeholder "Leave empty if you don't want to add a message".
- Reset password:** A form for resetting a password, with fields for Email, Security question, and Password.

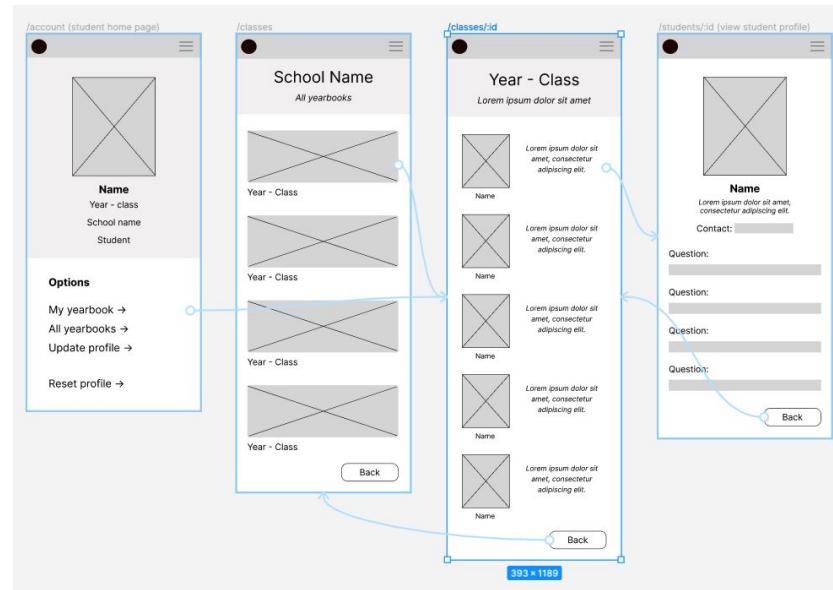
Desktop

[View in Figma](#)



Relationships between screens

- Student flow - Figma
- Admin flow - Figma



Code Examples: Front End

App component



- React router is used to enable client side routing
- Context API is used to allow components lower down the tree to easily access shared state without having to pass props through multiple levels
- Reducer pattern is applied to manage complex state (i.e. school - classes, years, students) logic in a maintainable and predictable way

```
return (
  <>
    /* passed in user and school as global states */
    <UserContext.Provider value={{ user, setUser, loadEmptyUser, loaded }}>
      <SchoolContext.Provider value={{ school, dispatch }}>
        <NavbarLine />
        <Routes>
          {/* landing, log in and sign up pages; they share the same background image, thus grouped together */}
          <Route path="/" element={<BackgroundImage />}>
            <Route index element={<RedirectRoute page={Landing} />} />
            <Route path="login" element={<RedirectRoute page={Login} />} />
            <Route path="signup" element={<RedirectRoute page={SignUp} />} />
          </Route>

          {/* yearbooks pages */}
          <Route path="/classes">
            <Route index element={<LoggedInRoute page={Classes} />} />
            <Route
              path=":id"
              element={<LoggedInRoute page={YearbookWrapper} />}
            />
          </Route>

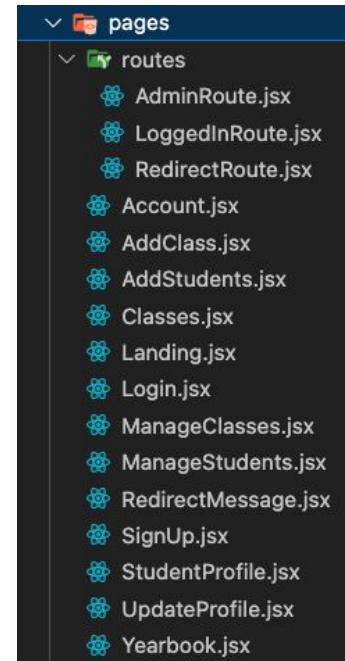
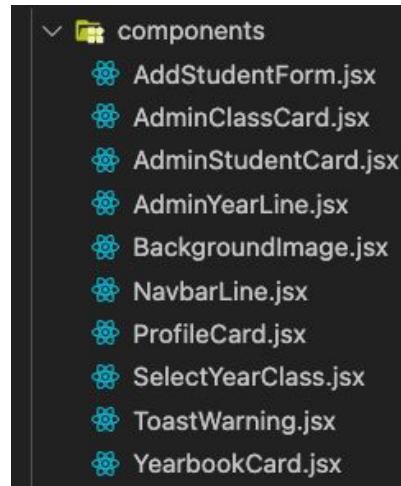
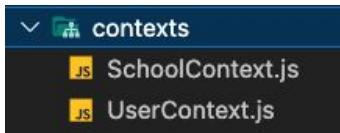
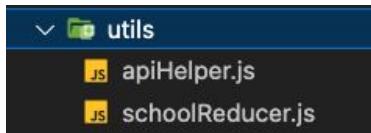
          {/* student pages */}
          <Route path="/students">
            <Route
              path=":id"
              element={<LoggedInRoute page={StudentProfileWrapper} />}
            />
            {/* update profile page for both student and admin, with different props */}
            <Route
              path=":id/edit"
              element={<LoggedInRoute page={UpdateProfileWrapper} />}
            />
          </Route>
        </Routes>
      </SchoolContext.Provider>
    </UserContext.Provider>
  </>
)
```

```
  /* student pages */
  <Route path="/students">
    <Route
      path=":id"
      element={<LoggedInRoute page={StudentProfileWrapper} />}
    />
    {/* update profile page for both student and admin, with different props */}
    <Route
      path=":id/edit"
      element={<LoggedInRoute page={UpdateProfileWrapper} />}
    />
  </Route>

  /* account pages */
  <Route path="/account">
    <Route index element={<LoggedInRoute page={Account} />} />
    <Route
      path="classes"
      element={<AdminRoute page={ManageClasses} />}
    />
    <Route
      path="classes/new"
      element={<AdminRoute page={AddClass} />}
    />
    <Route
      path="students"
      element={<AdminRoute page={ManageStudents} />}
    />
    <Route
      path="students/new"
      element={<AdminRoute page={AddStudents} />}
    />
  </Route>
  <Route
    path="/user-error"
    element={<RedirectMessage type="userError" />}
  />
<Route path="*" element={<RedirectMessage type="notFound" />} />
</Routes>
</SchoolContext.Provider>
</UserContext.Provider>
</>
```

Components/modules

Code is divided into different components and utilities that are responsible for one single purpose or a specific task, to meet the DRY principle and single responsibility principle, and to ensure maintainability, reusability and testability



Conditional Rendering

- isAdmin and isLoggedIn statuses are used in route wrappers to protect and redirect confidential pages
- isAdmin variable is used to conditionally render UI elements for students and admins

```
const AdminRoute = ({ page: Page }) => {
  const { user, loaded } = useContext(UserContext)
  if (!loaded) {
    return <p>Loading...</p>
  }

  if (!user.isAdmin) {
    if (!user.isLoggedIn) {
      return <Navigate to="/" replace />
    }
    return <Navigate to="/account" replace />
  }
  return <Page />
}
```

```
// set navbar text and direct links for different views
const views = [
  default: [
    { text: "Log In", navCtrl: "/login" },
    { text: "Sign Up", navCtrl: "/signup" }
  ],
  student: isLoggedIn &&
    isAdmin && [
      { text: `${user.name} - Student`, navCtrl: "/account" },
      { text: "My yearbook", navCtrl: `/classes/${studentClass._id}` },
      { text: "All yearbooks", navCtrl: "/classes" },
      { text: "Update profile", navCtrl: `/students/${student._id}/edit` }
    ],
  admin: isLoggedIn &&
    isAdmin && [
      { text: `${user.name} - Admin`, navCtrl: "/account" },
      { text: "Students", navCtrl: "/account/students" },
      { text: "Classes", navCtrl: "/account/classes" },
      { text: "Yearbooks", navCtrl: "/classes" }
    ]
]

// find out current view based on logged in and admin statuses
const currentView = isLoggedIn
  ? isAdmin
    ? views.admin
    : views.student
  : views.default
// nav options based on current view
const viewEl = currentView.map((option) => (
  <Nav.Link as={Link} to={option.navCtrl} key={option.text}>
    {option.text}
  </Nav.Link>
))
```

