

# Class Scheduler

Design Document

Team 1

ECpE Student Services

Brian Schomer (Meeting lead/minutes taker), Carter Everts  
(Testing lead), Isaiah John Ortiola (Ind. Component  
Designer), Lewis Callaway (Client Interaction), Michael Less  
(Tech Lead), Simeon Steward (Team organization)

[Sdmay24-01@iastate.edu](mailto:Sdmay24-01@iastate.edu)

<https://sdmay24-01.sd.ece.iastate.edu>

# Executive Summary

## Problem

Our senior design team developed a program to help the ECpE office schedule proposed new classes or sections by finding open slots based on conflicting courses and course groupings.

## Summary of Requirements

- Create a Desktop app that can find valid class times for new classes.
- App will minimize conflicts for students that the course is designed for.
- App will recommend timeslots so that user can choose what works best.
- App has a user interface that is simple to use.
- App must be able to input and store courses that are already scheduled.

## Applicable Courses from Iowa State University Curriculum

- Computer Science: 228, 311, 309, 319, 363
- Software Engineering: 317
- Computer Engineering: 230

## New Skills/Knowledge acquired that was not taught in courses

Scheduling Algorithms, Front-end Development

# Table of Contents

## Contents

Contents .....	2
1 Team .....	6
1.1 TEAM MEMBERS.....	6
1.2 REQUIRED SKILL SETS FOR YOUR PROJECT .....	6
Front-End Programming (specifically desktop applications), Backend Programming (including algorithms for scheduling/conflict graphs), communication with client, time management.....	6
1.3 SKILL SETS COVERED BY THE TEAM.....	6
All team members have gone through the same set of core classes and have an ability to troubleshoot and do full-stack development. This means there will be overlap with the skills. ....	6
Frontend: Michael Less.....	6
Backend: Simeon Steward, Brian Schomer .....	6
Testing: Carter Everts .....	6
Component Designer: Isaiah Ortiola.....	6
Client Communication: Lewis Callaway.....	6
Time Management: Simeon, Lewis, and Brian.....	6
1.4 PROJECT MANAGEMENT STYLE ADOPTED BY THE TEAM.....	6
We will follow the agile system with roles assigned to the teammates as detailed below. ....	6
1.5 INITIAL PROJECT MANAGEMENT ROLES.....	6
1.5.1 TEAM LEAD .....	6
1.5.2 MEETING LEAD .....	6
1.5.3 CLIENT COMMUNICATIONS .....	7
2 Requirements .....	7
2.1 PROBLEM STATEMENT.....	7
2.1.1 FUNCTIONAL REQUIREMENTS.....	7
2.1.1.1 Input each major's core classes per semester (a CPRE Freshman takes Math 165, Chem 167, CPRE 185, etc). Provide fields for course title, time requirement (50 minutes, 90 minutes, etc), and available spots.....	7
2.1.1.2 Add the remaining ECpE courses that are not required.....	7

2.1.1.3	Create a scheduling algorithm that takes into account various constraints and preferences. Algorithm should be used to find a free period for a new course Avoid scheduling two courses with overlapping content at the same time.....	7
	Ensure that specific courses (as defined in the rules) are not scheduled back to back .....	7
2.1.1.4	Input the available times for the courses in the system. If possible, pull data from the classes.iastate.edu server. ....	7
2.1.1.5	Create a new course that needs to be scheduled. ....	7
2.1.1.6	When creating the new course, add "rules" that the class needs to find free periods around. Ex: don't conflict with other 6,7,8 semester CPRE core classes Ex: don't schedule the same time as another determined class.....	7
2.1.1.7	Allow for course time import from classes.iastate.edu (if possible) .....	7
2.1.1.8	Handle cross-listed courses correctly (it is the same course so don't treat it as two separate courses).....	7
2.1.1.9	User shall be able to view current schedule .....	7
2.1.1.10	User shall be able to view class details .....	7
2.1.2	RESOURCE REQUIREMENTS.....	8
2.1.2.1	Modern Development Environment (IntelliJ IDE, Gitlab) .....	8
2.1.2.2	Library to develop a desktop application smoothly.....	8
2.1.2.3	Program should be easy to install since it is going to be non-engineers installing the program (resource constraint) .....	8
2.1.3	UI REQUIREMENTS AND QUALITATIVE AESTHETIC REQUIREMENTS .....	8
2.1.3.1	Desktop application .....	8
2.1.3.2	Easily read grid of already scheduled classes.....	8
2.1.3.3	Menus to create classes and look for available times. ....	8
2.1.3.4	Ensure that schedules are formatted in a user-friendly and legible manner .....	8
2.3	INTENDED USERS AND USES.....	8
3	Project Plan.....	8
3.1	TASK DECOMPOSITION .....	8
3.2	PROJECT MANAGEMENT/TRACKING PROCEDURES.....	10
3.3	PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA .....	11
3.4	PROJECT TIMELINE/SCHEDULE.....	12

