Project Proposal Report - Team 6

Abby Davidow, Grant Schnettgoecke, Archana Ramakrishnan, Adam Soelter, Alfonso Martello

Project Name: What's Due When (WDW)

Project Synopsis:

What's Due When web application lets students see upcoming schoolwork for all their classes in one place and allows professors to add/edit their class schedules.

Project Description:

College students are constantly barraged by emails, announcements, and handouts from a wide variety of different sources. Keeping track of this deluge can be overwhelming, especially for freshmen just entering the college world. This information overload can add unnecessary stress to students' lives and managing it can take substantial time and resources. What's Due When (WDW) aims to solve this problem by providing a central application that faculty and students can use to control the flow of information. Faculty can use WDW to add events (such as notes, due dates, or comments) directly to their students' calendars on either a weekly, monthly, or semester basis. Furthermore, faculty can opt-in to receive reminder emails to input information. Students can use WDW to edit or add events to their own calendar with the extra benefit of having all announcements relevant to them gathered in one convenient place. The result will be an online platform (specifically, a website designed using React.js) where students and faculty can have schedules delivered in a manageable and convenient way.

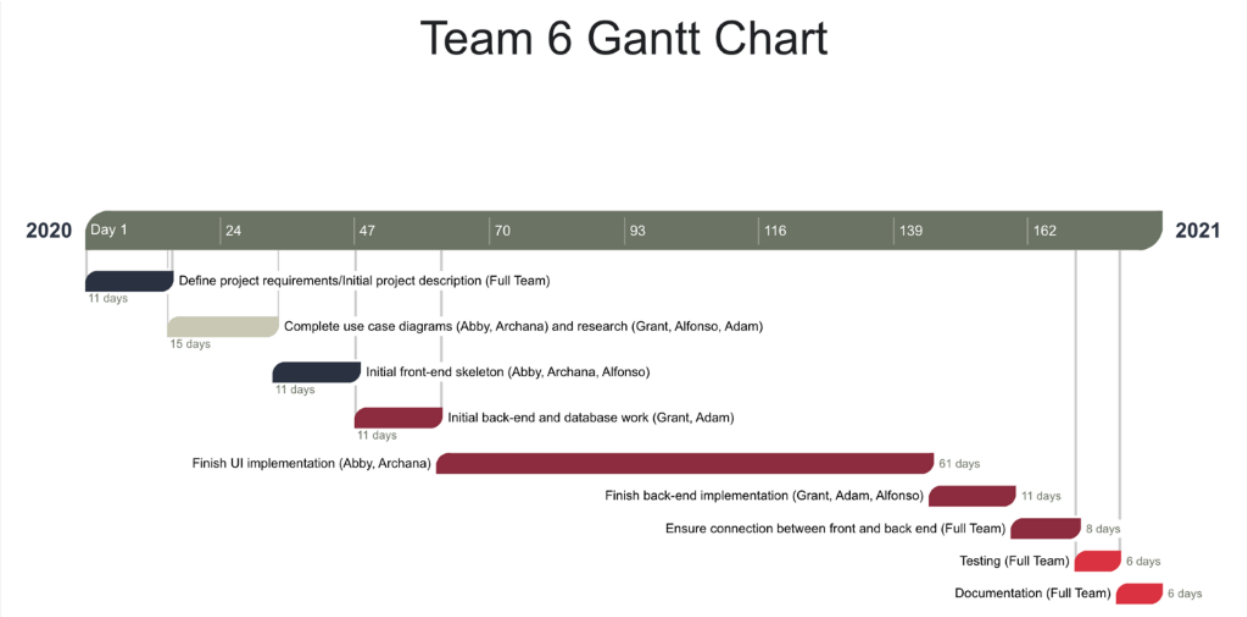
Project Milestones:

Fall Semester

- 10/5/20 - Define project requirements/Initial project description
- 10/23/20 - Complete use case diagrams and research
- 11/6/20 - Initial front-end skeleton
- 11/20/20 - Initial back-end and database work