Error Handling

- Try-catch blocks are used to handle possible errors during API calls
- React-toastify library is used to display error message on the client's end

```
async function handleSubmit(e) {
  e.preventDefault()
  try {
    const { token, user: registeredUser } = await apiPost("/signup", content)
    const { __v, role, ...filteredUser } = registeredUser
    const newUser = {
      token,
      isLoggedIn: true,
      isAdmin: role === "admin" ? true : false,
      ...filteredUser
    }
    setUser(newUser)
    localStorage.setItem("user", JSON.stringify(newUser))
    nav("/account")
  } catch (e) {
    console.error(e)
    toast.warn("Sign up failed. Please check your credentials and try again")
  }
}
```

Example component - Add Class (admin)

- React-bootstrap is used for styling UI elements
- useState is used to handle the input state specific to this component
- handleSubmit and handleInputChange functions are used to encapsulate the workflows of form actions
- Dispatch method from the SchoolContext is used to update the global state with the new class (and year)
- Error message will be displayed when API call fails

```
const AddClass = () => {
  const nav = useNavigate()
  const [content, setContent] = useState({ name: "", year: "" })
  const { school, dispatch } = useContext(SchoolContext)
  const { user } = useContext(UserContext)

  function handleInputChange(changed) {
    setContent((prev) => ({ ...prev, ...changed }))
  }

  async function handleSubmit(e) {
    e.preventDefault()

    try {
      // check if year exists
      const yearExists = school.years.find((y) => y.name === content.year)
      if (!yearExists) {
        // if not exists, create new year
        const newYear = await apiPost(
          "/years",
          { name: content.year },
          user.token
        )
        // add new year to school state
        dispatch({ type: "add_year", year: newYear })
      }
      const createdClass = await apiPost("/classes", content, user.token)
      dispatch({ type: "add_class", class: createdClass })

      nav("/account/classes")
    } catch (e) {
      // if POST request failed, show toast message
      console.error(e)
      toast.warn("Class creation failed. Please check input and try again!")
    }
  }
}
```

```
return (
  <Container className="mt-4">
    <Row>
      <h1 className="fs-2">Add Class</h1>
    </Row>
    <Form onSubmit={(e) => handleSubmit(e)}>
      <Row xs={1} md={2}>
        <Form.Group controlId="formYear" as={Col}>
          <Form.Label>Year</Form.Label>
          <Form.Control
            value={content.year}
            onChange={(e) => handleInputChange({ year: e.target.value })}
            type="number"
            min="1960"
            max="2099"
          />
        </Form.Group>
        <Form.Group controlId="formClass" as={Col}>
          <Form.Label>Class</Form.Label>
          <Form.Control
            value={content.name}
            onChange={(e) => handleInputChange({ name: e.target.value })}
            type="text"
            maxLength="100"
          />
        </Form.Group>
      </Row>
      <Row className="text-center mt-4">
        <Col>
          <Button as={Link} to="/account">
            Cancel
          </Button>
        </Col>
        <Col>
          <Button type="submit">Save</Button>
        </Col>
      </Row>
    </Form>
    <ToastWarning />
  </Container>
)
```

Test: Log in component

- Vitest and react testing library are used for automated testing
- userEvent is used to simulate actual browser interaction
- `screen.getByRole` is used as the preferred method to query elements in the accessibility tree
- Mocked data is used to pass into the Context Provider

```
describe("Login component", () => {
  let mockedSetUser = vi.fn()
  beforeEach(() => {
    render(
      <BrowserRouter>
        <UserContext.Provider
          value={{
            | setUser: mockedSetUser
          }}
        >
          <Login />
        </UserContext.Provider>
      </BrowserRouter>
    )
  })

  it("renders without crashing", () => {
    expect(screen.getByRole("heading", { level: 1 })).toBeInTheDocument()
    expect(screen.getByRole("heading", { level: 1 })).toHaveTextContent(
      "Log In"
    )
  })

  it("accepts user input and changes value", async () => {
    expect(screen.getByLabelText("Email address")).toBeInTheDocument()
    expect(screen.getByLabelText("Password")).toBeInTheDocument()

    await userEvent.type(
      screen.getByLabelText("Email address"),
      "test@example.com"
    )
    await userEvent.type(screen.getByLabelText("Password"), "password")

    expect(screen.getByLabelText("Email address")).toHaveValue(
      "test@example.com"
    )
    expect(screen.getByLabelText("Password")).toHaveValue("password")
  })
})
```

Code Examples: Back End

Authentication

- Middleware for route protection
- Checks if the incoming request has an appropriate JWT token
- Error handling for non-existent user, student or user that isn't linked to a student in the database

```
4 // Authentication middleware
5 const authenticateToken = async (req, res, next) => {
6   const authHeader = req.headers['authorization']
7   const token = authHeader && authHeader.split(' ')[1]
8
9   if (!token) {
10     return res.status(401).json({ error: 'Authorization token missing.' })
11   }
12
13   try {
14     const decodedToken = jwt.verify(token, process.env.JWT_SECRET)
15     const user = await UserModel.findById(decodedToken.id);
16
17     if (!user) {
18       return res.status(403).json({ error: 'Invalid user.' })
19     }
20
21     // Skip the student linkage check for admin users
22     if (user.role === 'admin') {
23       req.user = user
24       next()
25     } else {
26       // Check if the user is linked to a student
27       if (!user.student) {
28         return res.status(403).json({ error: 'User is not linked to a student.' })
29       }
30
31       // Verify if the linked student exists
32       const student = await StudentModel.findById(user.student)
33       if (!student) {
34         return res.status(403).json({ error: 'Linked student not found.' })
35       }
36
37       req.user = user
38       req.student = student
39       next()
40     }
41   } catch (err) {
42     return res.status(403).json({ error: 'Invalid token.' })
43   }
44 }
```

Authorization

- Middleware for route protection
- One for only admin access
- One for admin OR linked student access (e.g. updating student details)

```
46 // Authorization middleware for admin
47 const authorizeAdmin = (req, res, next) => {
48   if (req.user && req.user.role === 'admin') {
49     next()
50   } else {
51     return res.status(403).json({ error: 'Unauthorized.' })
52   }
53 }
54
55 // Authorization middleware for admin or linked student
56 const authorizeAdminOrLinkedStudent = (req, res, next) => {
57   if (req.user.role === 'admin' || req.user.student.equals(req.student._id)) {
58     next()
59   } else {
60     return res.status(403).json({ error: 'Unauthorized.' })
61   }
62 }
```

Route: Register

- POST '/signup'
- Student will have unique code (the student ID that is created when the admin adds a student to the database) that will link the new account with an existing student profile
- The registered email must also match the email in the student profile (that was input by the admin)
- Password is encrypted
- JWT token is generated

```
8 // Register an account as a student
9 router.post('/signup', async (req, res) => {
10   try{
11     const { name, email, role, studentId } = req.body // studentId has been emailed to user as registration code
12
13     // Password encryption
14     let password = req.body.password
15     const salt = await bcrypt.genSalt(10)
16     password = await bcrypt.hash(password, salt)
17
18     // Find the student by ID
19     const student = await StudentModel.findById(studentId)
20
21     if (!student) {
22       return res.status(400).json({ error: 'Invalid registration code.' })
23     }
24
25     // Check if the provided email matches the student's email
26     if (email !== student.email) {
27       return res.status(400).json({ error: 'Email does not match.' })
28     }
29
30     const newUser = await UserModel.create({ name, email, role, password, student: student._id })
31
32     // return user without password
33     const returnedUser = await UserModel.findById(newUser._id)
34     // Generate a JWT token for the new user
35     const token = jwt.sign({ id: newUser._id }, process.env.JWT_SECRET, {
36       expiresIn: process.env.JWT_EXPIRE,
37     })
38
39     res.status(201).json({ token, message: `Registration successful, welcome ${name}!`, user: returnedUser })
40   } catch (err) {
41     res.status(500).send({ error: err.message })
42   }
43 }
44 }
```