**SD May 24 Team 1**

**Scheduler for ECPe office**

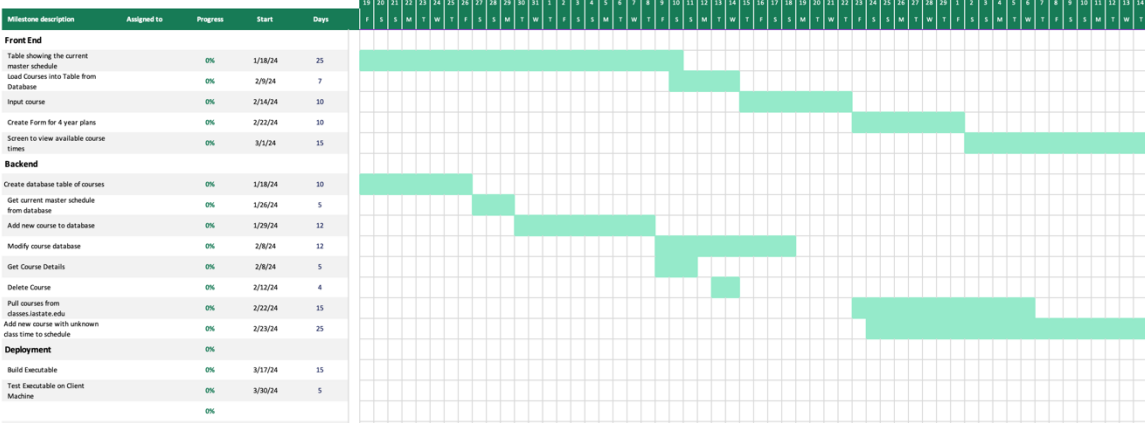
**Team 1**

Project start date: 1/18/24



Scrolling increment: 1

Milestone marker: 1



.....12

**Scheduler for ECPe office**

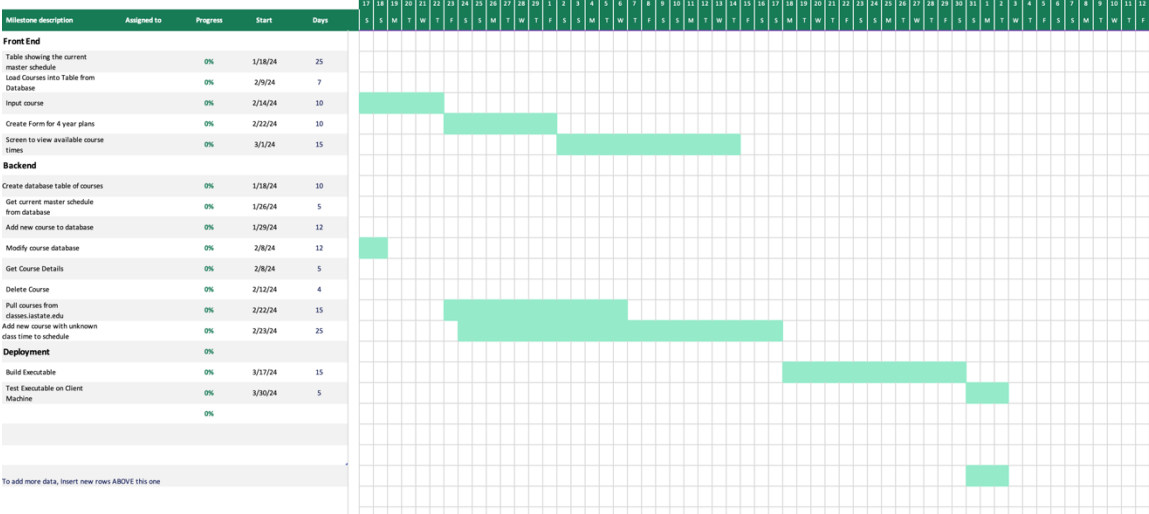
**Team 1**

Project start date: 1/18/24



Scrolling increment: 30

Milestone marker: 1



.....12

3.5 Risks And Risk Management/Mitigation .....12

3.6 PERSONNEL EFFORT REQUIREMENTS..... 16

3.7 OTHER RESOURCE REQUIREMENTS ..... 16

4 Design ..... 16

4.1 Design Content ..... 16

4.2 Design Complexity.....17

4.3 Modern Engineering Tools.....17

4.4 Design Context..... 18

4.5	Prior Work/Solutions .....	19
4.6	Design Decisions.....	19
4.7	Security Concerns .....	25
4.8	Appendix A – Time Slots .....	25
4.9	Appendix B – Database Design Diagram.....	26
5	Testing .....	26
5.1	Unit Testing.....	26
5.2	Interface Testing.....	27
5.3	Integration Testing.....	27
5.4	System Testing .....	27
5.5	Regression Testing .....	27
5.6	Acceptance Testing.....	27
5.7	Security Testing.....	28
5.8	Results.....	28
6	Implementation.....	28
6.1	Detailed Design.....	28
6.2	Description of Functionality .....	28
6.3	Notes on Implementation .....	29
7	Professionalism .....	29
7.1	Areas of Responsibility .....	29
7.2	Project Specific Professional Responsibility Areas .....	29
7.3	Most Applicable Professional Responsibility Area .....	30
8	Conclusions .....	32
8.1	Review Progress .....	32
8.2	Value Provided by Design .....	32
8.3	Future Steps.....	32
9	Appendix 1 – Install/Operation Manual .....	33
9.1	Installation .....	33
9.2	Operation .....	39

# 1 Team

## 1.1 TEAM MEMBERS

Simeon Steward

Brian Schomer

Lewis Callaway

Isaiah Ortiola

Carter Everts

Michael Less

## 1.2 REQUIRED SKILL SETS FOR YOUR PROJECT

Front-End Programming (specifically desktop applications), Backend Programming (including algorithms for scheduling/conflict graphs), communication with client, time management.

## 1.3 SKILL SETS COVERED BY THE TEAM

All team members have gone through the same set of core classes and have an ability to troubleshoot and do full-stack development. This means there will be overlap with the skills.

Frontend: Michael Less

Backend: Simeon Steward, Brian Schomer

Testing: Carter Everts

Component Designer: Isaiah Ortiola

Client Communication: Lewis Callaway

Time Management: Simeon, Lewis, and Brian

## 1.4 PROJECT MANAGEMENT STYLE ADOPTED BY THE TEAM

We will follow the agile system with roles assigned to the teammates as detailed below.