Spring Semester

- 2/12/21 - Finish UI implementation
- 2/26/21 - Finish back-end implementation
- 3/9/21 - Ensure connection between front and back end
- 3/16/21 - Testing
- 3/23/21 - Documentation



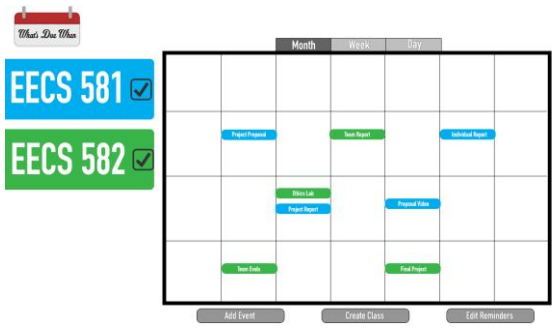
Project Budget:

The costs incurred will be for the domain name, web hosting service, and the database. A one-year registration from *Namecheap for Education* costs $8.88 for whatsduewhen.com and $24.88 for whatsduewhen.io. Until the KU web server hosts our website, we could use GitHub to host the application for free. If we do not integrate with KU, we can use Hostinger, which has a Business Shared Hosting plan for small businesses. It would cost us $3.99 monthly with $7.95 monthly when you renew the plan. On an enterprise level, MongoDB would have a monthly cost of $57, providing 10 GB to 4 TB of storage and 2 GB to 768 GB of RAM. It also has a developer version which is free of cost and scales up to 5 GB in storage. Special training in React JS and databases will be required by the end of October/early November to start implementing the project.

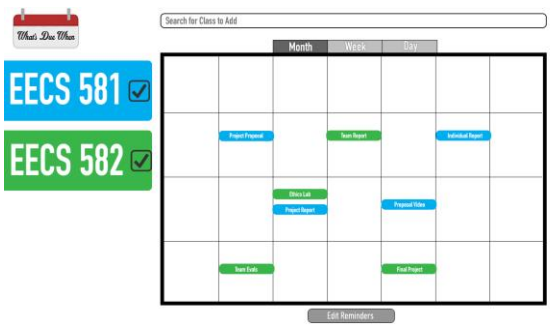
Preliminary Project Design:

How the Software Works

Faculty View



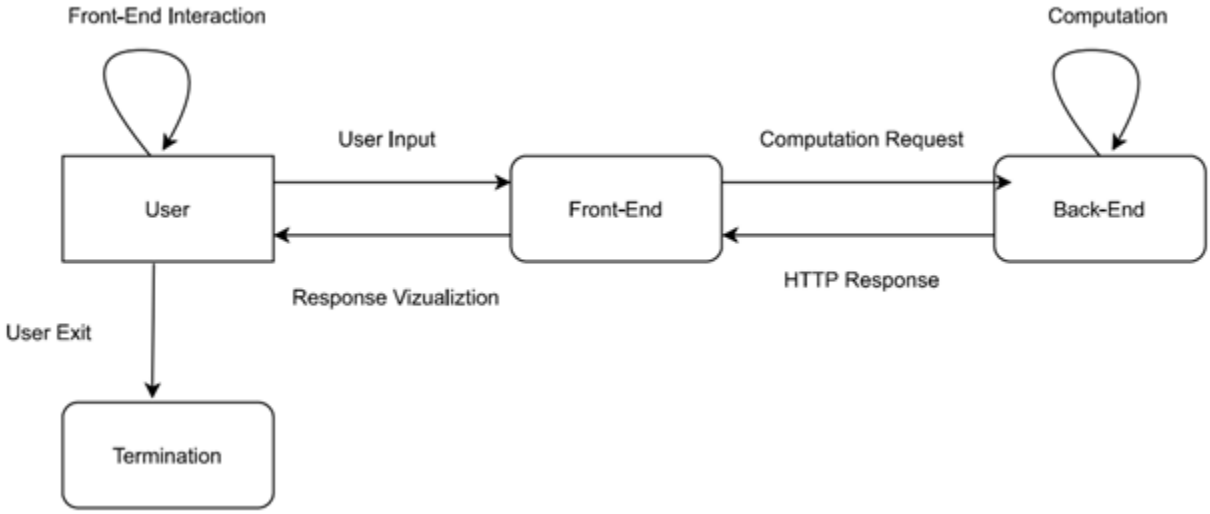
Student View



Our website will have five main UI views for users. The first view for any user will be the login page. Faculty and student users will have separate homepages, however, the view for both users will display a calendar containing events for all their classes as shown in the mock designs above. On this page, they will be able to adjust their view and choose other editing options. Users can access another view to edit the email reminders they receive for each event. The difference between faculty and student views is the additional option to create a new class and add events. Clicking on these options will bring up a form view where all required input will be entered in order to create a new course or event.

For the front end of our website, we will be using React JS library to build our UI. We decided on React due to its fast learning curve and re-usability in terms of components. A couple of our team members have some experience with React and found it to be straightforward to learn and has helpful developer tools. The reusability will be key for our design since the student and faculty views will require many similar components.

Another key component of our, and any, web project is the backend. The backend is the portion of the code that runs on a server, performs computations, and responds to user input remotely. The user input is generated by the user by interacting with our ReactJS front-end before being transmitted through an HTTP protocol to our back end. After the back end performs the necessary computations requested by the user, it generates a response which it transmits back through an HTTP protocol. This process continues until the user stops interacting with the application. This process is summed up in the figure shown below.