Route: Login

- POST '/login'
- Checks that both email and password have been entered
- JWT token is returned if email and password are both correct

```
46 // Login route
47 router.post('/login', async (req, res) => {
48   const { email, password } = req.body
49
50   // Checking that both email and password are entered
51   if (!email || !password) {
52     return res.status(400).json({ error: 'Email and password are required.' })
53   }
54
55   try {
56     const user = await UserModel.findOne({ email }).select('+password')
57
58     // Check the password if user is found
59     if (user) {
60       const isPasswordMatch = bcrypt.compare(password, user.password)
61       // Return jwt token if password is correct
62       if (isPasswordMatch) {
63         const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, {
64           expiresIn: process.env.JWT_EXPIRE
65         })
66         const returnedUser = await UserModel.findOne({ email })
67         res.status(200).json({ token, message: 'Login successful!', user: returnedUser })
68       } else {
69         res.status(400).send('Invalid credentials.')
70       }
71     } else {
72       res.status(404).send({ error: 'User not found.' })
73     }
74   }
75   catch (err) {
76     res.status(500).send({ error: err.message })
77   }
78 })
```

Route: Creating a student

- POST '/students'
- Route can only be accessed with valid admin token
- Class input needs to be a valid (to be chosen from a drop down menu on the front end)

```
40 // Create a student
41 // Admin access only
42 router.post('/', authenticateToken, authorizeAdmin, async (req, res) => {
43   try {
44     const { firstName, lastName, email, class: classId, photo } = req.body
45
46     // Check if class exists
47     const selectedClass = await ClassModel.findById(classId)
48     if (!selectedClass) {
49       return res.status(404).json({ error: 'Class not found' })
50     }
51
52     const newStudent = await StudentModel.create({
53       firstName,
54       lastName,
55       email,
56       class: selectedClass._id,
57       photo,
58       questionOne: '',
59       questionTwo: '',
60       questionThree: '',
61       questionFour: '',
62       contactDetails: '',
63       quote: ''
64     })
65
66     res.status(201).send(newStudent)
67
68   } catch (error) {
69     res.status(500).send({ error: error.message })
70   }
71 })
```

Test: Get Students

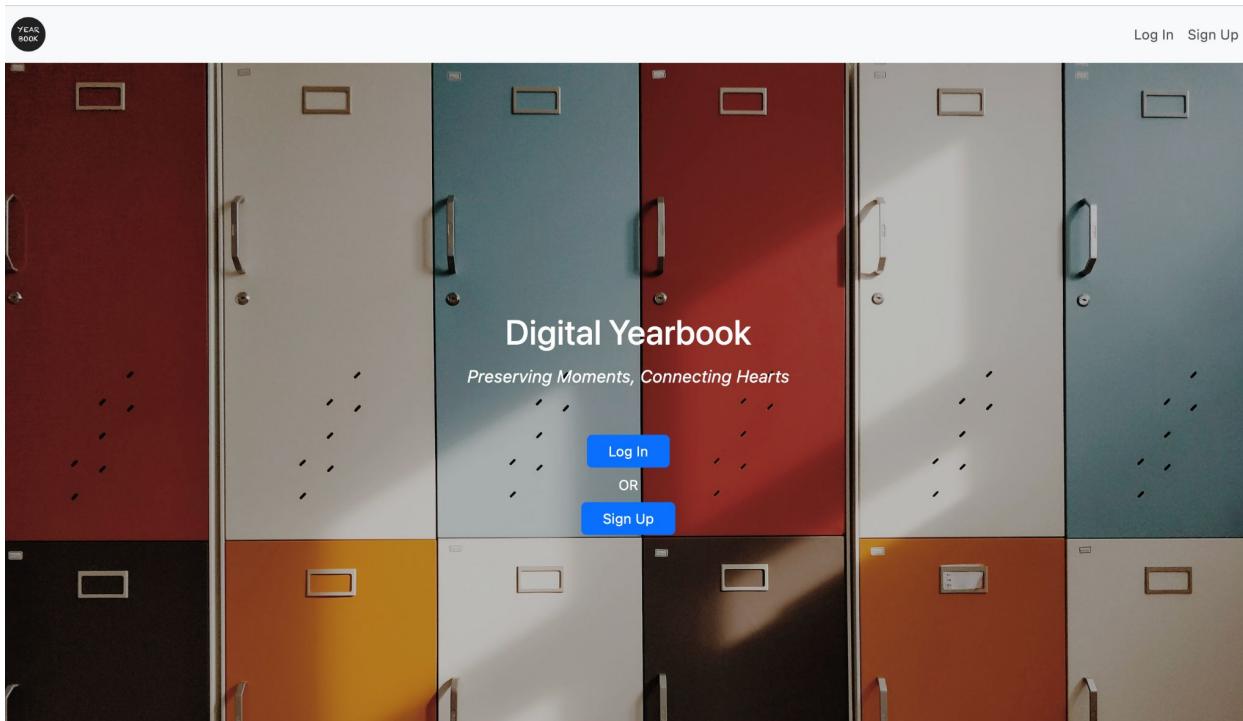
- Sends bearer token with the test request
- Checks that status code is 200 and that JSON is returned
- Checks that an ID of 24 characters exists for each student
- Checks that each student has necessary fields returned

```
6 // Testing get all students
7 describe ('GET /students', () => {
8   let res
9
10  beforeEach(async () => {
11    res = await request(app).get('/students').set('Authorization', `bearer ${token}`)
12  })
13
14  test ('Returns JSON', async () => {
15    expect(res.status).toBe(200)
16    expect(res.header['content-type']).toMatch('json')
17  })
18
19  test('Each student has an "_id" of 24 characters', () => {
20    res.body.forEach(el => {
21      expect(el._id).toBeDefined()
22      expect(el._id).toHaveLength(24)
23    })
24  })
25
26  test('Each student has a first name, last name, class and email', () => {
27    res.body.forEach(el => {
28      expect(el.firstName).toBeDefined()
29      expect(el.lastName).toBeDefined()
30      expect(el.class).toBeDefined()
31      expect(el.email).toBeDefined()
32    })
33  })
34})
```