## 1.5 INITIAL PROJECT MANAGEMENT ROLES

### 1.5.1 TEAM LEAD

Simeon Steward

### 1.5.2 MEETING LEAD

Brian Schomer

### 1.5.3 CLIENT COMMUNICATIONS

Lewis Callaway

## 2 Requirements

### 2.1 PROBLEM STATEMENT

When new courses need to be added, the ECpE department needs to schedule classes in a manner where courses are scheduled at a time that does not conflict for most students. The schedule is currently scheduled manually by utilizing a master spreadsheet and testing potential times by hand. Additionally, the reliance on a master spreadsheet makes it difficult to quickly adapt without more manual attempts at finding a free time. A user-friendly desktop application that can streamline the course scheduling process for the ECpE department will reduce scheduling conflicts for students by intelligently assigning class times based on predefined rules and constraints. There is currently no automated system in place to optimize the scheduling classes so our solution will be new.

### REQUIREMENTS & CONSTRAINTS

#### 2.1.1 FUNCTIONAL REQUIREMENTS

- 2.1.1.1 Input each major's core classes per semester (a CPRE Freshman takes Math 165, Chem 167, CPRE 185, etc). Provide fields for course title, time requirement (50 minutes, 90 minutes, etc), and available spots
- 2.1.1.2 Add the remaining ECpE courses that are not required.
- 2.1.1.3 Create a scheduling algorithm that takes into account various constraints and preferences.
  - Algorithm should be used to find a free period for a new course
  - Avoid scheduling two courses with overlapping content at the same time
  - Ensure that specific courses (as defined in the rules) are not scheduled back to back
- 2.1.1.4 Input the available times for the courses in the system. If possible, pull data from the classes.iastate.edu server.
- 2.1.1.5 Create a new course that needs to be scheduled.
- 2.1.1.6 When creating the new course, add "rules" that the class needs to find free periods around.
  - Ex: don't conflict with other 6,7,8 semester CPRE core classes
  - Ex: don't schedule the same time as another determined class
- 2.1.1.7 Allow for course time import from classes.iastate.edu (if possible)
- 2.1.1.8 Handle cross-listed courses correctly (it is the same course so don't treat it as two separate courses)
- 2.1.1.9 User shall be able to view current schedule
- 2.1.1.10 User shall be able to view class details



### 2.1.2 RESOURCE REQUIREMENTS

- 2.1.2.1 Modern Development Environment (IntelliJ IDE, Gitlab)
- 2.1.2.2 Library to develop a desktop application smoothly.
- 2.1.2.3 Program should be easy to install since it is going to be non-engineers installing the program (resource constraint)

### 2.1.3 UI REQUIREMENTS AND QUALITATIVE AESTHETIC REQUIREMENTS

- 2.1.3.1 Desktop application
- 2.1.3.2 Easily read grid of already scheduled classes.
- 2.1.3.3 Menus to create classes and look for available times.
- 2.1.3.4 Ensure that schedules are formatted in a user-friendly and legible manner

## 2.2 ENGINEERING STANDARDS

There are not many engineering standards apply to our project since it is a purely software project and it does not have to abide by any data protection standards since it does not deal with any type of protected information.

We plan to use Java and Spring Boot and will try to align with the respective programming standards. For Spring Boot we will separate the layers when possible to lead to smaller class files that are more easily maintained.

We are also going to attempt to make bug free code, but since bugs are in all code we will have to fix them as they pop up. We will make use of the sprint board to plan out which bugs have the largest impact and work to fix them as they pop up, leading to a much more stable product.

## 2.3 INTENDED USERS AND USES

The ECpE office staff (Vicky and Tina in particular) are the primary users of this program. They will use this application to record the already scheduled courses from ECpE and enter in the core schedule for an ECpE student. There would be one for EE, one for CPRE, and one for CyberE. Next, a new course would need to be scheduled. A new course would be created and then any conflicts that need to be avoided would be entered. The program would then return the potential times for the new course.

# 3 Project Plan

## 3.1 TASK DECOMPOSITION

[https://docs.google.com/drawings/d/1oRg6CpgnTUuXv8INf3QCAjmepitAHT3fR\\_11pdMPUHM/edit?usp=sharing](https://docs.google.com/drawings/d/1oRg6CpgnTUuXv8INf3QCAjmepitAHT3fR_11pdMPUHM/edit?usp=sharing)