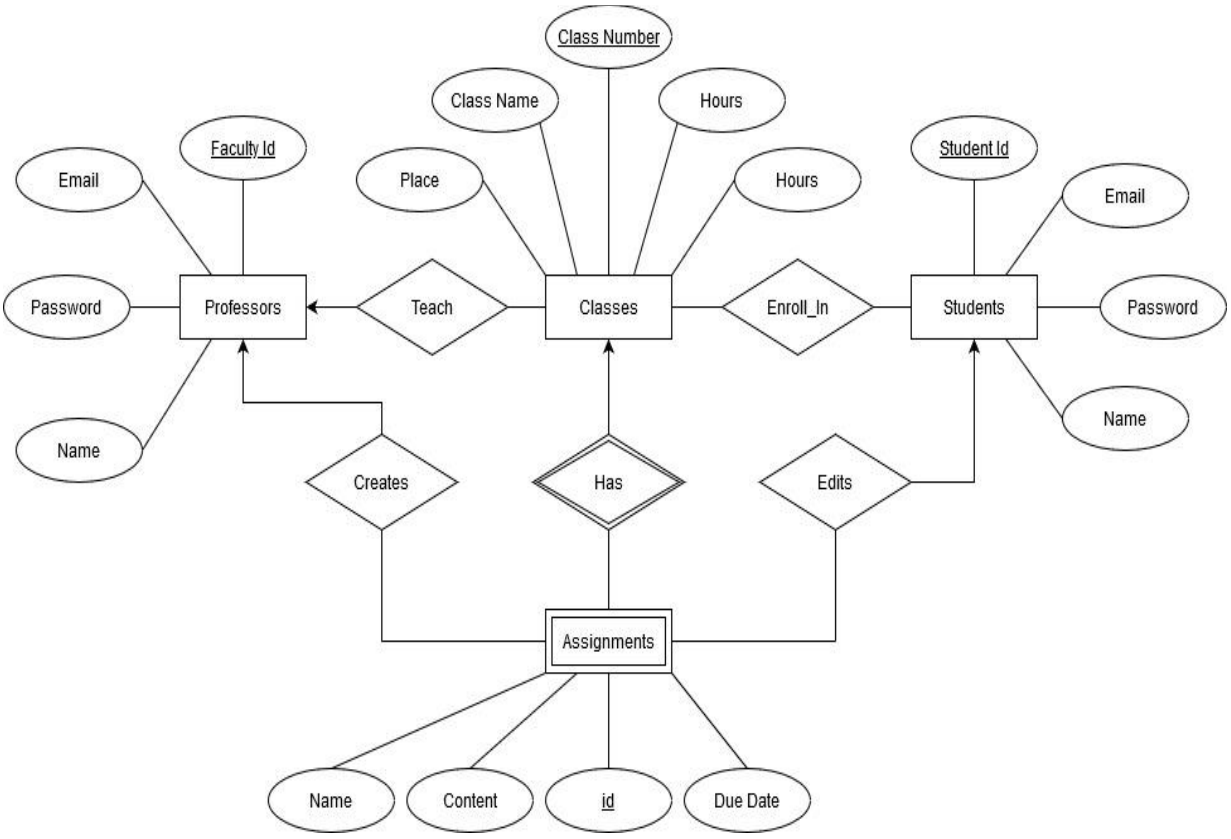


This server client architecture allows us to move computations away from possibly resource limited user devices such as cell phones and tablets and towards the more performant machines that run the server. Additionally, it allows us to share common data in one process (the back end) rather than duplicating the same data across multiple processes/devices on the front end. For example, many students are enrolled in the same class. So, it would be wasteful to store information about that class on every user device. Instead, we can store one copy of that information on the server and let every user access that information over the internet when that information is needed by the user.

For the back end of our project, we intend to utilize NodeJS. NodeJS is a cross platform backend for web applications that allows the user to run JavaScript code outside of the browser.

One of the main benefits we receive from using NodeJS, in conjunction with our ReactJS frontend, is that we can write the front end and back ends in the same language. Additionally, NodeJS is designed to be more performant than traditional back-end models because it makes use of a single threaded event loop rather than a traditional request/response model. This allows it to avoid the context switching overhead that is common in other back-end API's.

For our database, we were split between using MongoDB and MySQL. Both database systems have free versions we could use for our project. MongoDB stores its data inside of JSON files so changing the structure of the data is simple and quick, which would allow us to easily change how our project functions in the future. MySQL on the other hand stores its data inside of tables and requires new tables to be made or deleted whenever a change is made making it a more expensive process to change the project if data needs to be stored in a different way. However, there are three compelling reasons for why we chose to use MySQL. In accordance with the ACM Code of Ethics and Professional Conduct rule 2.6 "Perform work only in areas of competence", since more team members know how to use MySQL than MongoDB, we have chosen to use MySQL. Furthermore, our data has a simple structure with clear relations making it an ideal candidate for using MySQL as it is a relational database. Finally, KU has MySQL servers that we can use, even if only for one semester, to store our data so that way student data can be protected by KU and our application can meet FERPA standards during development. Observe the ER Diagram (Entity Relationship Diagram) for our MySQL database below:



The above diagram shows our 4 data tables (Professors, Classes, Students, and Assignments) and how they relate to each other (the diamond connections). Additionally, it shows what types of data each table will hold (for example, the Student table will contain a student's id, email,

password, and name). This diagram is a rough sketch for now. Ideally, we will not actually need to store student and professor emails or password as we will be using a single-sign in API (possibly the KU single-sign on or Google sign in) to handle user authentication. Every table must have a primary key for data entry identification purposes. These primary keys are denoted by an underlined attribute. Furthermore, the Assignments table has an identifying relationship with the Classes table as an assignment cannot exist without it being associated with a class. Finally, the use of arrows denotes a one-to-many relationship (with the arrow representing the one and the lack of an arrow representing many). For example, one professor can teach many classes, but a class is usually not taught by many professors.

Design Constraints

Technical Constraints

Programming language: We are considering using the MERN stack. It stands for MongoDB, Express, ReactJS and NodeJS. Express is a framework that works with Node.js's web server to organize the functionality of the application. It simplifies routing, renders dynamic HTTP objects, and adds utilities to the HTTP objects. ReactJS is a library to build interfaces using reusable components. NodeJS is a backend environment that facilitates returning content from the server to the client.

Upon looking at the differences between a traditional RDBMS and MonogoDB, which uses noSQL, we made some observations. While RDBS will help lay out schema design that links related tables, noSQL databases are built to have flexible schemas and specific data models. MongoDB will be fast and highly scalable. Depending on if our application will be put to mass usage in the future, we will choose our database accordingly.