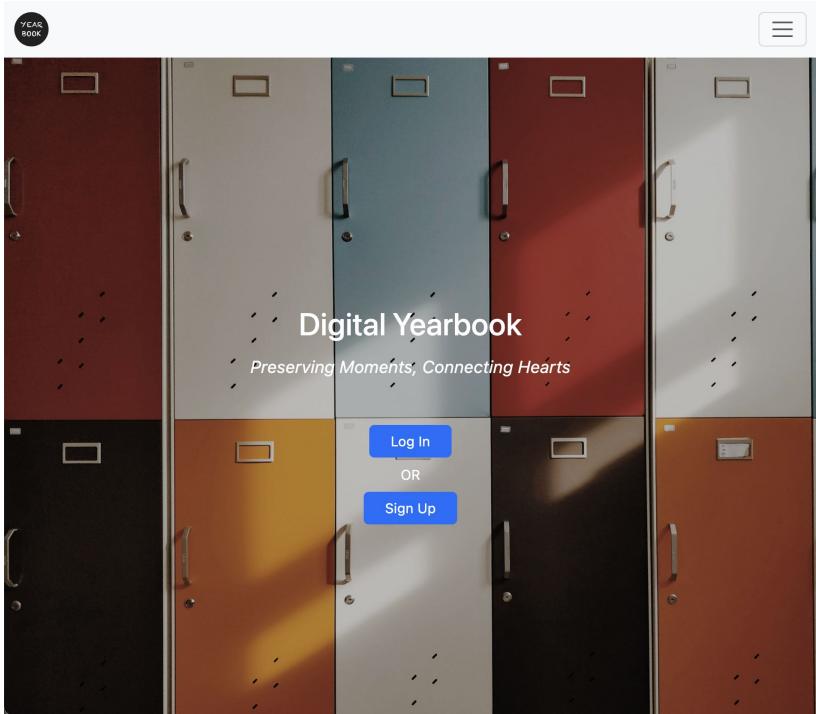
Deployed App

<https://student-yearbook.vercel.app/>

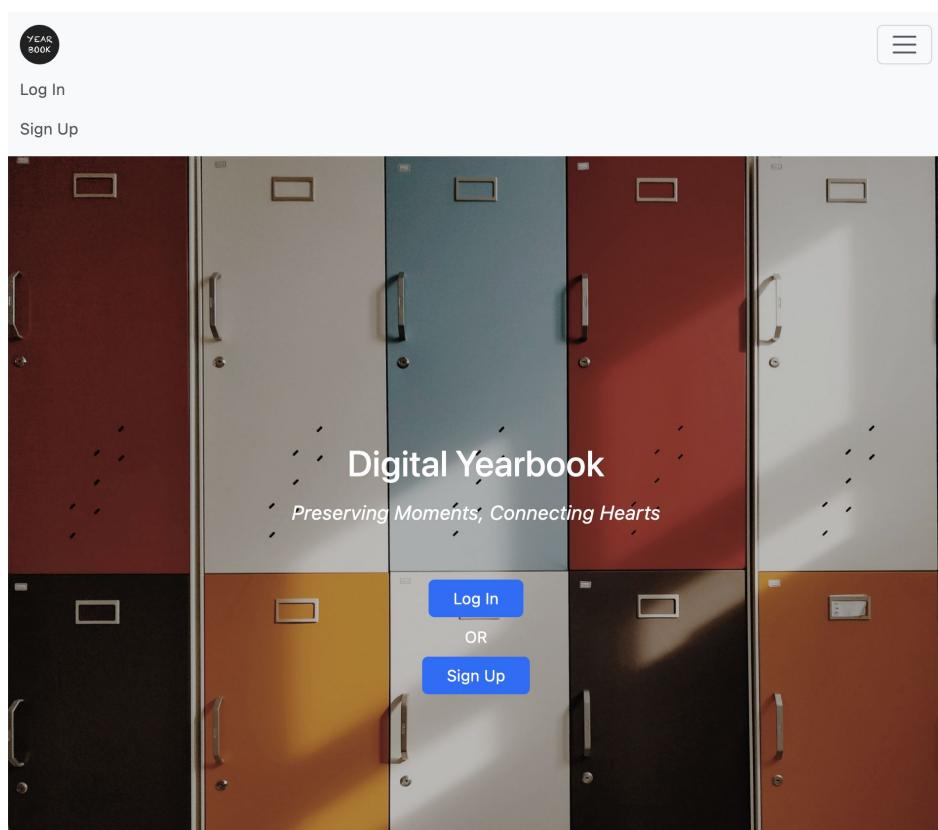
Home Page - Desktop



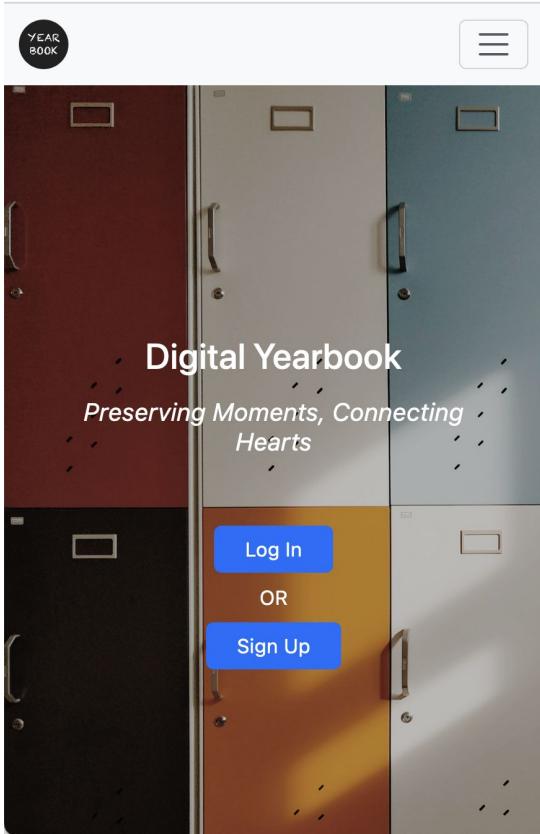
Home Page - Tablet



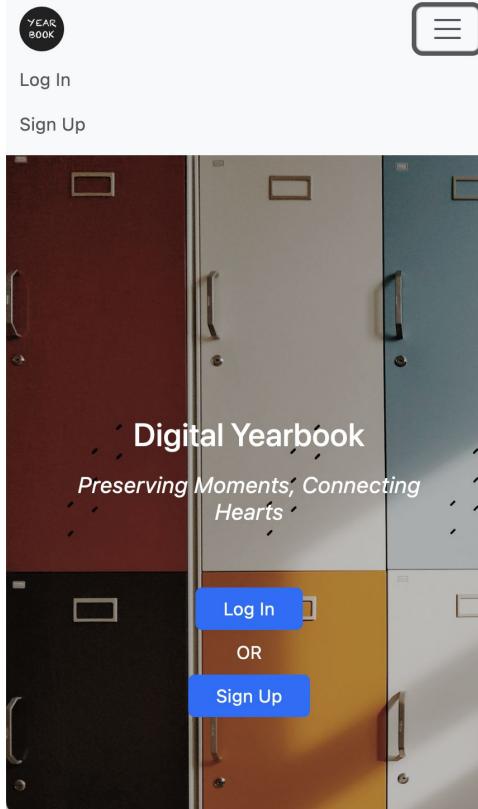
Expanded Navigation Bar:



Home Page - Mobile

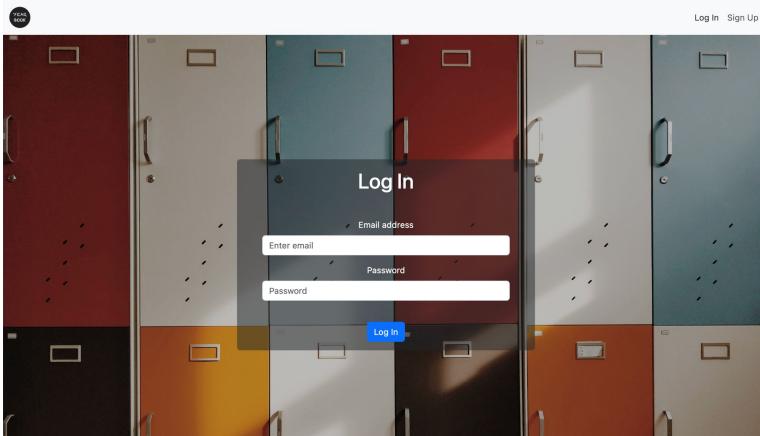


Expanded Navigation Bar:

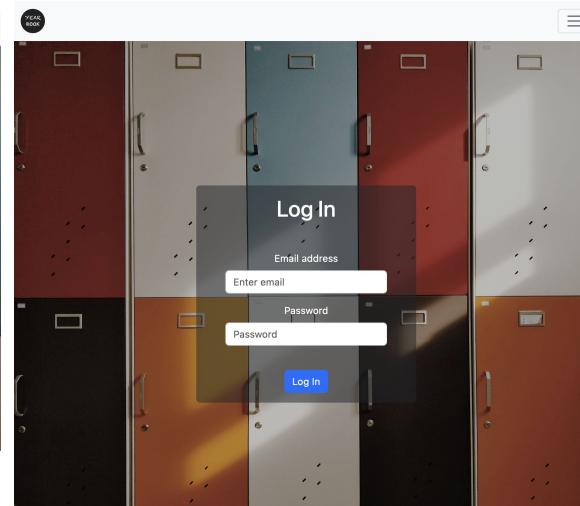


/login

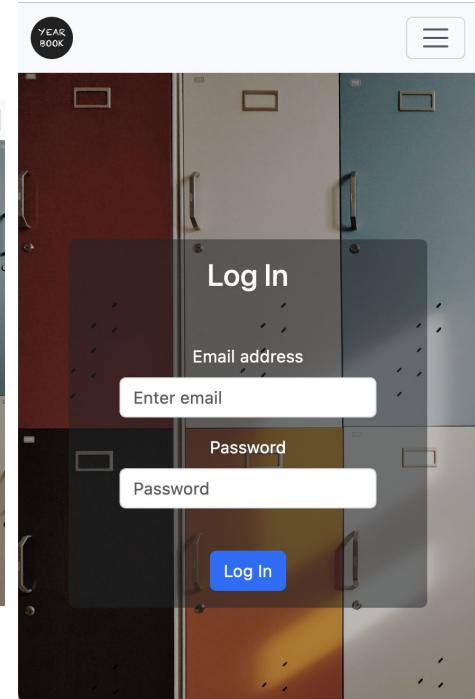
Desktop



Tablet

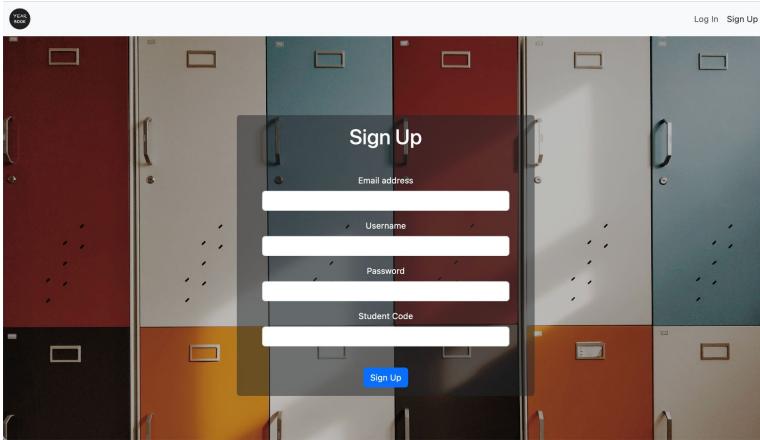


Mobile

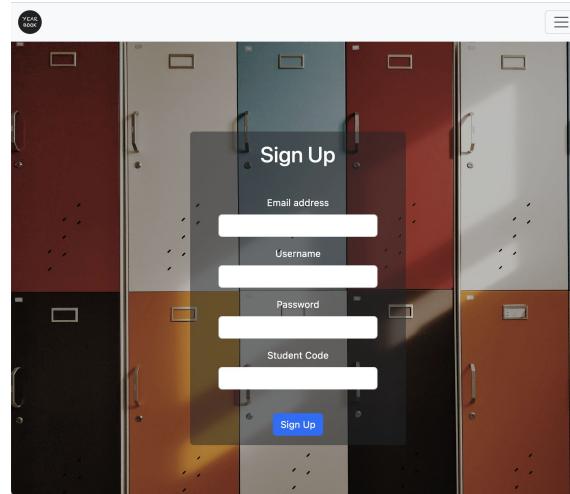


/signup

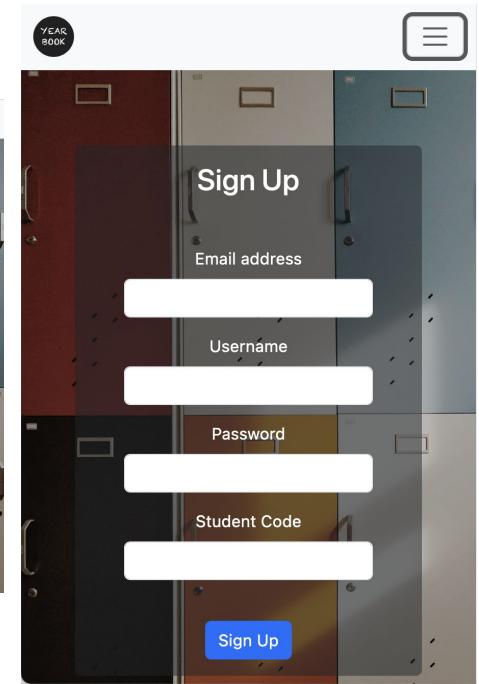
Desktop



Tablet



Mobile



Student can sign up using student code provided by the admin

/account (admin)

Desktop

The screenshot shows a desktop browser window. At the top, there's a circular icon with a person icon and the word "Yearbook". The main content area has a large gray box on the left labeled "Admin". Below it, the user information "John Admin" is displayed. On the right, there's a sidebar titled "Options" with links: "Add new students →", "Add new class →", "Manage all classes →", "Manage all students →", "All yearbooks →", and "Log Out →". At the very top of the page, there's a navigation bar with the text "John - Admin Students Classes Yearbooks".

Tablet

The screenshot shows a tablet device displaying the same account (admin) page. The layout is similar to the desktop version, with the "Admin" section on the left and the "Options" sidebar on the right. The top navigation bar is visible at the top of the screen.

Mobile

The screenshot shows a mobile device displaying the account (admin) page. The "Admin" section is centered on the screen. To the right, there's a vertical sidebar with the user information "John Admin" at the top, followed by the "Options" section with the same list of links as the desktop and tablet versions. At the top of the screen, there's a circular icon with "YEARBOOK" and a three-line menu icon.

/account (student)

Desktop



YEARBOOK

Bill - Student My yearbook All yearbooks Update profile

Options

[My yearbook \(Gecko\) →](#)

[All yearbooks →](#)

[Update Profile →](#)

[Log Out →](#)

Bill
Student
Gecko - 2009

[Reset profile](#)

Tablet



YEARBOOK

Bill - Student My yearbook All yearbooks Update profile

Options

[My yearbook \(Gecko\) →](#)

[All yearbooks →](#)

[Update Profile →](#)

[Log Out →](#)

Bill
Student
Gecko - 2009

[Reset profile](#)

Mobile



YEARBOOK

Bill

Student

Gecko - 2009

Options

[My yearbook \(Gecko\) →](#)

[All yearbooks →](#)

[Update Profile →](#)

[Log Out →](#)

[Reset profile](#)

By resetting profile, all information of the linked student will be cleared except for names, email, class and photo link

/classes

Desktop

All yearbooks

John - Admin Students Classes Yearbooks

2009 Gecko

2009 Salamander

2010 Kangaroo

2010 Possum

2020 Tiger

Back

Tablet

YEAR BOOK

All yearbooks

All yearbooks

2009 Gecko

2009 Salamander

2010 Kangaroo

2010 Possum

2020 Tiger

Back

Mobile

YEAR BOOK

All yearbooks

2009 Gecko

2009 Salamander

2010 Kangaroo

2010 Possum

2020 Tiger

Back

/classes/:classId

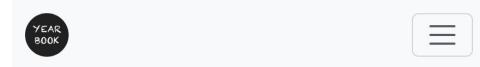
Desktop

The screenshot shows a desktop browser window with a header bar containing a logo, 'John - Admin', 'Students', 'Classes', and 'Yearbooks'. Below the header, the title 'Kangaroo 2010' is displayed. Underneath the title are two user profiles: 'john doe' (a woman with long hair) and 'Greg Carpenter' (a man with glasses). A green checkmark icon is overlaid on the profile picture of 'Greg Carpenter'. At the bottom left is a blue 'Back' button.

Tablet

The screenshot shows a tablet device displaying the same class page as the desktop version. The title 'Kangaroo 2010' is centered above two profile cards. The card for 'john doe' shows a woman's face. The card for 'Greg Carpenter' shows a man's face with a red circular overlay and a green checkmark. A blue 'Back' button is visible at the bottom left.