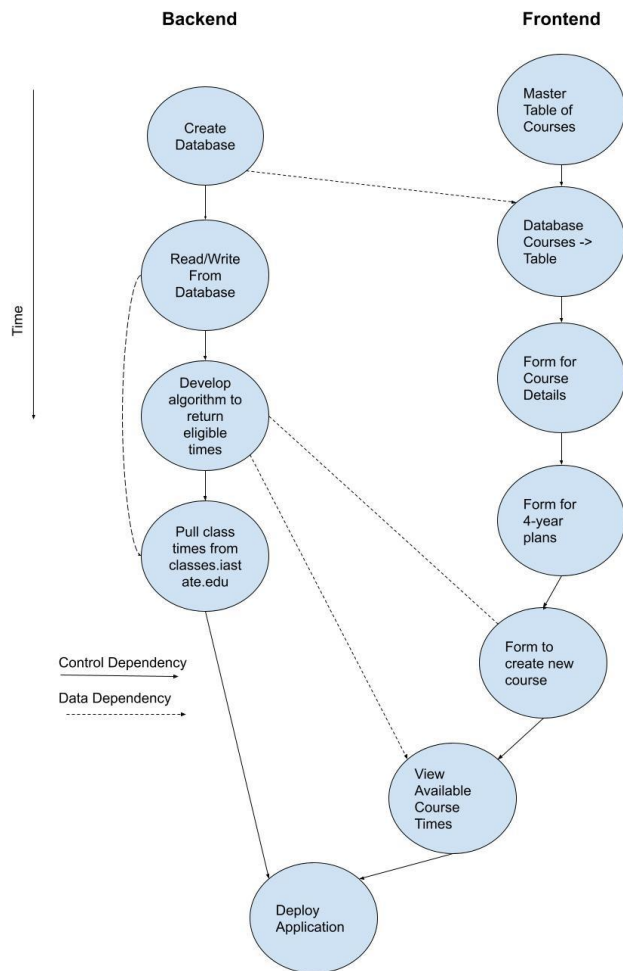


Figure 1: Task Decomposition Diagram

### 3.1.1 Front End - UI

- 3.1.1.1 Table showing the current master schedule for a specific semester (Req 3.2)
- 3.1.1.2 Make the table showing current master schedule read from backend (Req 3.2)
- 3.1.1.3 Form to input course, course times, and courses or schedules it can't conflict with (Req 1.6)
- 3.1.1.4 Form to input 4-year plan courses (Req 1.1)
- 3.1.1.5 Form to create a new course (Req 1.5)
- 3.1.1.6 Screen to view available course times that a new course could fit into (Req 3.4)

### 3.1.2 Front End - Backend Interface:

- 3.1.2.1 GET current master Schedule (Req 1.9)

- 3.1.2.2 POST new course with known class time (including class times, courses it can't conflict with) (Req 1.1, 1.2, 1.8)
- 3.1.2.3 PUT modify Course (including class times, courses it can't conflict with) (Req 1.1, 1.2, 1.8)
- 3.1.2.4 GET course details (including class times, courses it can't conflict with) (Req 1.10)
- 3.1.2.5 DELETE Course and associated data (Req 1.1, 1.2, 1.8)
- 3.1.2.6 [Extension] POST Add new course with unknown class time to current schedule (Req 1.5)
- 3.1.2.7 [Extension] GET possible schedules generated based on requirements (Req 1.3)
- 3.1.2.8 [Extension] POST Set current possible schedule to current master schedule (Req 1.3)

### 3.1.3 Backend - Business Logic

- 3.1.3.1 Get current master Schedule (Req 1.9)
- 3.1.3.2 Add new course (including class times, courses it can't conflict with) (Req 1.1, 1.2, 1.8)
- 3.1.3.3 Modify Course (including class times, courses it can't conflict with) (Req 1.1, 1.2, 1.8)
- 3.1.3.4 Get course details (including class times, courses it can't conflict with) (Req 1.10)
- 3.1.3.5 Delete Course and associated data (Req 1.1, 1.2, 1.8)
- 3.1.3.6 [Extension] Add new course with unknown class time to current schedule (Req 1.5)
- 3.1.3.7 [Extension] Generate possible schedules generated based on existing schedule (Req 1.3)
- 3.1.3.8 [Extension] Post Set current possible schedule to current master schedule (Req 1.3)

### 3.1.4 Data Access Layer / ORM & Database

- 3.1.4.1 Create a database table of courses
  - Course Name
  - Degree Requirements i.e. SE semester 3
  - Time & Duration
  - Professor?
- 3.1.4.2 Develop a solution to pull from classes.iastate.edu (Req 1.7)
- 3.1.4.3 Deployment:
- 3.1.4.4 Create an executable that can be run on the client's computer (Req 3.1)

## 3.2 PROJECT MANAGEMENT/TRACKING PROCEDURES

We are using waterfall+agile for this project. The reason it is waterfall+agile is that we do have a base level of code that needs to be written before it can even be confirmed with the client. We have clearly defined requirements upfront so this will give us a solid basis to build the software upon. We will check in with the client more frequently as soon as we have more visible changes and when it comes time to confirm changes were successful. Additionally, the general rule of thumb is that below a 200 person-day effort project, agile overhead is too much. So, we'll have a waterfall task decomposition and schedule but finish each task in an agile manner.

What will your group use to track progress throughout the course of this and the next semester. This could include Git, Github, Trello, Slack or any other tools helpful in project management.

Our team will use GitLab to formally track progress as well as use Discord for more informal updates.

### 3.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

The milestones will be split into front and backend to try and keep as little dependency as possible

Front end (these do not have metrics as they are more visual tasks):

Display the current master schedule

Load the table to display from a database

Import new courses

Create 4 Year Plans

Screens to view available course times

Rudimentary Desktop interface

Make sure Input is taken

Ensure some sort of Output

Backend:

Create database table of courses – Needs to be a file that can loaded and transferred from one person to the next

Get current master schedule from database – After selecting the file, we do not want the set up/loading to take more than 10 seconds

Add new course to database – App should respond in .5 seconds

Modify course database – App should respond in .5 seconds

Get Course details – App should pull up new information in 1 second

Pull courses from classes.iastate.edu – Make minimal API calls, process returned information in 5 seconds

Add new course with unknown class time – We want the algorithm to find new spots within 10 seconds \*NOTE: Start Development on the Algorithm early to ensure time to deal with any difficulties later on\*