Use of specific libraries**:** Node Package Manager (NPM) has a react-calendar package that we could implement for the user interface. This package provides a small calendar view to pick days, months, years, or even decades. It is fast, lightweight and easy to style. Additionally, the react-big-calendar is an events calendar component, which uses flexbox. This will be helpful to build in events into each date in addition to the mini calendar view that the first package provides. NPM also provides a react-google-calendar-api to manage a google calendar and node-outlook to integrate the Outlook calendar. We may opt into using one of these if we are able to introduce some level of customization to fit our use case. It might help to stay consistent if we use google/outlook authentication.

Operating system or platforms supported: We aim to build the web application to be accessible through a desktop or mobile browser. The proper rendering of a lot of the front-end components in React will determine the platforms supported. The react-calendar package supports only modern browsers-Internet Explorer 11, Edge 12-85 and 86, and modern versions of Chrome, Safari and Opera on desktop. On mobile, it supports iOS Safari 10-13.7 and 14, Android Browser 81, Opera Mobile 59, Chrome 85 and Firefox 79 on Android. The react-big-calendar package also works on modern browsers but does not have specific constraints of supported browsers.

Business Constraints

Schedule: The project will be ready to potentially integrate into the KU system by the end of April 2021. The timeline leading up to that goal involves planning, communicating with the

stakeholders in addition to iterative prototyping, implementing and getting user feedback. We picked the project idea in late September 2020. In early October, we outlined the requirements by meeting with the professor who introduced this idea. Following that was researching the technologies we would use and planning the design through use case, database schema, and wireframe diagrams. The bulk of the implementation will start in late November and a minimum viable product will be built by December. We will iterate on that product after some feedback to integrate the student and professor use case by March. March and April will be our last spring of enhancements, bug fixes and database integration.

Team composition and make-up: We have a team with diverse skill sets and interests. Grant and Alfonso will work as an intermediary between the front end and back end sides of the application. Adam will focus on perfecting the back-end functionality, while Alfonoso will support the database management and configurations. Abby and Archana will primarily work to design and enhance the user interface and experience on the front end. These roles will complement each other and shift as our priorities shift through the project timeline.

Ethical Issues:

A major ethical issue that we will have to account for as we implement our project is ensuring students' rights as laid out in the Family Educational Rights and Privacy Act (FERPA) are protected. Students have the right to choose what is disclosed about their education records. This includes grades, the classes they take, or any other identifiable information from their education record. If we are going to store information about the classes a student is taking in our database, we need to take appropriate measures to protect their information from possible hacks into our database. One way we could increase the security of our database is by double hashing the information we store, so if there is a breach, the data that would be vulnerable is protected by a second hash. Another approach would be to store our data on KU's MySQL servers during development so user data is already protected by safeguards KU has in place.

Another minor ethical issue that we need to abide by is ACM Code of Ethics and Professional Conduct rule 1.3 "Be honest and trustworthy." The concern for this rule is that our project is being made to make time management easier for students. However, in achieving this goal we also need to convince faculty to use our project. Thus, we have the two-pronged job of designing our software with students and teachers in mind. We will need to be upfront about the purpose behind our project and its capabilities throughout the development process. We are toying around with many ideas to make our software more faculty friendly, but we should not advertise these ideas as features until we know they will be implemented.

Intellectual Property Issues:

An intellectual property issue that could arise is if we plan to integrate our project with KU. We plan to integrate single sign on through Blackboard for students and professors. KU would also be hosting our application and storing information regarding students' class schedules. Because of this, KU might not allow us the right to some of the code. There are a couple of routes that we could go. We could decide that we want to have the rights to everything that we create and have it as a stand-alone application. KU would then decide if they want to buy our software or not. We would then want to make sure we copyright the code that contains the core functionality of our application. Another route we could take is to work with KU. We would sign a contract

stating what our property is and what property they will have. The code that will be directly related to helping KU professors and students will be theirs. We will still have rights to the code that is responsible for the main functionality of the web app. Lastly, we may decide that we want to make our project open-source and let anyone use our application. This would allow anyone to use our code and modify it however they like. Since anyone will be able to use our application, we would not have to worry about any intellectual property issues.

Change Log:

- We have reworked the budget for our application to include specific vendors and pricing.