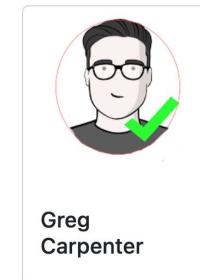
Mobile



Kangaroo
2010



john doe

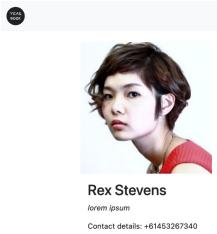


Greg
Carpenter

Back

/students/:studentId

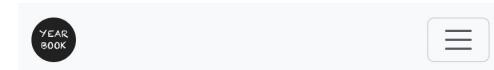
Desktop



Tablet

A tablet screenshot showing the same student profile as the desktop version. The layout is similar, with the student's photo and name at the top, followed by placeholder text and a contact link. The sidebar on the right contains the same four questions and answers as the desktop version.

Mobile



Rex Stevens

lorem ipsum

Contact details: +61453267340

Cras justo odio Dapibus ac facilisis in?

answer 1

In orci orci, convallis non arcu id?

answer 2

Integer non eros ut metus imperdiet?

answer 3

Etiam a tincidunt est, in semper odio?

answer 4

Back

/students/:studentId/edit (student)

Desktop

Bill - Student My yearbook All yearbooks Update profile

Update profile

Year Class

2009 Gecko

First Name John

Last Name Rogers

Email address john.rogers@gmail.com

Yearbook Photo <https://i.pravatar.cc/300?img=16>

Contact displayed in profile

Yearbook quote

Lorem ipsum dolor sit amet, consectetur adipiscing elit?

Lorem ipsum dolor sit amet, consectetur adipiscing elit?

Lorem ipsum dolor sit amet, consectetur adipiscing elit?

Cancel Save

Student can't update critical information including year-class, name, email and photo link

Tablet

Update profile

Year	Class
2009	Gecko
First Name	
John	
Last Name	
Rogers	
Email address	
john.rogers@gmail.com	
Yearbook Photo	
https://i.pravatar.cc/300?img=16	
Contact displayed in profile	
Yearbook quote	
Lorem ipsum dolor sit amet, consectetur adipiscing elit?	
Lorem ipsum dolor sit amet, consectetur adipiscing elit?	
Lorem ipsum dolor sit amet, consectetur adipiscing elit?	
Lorem ipsum dolor sit amet, consectetur adipiscing elit?	

Mobile

/account/students/new (admin)

Desktop

The screenshot shows a web-based application for managing student records. At the top, there's a navigation bar with links for 'John - Admin', 'Students', 'Classes', and 'Yearbooks'. Below the navigation, the main title is 'Add Students'. On the left, there's a dropdown menu for 'Year' set to '2010'. To its right is another dropdown for 'Class' with 'Kangaroo' selected. A red 'Delete' button is positioned next to the class name. The main form area contains fields for 'First Name', 'Last Name', 'Email address', and 'Yearbook photo link'. At the bottom, there are 'Cancel' and 'Save' buttons.

Tablet

This screenshot shows the same 'Add Students' form as the desktop version, but it's displayed on a tablet screen. The layout is similar, with the 'Add Students' title at the top. The 'Year' and 'Class' dropdowns are present. The student information fields (First Name, Last Name, Email address, Yearbook photo link) are below. The 'Add new student' button is at the bottom, along with 'Cancel' and 'Save' buttons.

Mobile

This screenshot shows the 'Add Students' form on a mobile device. The interface is designed for touch input. At the top, there's a 'YEAR BOOK' icon and a three-line menu icon. The 'Add Students' title is followed by 'Year' and 'Class' dropdowns, both set to '2010' and 'Kangaroo'. Below these are the student information fields. The 'Add new student' button is at the bottom, flanked by 'Cancel' and 'Save' buttons. The overall design is more compact and focused on mobile interaction.

/account/students (admin)

Desktop

The screenshot shows a 'Manage Students' page with two student records listed. Each record includes First Name, Email, Last Name, Student Id, and two buttons: 'Delete' (red) and 'Edit' (blue). A note at the top states: 'Note: Deleting students will delete the linked user as well'. Filter dropdowns for 'Year' (2010) and 'Class' (Kangaroo) are visible at the top.

First Name	Email	Last Name	Student Id
john	aa@gmail.com	doe	64f078f3c41cc9e56ea4a0c9
Greg	gregcarpenter@email.com	Carpenter	64f186cf5102a00fe3246899

[Add new students](#)

Tablet

The screenshot shows the same 'Manage Students' page as the desktop version, but with three student records listed. The layout and data are identical to the desktop version, including the note about deleting users, filter dropdowns, and the 'Add new students' button.

First Name	Email	Last Name	Student Id
john	aa@gmail.com	doe	64f078f3c41cc9e56ea4a0c9
Greg	gregcarpenter@email.com	Carpenter	64f186cf5102a00fe3246899

[Add new students](#)

Mobile

The screenshot shows the same 'Manage Students' page as the desktop and tablet versions, but with four student records listed. The layout and data are identical to the previous versions, including the note about deleting users, filter dropdowns, and the 'Add new students' button.

First Name	Email	Last Name	Student Id
john	aa@gmail.com	doe	64f078f3c41cc9e56ea4a0c9
Greg	gregcarpenter@email.com	Carpenter	64f186cf5102a00fe3246899

[Add new students](#)

After students are created in the database (from last slide), admin can go to this link to view generated student id. Admin can then send these details to students for them to register in the system (assuming this is done outside the app)

/account/classes/new (admin)

Desktop

A screenshot of a web application showing an 'Add Class' form. At the top, there's a navigation bar with links for 'John - Admin', 'Students', 'Classes', and 'Yearbooks'. Below the navigation is a title 'Add Class'. The form consists of two input fields: 'Year' and 'Class'. Each field has a placeholder text ('Year' and 'Class' respectively) and a blue 'Save' button at the bottom right. There are also 'Cancel' buttons at the bottom left of each field.

Tablet

A screenshot of a web application showing an 'Add Class' form. The layout is similar to the desktop version, with a 'Year' input field and a 'Class' input field, each with a 'Save' button at the bottom right and a 'Cancel' button at the bottom left. The overall design is responsive, though some elements like the navigation bar and title are partially cut off by the tablet's screen boundaries.

Mobile

A screenshot of a web application showing an 'Add Class' form. The mobile view is very compact. The 'Year' and 'Class' input fields are stacked vertically. Each field has a 'Save' button at the bottom right and a 'Cancel' button at the bottom left. A small circular icon labeled 'YEAR BOOK' is visible in the top left corner, and a three-line menu icon is in the top right corner.

/account/classes (admin)

Desktop

Manage Classes

Note: Only classes without students and empty years without classes can be deleted.

Year: 2020

Year	Class
2020	Tiger

Total Students
0

[Delete](#) [Edit](#)

Year: 2010

Year	Class
2010	Kangaroo

Total Students
2

[Delete](#) [Edit](#)

Year	Class
2010	Possum

Total Students
0

[Delete](#) [Edit](#)

Year: 2009

Year	Class
2009	Gecko

Total Students
1

[Delete](#) [Edit](#)

Year	Class
2009	Salamander

Total Students
1

[Delete](#) [Edit](#)

[Add new class](#)

Tablet

Manage Classes

Note: Only classes without students and empty years without classes can be deleted.

Year: 2020

Year	Class
2020	Tiger

Total Students
0

[Delete](#) [Edit](#)

Year: 2010

Year	Class
2010	Kangaroo

Total Students
2

[Delete](#) [Edit](#)

Year	Class
2010	Possum

Total Students
0

[Delete](#) [Edit](#)

Year: 2009

Year	Class
2009	Gecko

Total Students
1

[Delete](#) [Edit](#)

Year	Class
2009	Salamander

Total Students
1

[Delete](#) [Edit](#)

[Add new class](#)

Mobile

Manage Classes

Note: Only classes without students and empty years without classes can be deleted.