### 3.4 PROJECT TIMELINE/SCHEDULE

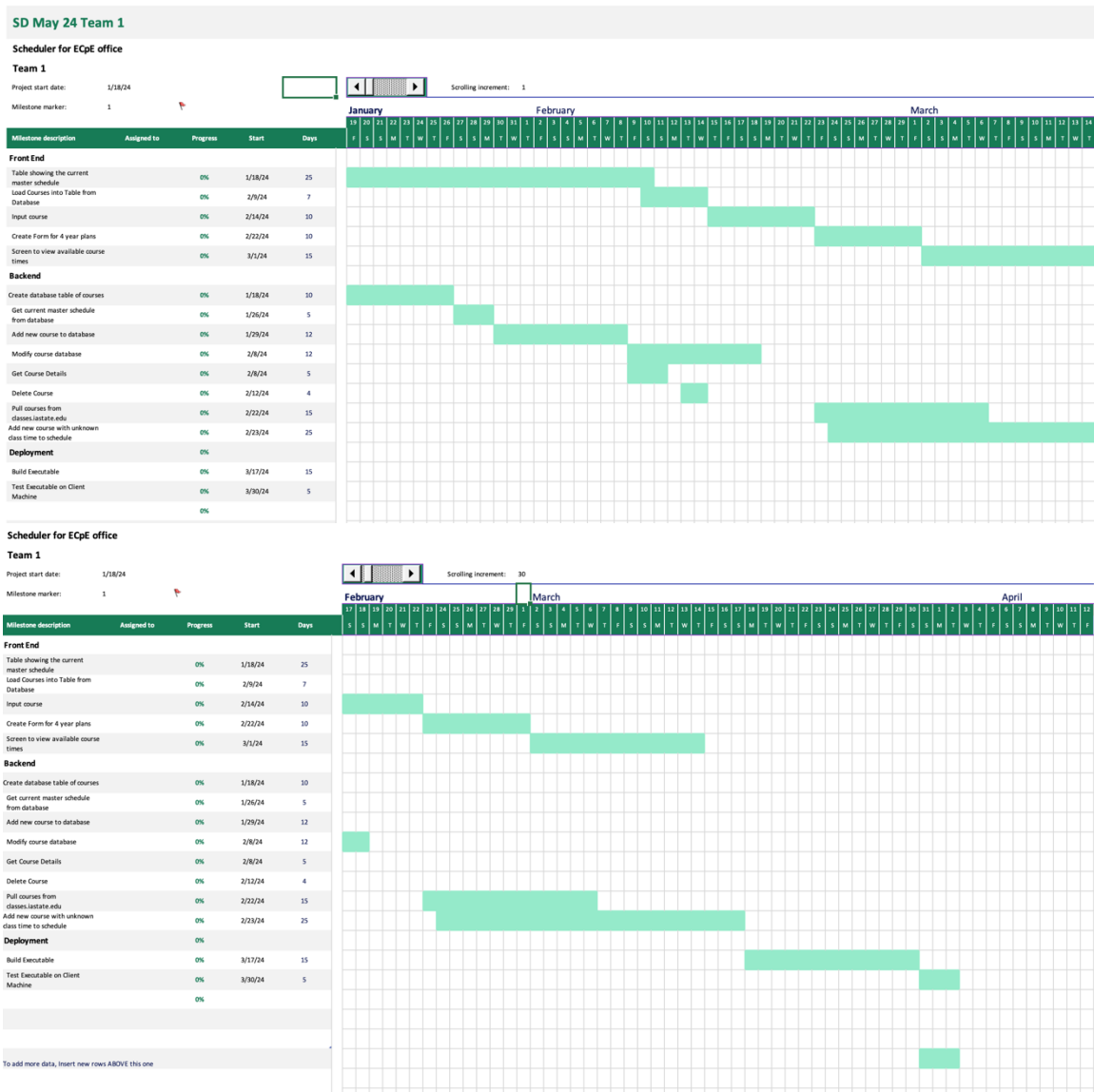


Figure 2: Gantt Chart

### 3.5 RISKS AND RISK MANAGEMENT/MITIGATION

Consider for each task what risks exist (certain performance target may not be met; certain tool may not work as expected) and assign an educated guess of probability for that risk. For any risk factor with a probability exceeding 0.5, develop a risk mitigation plan. Can you eliminate that task and add another task or set of tasks that might cost more? Can you buy something off-the-shelf from the market to achieve that functionality? Can you try an alternative tool, technology, algorithm, or board?

Agile projects can associate risks and risk mitigation with each sprint.

## Front End - UI

1. Table showing the current master schedule for a specific semester (Req 3.2)  
**Risk:** Data representation is unclear or formatted incorrectly.  
**Probability:** 0.4
2. Make the table showing current master schedule read from backend (Req 3.2)  
**Risk:** Backend connectivity issues or data mismatch.  
**Probability:** 0.5  
**Mitigation Plan:** Test connectivity rigorously. Consider adding a system for manual creation
3. Form to input course, course times, and courses or schedules it can't conflict with (Req 1.6)  
**Risk:** Users input incorrect data leading to scheduling conflicts.  
**Probability:** 0.6  
**Mitigation Plan:** Add input validation and error messages to guide users. Consider adding a review stage before data submission.
4. Form to input 4-year plan courses (Req 1.1)  
**Risk:** Mismatch between expected course sequences and actual university offerings.  
**Probability:** 0.4
5. Form to create a new course (Req 1.5)  
**Risk:** Duplication of courses due to different course codes but similar content.  
**Probability:** 0.5  
**Mitigation Plan:** Implement a course search functionality to check for existing courses before creation.
6. Screen to view available course times that a new course could fit into (Req 3.4)  
**Risk:** The algorithm doesn't find the optimal time slot, leading to inefficiencies.  
**Probability:** 0.4

## Front End - Backend Interface

1. GET current master Schedule (Req 1.9)  
**Risk:** Incomplete or corrupted data transfer.  
**Probability:** 0.4
2. POST new course with known class time (including class times, courses it can't conflict with) (Req 1.1, 1.2, 1.8)

- Risk:** Data mismatch or incorrect storage due to invalid input or server error.  
**Probability:** 0.4
3. PUT modify Course (including class times, courses it can't conflict with) (Req 1.1, 1.2, 1.8)  
**Risk:** Overwriting existing data unintentionally.  
**Probability:** 0.5  
**Mitigation Plan:** Always create a backup of existing data before modification. Version control for course details.
  4. GET course details (including class times, courses it can't conflict with) (Req 1.10)  
**Risk:** Data might not reflect the most recent updates or could be incomplete.  
**Probability:** 0.4
  5. DELETE Course and associated data (Req 1.1, 1.2, 1.8)  
**Risk:** Accidental deletion or corruption of associated data.  
**Probability:** 0.6  
**Mitigation Plan:** Add a validation screen before deletion and hold backups for undo's.
  6. [Extension] POST Add new course with unknown class time to current schedule (Req 1.5)  
**Risk:** Conflicting data entries when course time becomes known.  
**Probability:** 0.5  
**Mitigation Plan:** Continually check to see if the now known times conflict.
  7. [Extension] GET possible schedules generated based on requirements (Req 1.3)  
**Risk:** The generated schedules might not be optimal or could miss out on potential configurations.  
**Probability:** 0.5  
**Mitigation Plan:** Perform routine tests and gather user feedback to refine scheduling algorithms over time.
  8. [Extension] POST Set current possible schedule to current master schedule (Req 1.3)  
**Risk:** Overwriting existing master schedule without proper validation.  
**Probability:** 0.6  
**Mitigation Plan:** Provide a preview before overwriting and maintain version history of schedules to roll back if needed.

## Data Access Layer & Database

1. Create a database table of courses  
**Risk:** Data structure constraints might lead to issues with scalability or extensibility.

**Probability:** 0.4

2. Develop a solution to pull from classes.iastate.edu (Req 1.7)

**Risk:** External website structure or data format changes, leading to pulling failures.

**Probability:** 0.4

3. Create an executable that can be run on the client's computer (Req 3.1)

**Risk:** Compatibility or performance issues on varying client systems.

**Probability:** 0.3



### 3.6 PERSONNEL EFFORT REQUIREMENTS

Tasks	Est. Hours	Hours Worked					
		Brian Schomer	Simeon Steward	Lewis Callaway	Carter Evans	Michael Less	Isaiah Ortiola
<b>Front End</b>							
Table showing the current master schedule	20						
Load Courses into Table from Database	15						
Input course	15						
Create Form for 4 year plans	10						
Screen to view available course times	5						
<b>Backend</b>							
Create database table of courses	5						
Get current master schedule from database	15						
Add new course to database	15						
Modify course database	15						
Get Course Details	15						
Delete Course	10						
Pull courses from classes.iastate.edu	15						
Add new course with unknown class time to schedule	15						
<b>Deployment</b>							
Build Executable	5						
Test Executable on Client Machine	15						
	190						

Figure 3: Effort Chart

The table above shows the estimated hours for each task (as seen in the Gantt Chart) and the rightmost portion of the table will help our group members log the time they've put into each task.

### 3.7 OTHER RESOURCE REQUIREMENTS

Since our project is entirely a software project, there should be no need for any types of additional hardware or parts needed for our project. We shouldn't need any other physical materials to create our project, however one thing that we might require is some kind of server to allow for the application to be accessed at any time. This is only if we are unable to create a desktop application which is unlikely. Besides a possible server, there should be no other requirements needed for our project.

## 4 Design

### 4.1 DESIGN CONTENT

This section covers the design for the Class Scheduler's user interface and backend. The design will give an overview of the different parts of the overall software executable and describes the

components that will comprise the program. It also describes the broader context and design decisions that were made.

#### 4.2 DESIGN COMPLEXITY

Provide evidence that your project is of sufficient technical complexity. Use the following metric or argue for one of your own. Justify your statements (e.g., list the components/subsystems and describe the applicable scientific, mathematical, or engineering principles)

1. The design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, or engineering principles
  - A. User Interface (UI): The development of a user-friendly desktop application requires expertise in user experience (UX) design, including layout design, accessibility, and information presentation principles.
  - B. Data Management: Managing class schedules, course rules, and student data necessitates robust data handling principles, including database design and optimization.
  - C. Scheduling Algorithm: The core of the application is the scheduling algorithm, which must incorporate mathematical and computational principles to find optimal class times, considering various constraints and preferences. This may include graph theory, constraint satisfaction, and optimization techniques.
  - D. Integration with External Data Sources: If we can pull data from external servers like `classes.iastate.edu`, integrating and synchronizing this data requires knowledge of web APIs and data parsing.
  - E. Software Framework: Using Electron.js as the framework adds another layer of complexity. Understanding the framework's architecture, APIs, and best practices is crucial for a successful implementation.
  
2. The problem scope contains multiple challenging requirements that match or exceed current solutions or industry standards.
  - A. Current Process: We aim to replace a manual class scheduling process with an automated system. This will help with making the process faster and easier for new office staff.
  - B. End User: Intending to make the application easy to install for non-technical users adds another layer of complexity.
  - C. Scheduling Algorithm: The need to create a new course with customizable rules for scheduling, including avoiding conflicts and adhering to specific semester schedules, requires additional considerations that a generic scheduling algorithm may not consider.

#### 4.3 MODERN ENGINEERING TOOLS

WebStorm IDE: WebStorm IDE is a powerful integrated development environment that will be used for coding and development. Its role included code editing and debugging.

Gitlab: Gitlab will be used for version control and continuous integration. It will play a pivotal role in managing the project's source code. We will collaborate, track changes, and merge code seamlessly using Gitlab.

.js: Electron.js has been chosen as the framework for building the desktop application. Its role was to enable the development of desktop applications using web technologies (HTML, CSS, and JavaScript). It also has the functionality to work independently of any servers which is a requirement for our desktop-based program.

DBMS: SQLite will be used as a database platform to store class information, preferences, and rules. It will interface with Electron.js using an API.

#### 4.4 DESIGN CONTEXT

Describe the broader context in which your design problem is situated. What communities are you designing for? What communities are affected by your design? What societal needs does your project address?

Our problem is solving a very specific need for the ECpE office, however if successful, it could easily be extended to other departments across Iowa State. Our project is addressing a need to assist in making the scheduling of classes easier for ECpE office staff.