Year: 2020

Year	Class
2020	Tiger

Total Students
0

[Delete](#) [Edit](#)

Year: 2010

Year	Class
2010	Kangaroo

Total Students
2

[Delete](#) [Edit](#)

Year	Class
2010	Possum

Total Students
0

[Delete](#) [Edit](#)

Year: 2009

Year	Class
2009	Gecko

Total Students
1

[Delete](#) [Edit](#)

Year	Class
2009	Salamander

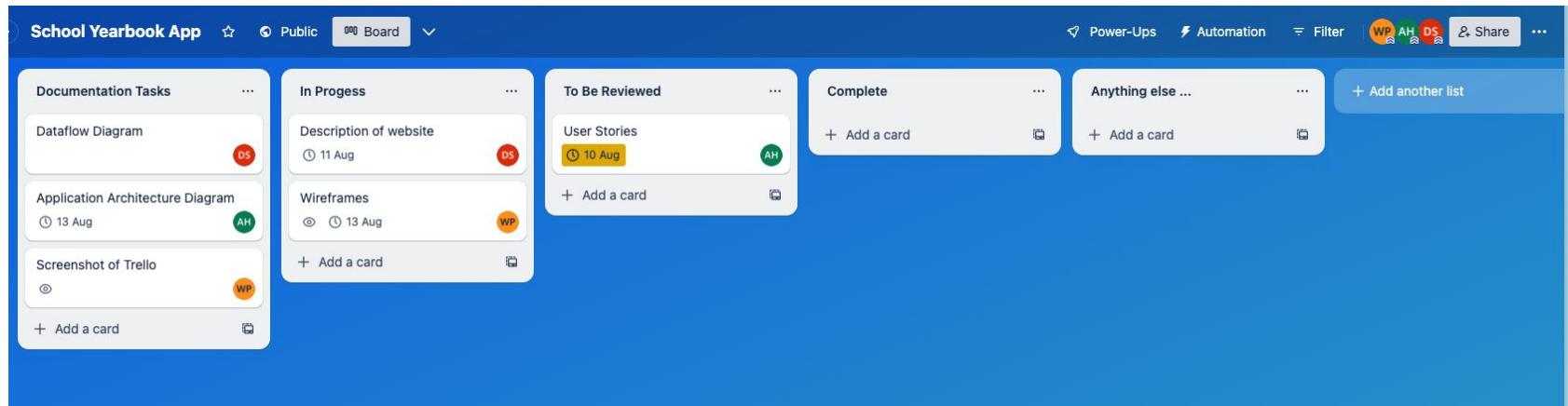
Total Students
1

[Delete](#) [Edit](#)

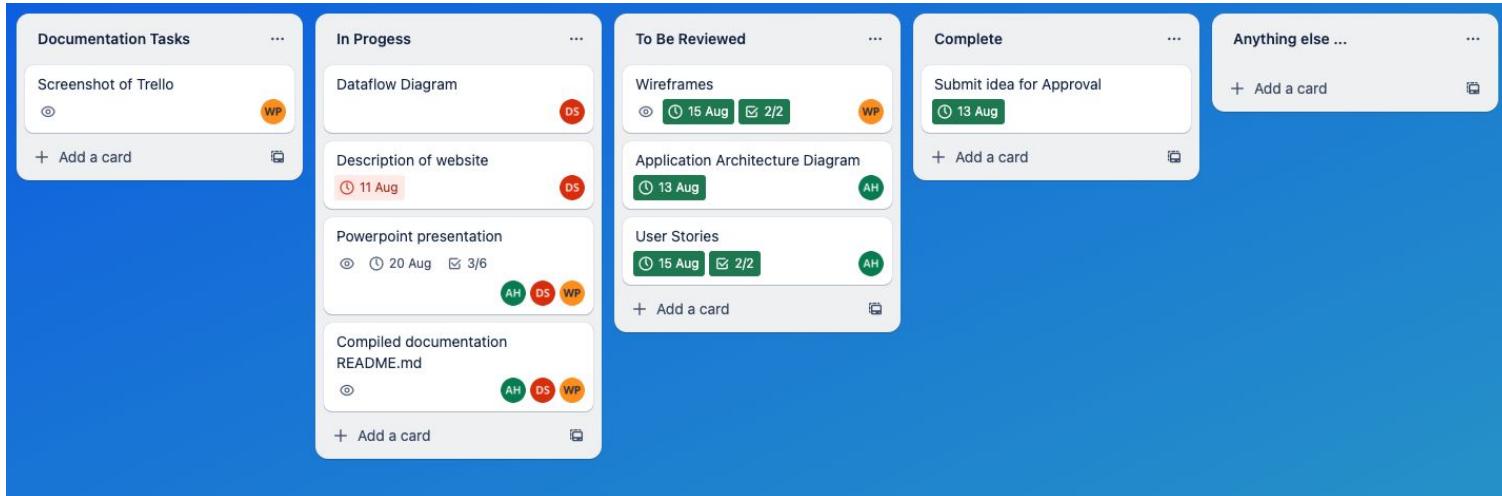
[Add new class](#)

Trello board

9 August: project start



19 August: near-end of part A



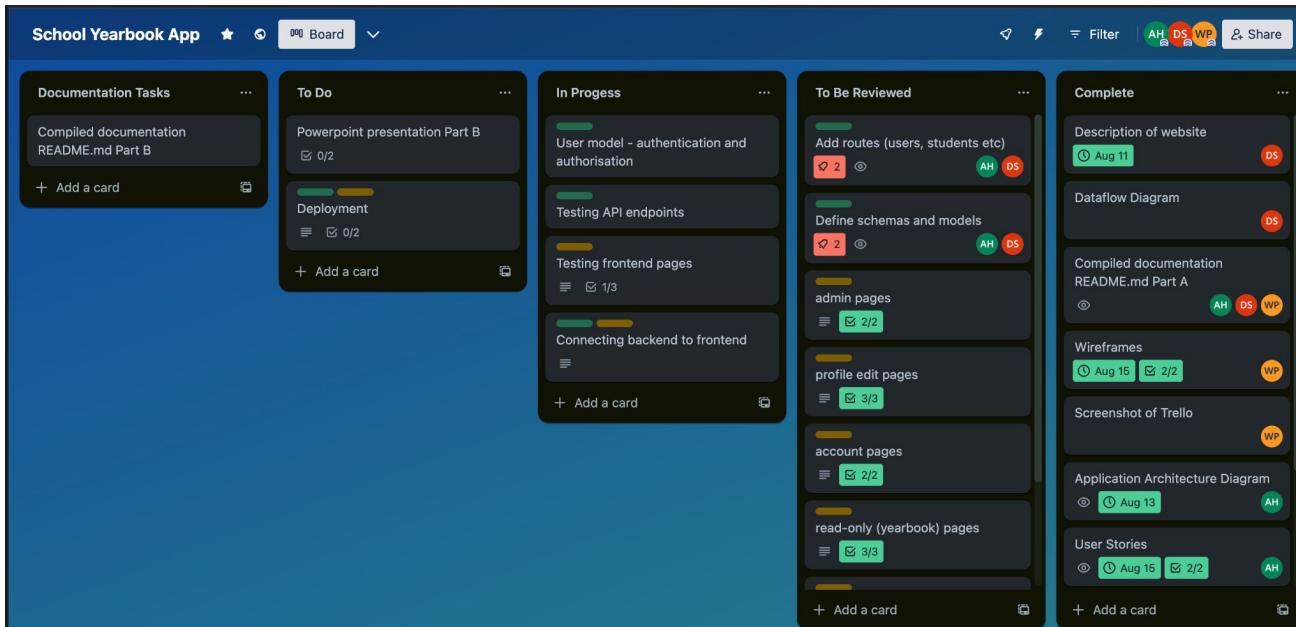
21 Aug: Start of Part B

The screenshot shows a Trello board titled "School Yearbook App". The board has six columns:

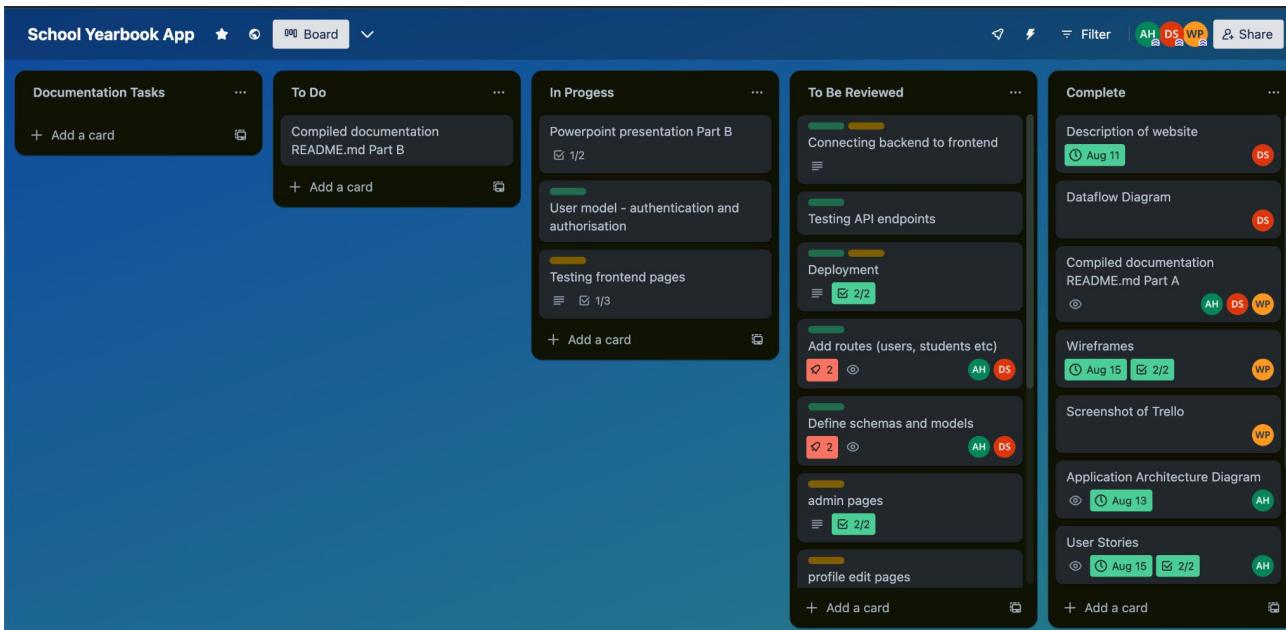
- Documentation Tasks**: Contains cards for "Powerpoint presentation Part B" (0/2) and "Compiled documentation README.md Part B". A "+ Add a card" button is available.
- To Do**: Contains cards for "User model - authentication and authorisation" (0/2), "Testing API endpoints" (0/3), "Authorisation pages" (0/3), "read-only pages" (0/3), "account pages" (0/2), "admin pages" (0/2), "profile edit pages" (0/3), "Testing frontend pages" (0/3), "Connecting backend to frontend" (0/2), and "Deployment". A "+ Add a card" button is available.
- In Progress**: Contains cards for "Define schemas and models" (0/2), "Add routes (users, students etc)" (0/3), and "Misc pages" (0/3). A "+ Add a card" button is available.
- To Be Reviewed**: Contains a "+ Add a card" button.
- Complete**: Contains cards for "Description of website" (11 Aug, 0/2), "Dataflow Diagram" (0/2), "Compiled documentation README.md Part A" (0/3), "Wireframes" (15 Aug, 2/2), "Screenshot of Trello" (13 Aug, 0/2), "Application Architecture Diagram" (13 Aug, 0/2), "User Stories" (15 Aug, 2/2), "Submit idea for Approval" (13 Aug, 0/2), and "Powerpoint presentation part A" (20 Aug, 6/6). A "+ Add a card" button is available.
- Anything else ...**: Contains a "+ Add a card" button.

At the top of the board, there are buttons for "Power-Ups", "Automation", "Filter", and "Share". There are also three circular icons with initials: WP, AH, and DS.

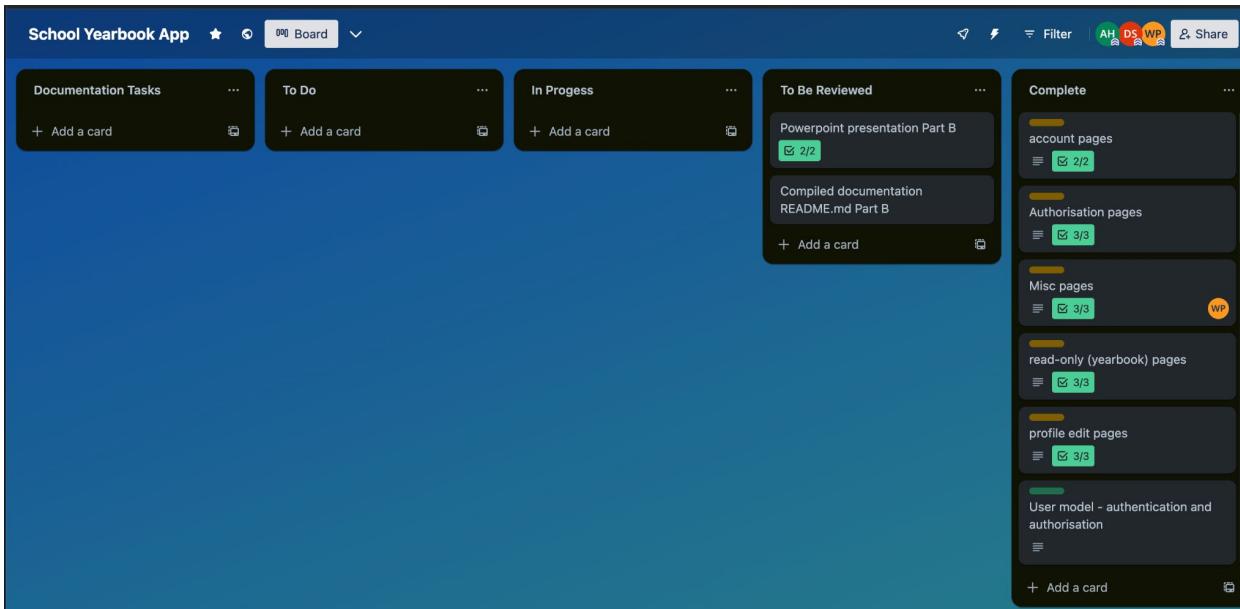
30 August



02 September



03 September - Completion



Subtasks (Part A)

User Stories
in list To Be Reviewed

Members Notifications Due date
 +  Watch  15 Aug at 17:00  Complete

Description
Add a more detailed description...

Checklist
100% 
 Initial user stories
 Refined user stories with new features

Add an item

Wireframes
in list To Be Reviewed

Members Notifications Due date
 +  Watching  15 Aug at 21:00  Complete

Description
Add a more detailed description...

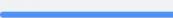
Checklist
100% 
 complete draft wireframes
 refine wireframes based on team discussion

Add an item

Powerpoint presentation
in list In Progress

Members Notifications Due date
 +  Watching  tomorrow at 23:30

Description
Add a more detailed description...

Checklist
50% 
 Description of website
 Dataflow Diagrams
 App-Architecture-Diagram
 User-Stories
 Wireframes
 Trello screenshot

Add an item

Subtasks (Part B)

❑ Add routes (users, students etc)
in list [To Be Reviewed](#)

Members Labels Notifications
 + Backend +

☰ Description Edit
Due: 27 Aug
Difficulty: Hard
Description:

- /students and /students/:id CRUD routes
- /classes and /classes/:id CRUD routes
- /years and /years/:id CRUD routes

Assigned to: Dieter, Alicia

❑ Misc pages
in list [To Be Reviewed](#)

Members Labels Notifications
 + Frontend +

☰ Description Edit
due: 22 Aug
Difficulty: Easy
Description:

- set up BrowserRouter and basic routes
- Finish components for landing page and general views (message, navbar)
- feature branch: misc-pages

Assigned to: Wen

More in [Trello](#)