List relevant considerations related to your project in each of the following areas:

Area	Description	Examples
Public health, safety, and welfare	While our project doesn't necessarily affect public health or safety, it does work to make an existing process less stressful for ECpE staff.	The ECpE office staff can schedule new courses quicker and easier than before. Professors can know a potential time for a new course quicker since the office staff can find an available time with less manual calculation.
Global, cultural, and social	The software does not offer many opportunities to reflect the values, practices, and aims of the cultural groups it affects, but will attempt to be aligned to the practices and aims of the ECpE department whenever possible.	Not super applicable to our project, but we ensured that the software aligns with the current standards and processes for scheduling classes so as many students have access to courses as possible.
Environmental	The project is purely project, so there is negligible environmental impact however it may provide a small positive impact.	Since this is a purely software project, it has a negligible impact. It may save a small amount of paper if the department printed the existing schedule out to find available times manually.

Economic	Our project will provide an economic benefit because it will help streamline office processes and save time for the ECpE office staff.	This project will likely save the ECpE office staff time in scheduling classes that they can rededicate toward assisting students which in turn could help Iowa State retain more students. The project has no recurring costs since it is locally run.
----------	----------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#### 4.5 PRIOR WORK/SOLUTIONS

Iowa State has a “schedule planner” tool at [classes.iastate.edu](http://classes.iastate.edu) and AccessPlus has similar functionality. However, these tools are aimed at a student being able to create a schedule that fits within their requirements. These tools wouldn’t work for our application. One reason is the workflow for planning a course. The aforementioned tools both help students schedule classes once they are entered in the master schedule. Our project helps the ECpE determine where new courses should be scheduled in the master schedule. Our project is not intended to generate student schedules. Additionally, our tool must consider different rules than the aforementioned schedulers since we are not attempting to create a schedule for a student, but rather schedule a course. For example, a prerequisite would be enforced on Access Plus, but our project can schedule a new course at the same time as a prerequisite.

#### 4.6 DESIGN DECISIONS

**No backend server:** We want to make the project as maintainable in the future as possible. Not requiring a server to be operational for the code to run would assist with that. The client requested a desktop application. Additionally, only one person will be using this software so there is no need for concurrent access. The project may pull data from the [classes.iastate.edu](http://classes.iastate.edu) API, but it will also be designed in a manner that can allow for manual class entry if the API changes or is unavailable. All the data that needs to persist in the project can be saved as a local file and doesn’t need to be synced on the cloud.

**Choosing Electron:** Our group has selected Electron.js as the framework to build a desktop application. We selected this platform for multiple reasons. One reason is that it is cross-platform so it can be used on both Mac and PC which is a convenient feature since the University uses both platforms. It also allows us to create a user interface in a web framework such as React, that most of our team members have some familiarity with unlike a Desktop UI framework like JavaFX. For a backend of NodeJS, our team members a little experience, however it is simpler to learn and frequently used in industry. Electron does have slightly worse performance than a native application since it runs on top of a variant of Chromium, however our program will not use many resources so this should not be a problem.

**Nodejs & Typescript Backend:** For the backend, we chose to go with NodeJS and Typescript as it is commonly and successfully used with electron, and we are confident that it will suit our needs as a backend framework. Typescript is a familiar language, and for a project of this size the types are appreciated over vanilla JavaScript.

**SQLite database:** Our group selected SQLite as a database for our project. We picked this instead of using a static file like CSV since users will be much less likely to attempt to open the file and corrupt it. There are many SQLite libraries (potentially an ORM) that would allow us to interface with our backend code which will allow for more streamlined development. We chose SQLite over a more extensive database like MySQL is because we don't require as much overhead to install the database. To run MySQL, the end user would need to be running a MySQL server which would complicate the installation. For SQLite, we can have a single file that our program interacts with. SQLite doesn't allow for simultaneous users to connect to the database, but that is not needed for our application.

#### 4.7.1 Design 0 (Initial Design)

##### Design Visual and Description

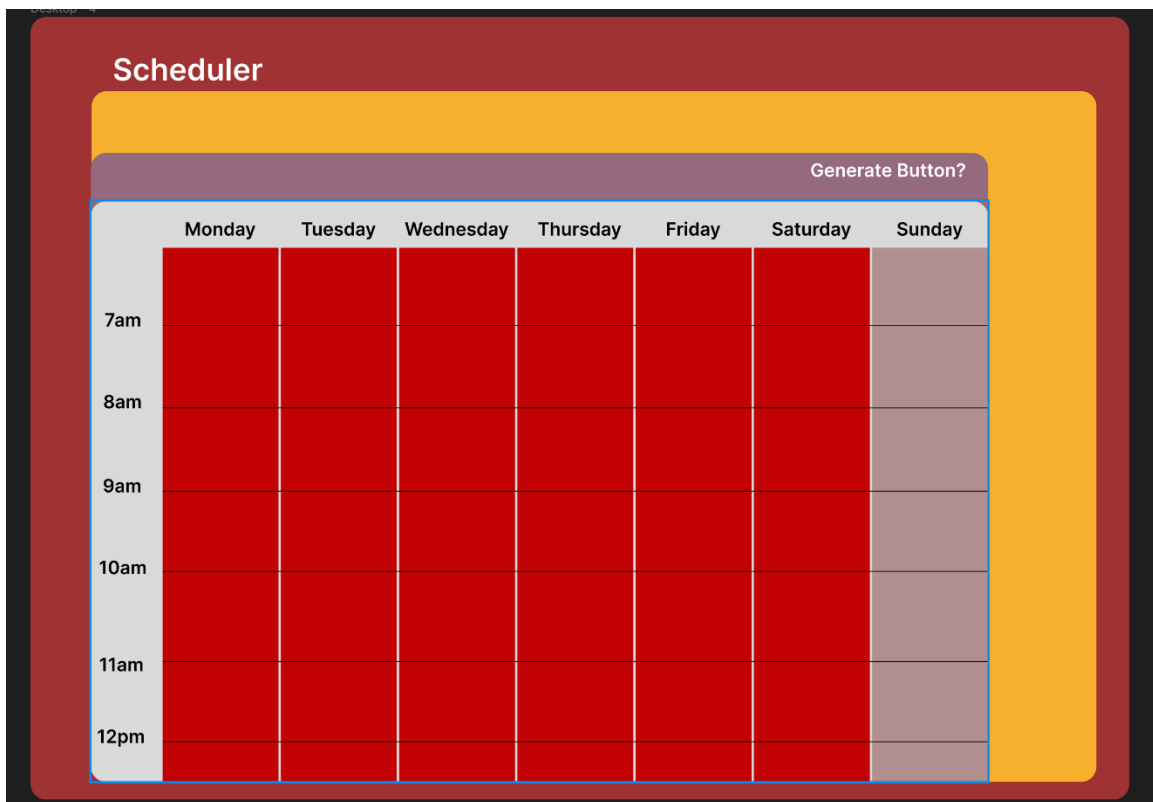


Figure 4: Prototype Design

This design is a rough idea for how our desktop app may look. Since this is Design 0, this is a very rough idea of what it will look like. This design includes a clean UI that is easy on the eyes and includes the ISU colors since it will be used by the ECpE department here on campus. The UI includes a clickable weekly schedule and possible button to randomly generate a schedule for the user.

##### Functionality

This design is intended to let the user insert the desired courses into the weekly schedule where they choose. However, there is also a button that the user can use to generate a schedule which

would be far less time consuming and would automatically generate a schedule with the desired classes that the user can then use for their scheduled courses for the semester. We believe that this design does a decent job satisfying the user's needs but could be improved.

#### 4.7.2 Design 1 (Design Iteration)

Include another most matured design iteration details. Describe what led to this iteration and what are the major changes that were needed in Design o.

New Class Setup:

Class Name  
CPR E 444x

Class Days  
Monday, Wednesday, Friday

Class Length  
50 Min

Schedule Around  
CPR E 430, CPR E Juniors

Add Cancel

Figure 5: Add new class popup

Fitness of the new course:

	Monday	Tuesday	Wednesday	Thursday	Friday
7am	70%	80%	20%	70%	70%
8am	70%	70%	80%	60%	50%
9am	70%	50%	20%	80%	50%
10am	70%	80%	70%	70%	20%
11am	70%	50%	80%	70%	20%
12pm	60%	50%	50%	20%	20%
1pm	50%	70%	70%	60%	80%
2pm	60%	70%	60%	20%	20%
3pm	70%	70%	70%	70%	70%
4pm	80%	80%	80%	80%	80%
5pm					
6pm					

Figure 6: How many students can take each course

Clicking on a timeslot shows the potential conflicts.

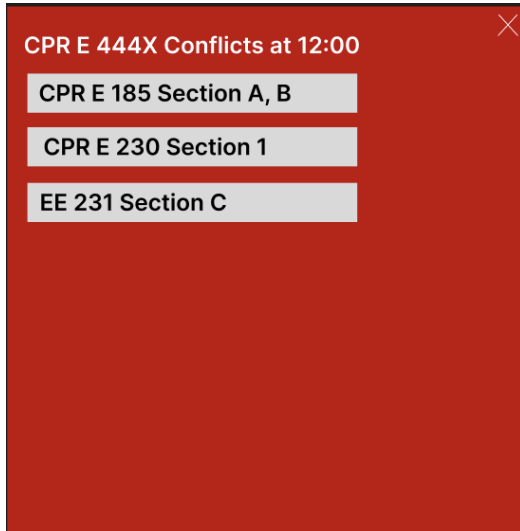


Figure 7: Conflict screen

Some changes we made from Design 0 is that we removed the weekend slots since classes will not be scheduled on the weekend. We also changed the time that classes can be scheduled at since classes start as early as 7:45 am, and end as late as 6:20 pm. Another thing that we added was a way to test multiple classes at once with the addition of the '+' button. We created an input box that is used to create a new course that will be evaluated in the schedule. Then the output will provide the fitness of each time slot for a new course. Finally, clicking on a time slot will tell you the class sections that would be a conflict.

#### 4.7.3 Design 2 (Implemented Design / 492 Changes)

Our final design did not vary much from the proposed design. There were minor differences in the user interface due to the UI libraries that were used. The time grid was nearly identical as was proposed in earlier designs. The technical implementation is identical to what was designed in 491. We did, however, use additional libraries like the Material UI and Material UI Table to help expedite development.

Scheduler	Classes	Cohort	Schedule	Import
CPRE 2222	EXPORT TO CSV		CLASS SETTINGS	
	Monday	Wednesday	Friday	
7:45 am	81.8	81.8	81.8	
8:50 am	100	100	100	
9:55 am	100	100	100	
11:00 am	100	100	100	
12:05 pm	72.7	72.7	22.7	
1:10 pm	72.7	72.7	22.7	
2:15 pm	72.7	72.7	72.7	
3:20 pm	100	100	100	
4:25 pm	100	50	100	
5:30 pm	100	50	100	

Figure 8: Conflict grid screen

To begin, a user would create the courses and a cohort grouping of courses that will need to be scheduled around. Then, once that is done, a user opens the class settings for a new course and inputs the potential course name, day and time sequence, classes to avoid, and cohort to avoid.

### Class Settings

Potential Class Name

Select day and time sequence  
MWF - 50 Min ▾

Select classes to avoid:  
▾  
ADD CLASS

Select Cohort  
▾  
ADD COHORT

Avoiding:  
E E 185

MATH 165

[CANCEL](#)   [SUBMIT](#)

Figure 9: Conflict grid screen



Our previous designs did not extensively address the class, cohort, and import screen. Our team used a material UI table library to display the courses, cohorts, and imports in an easy to navigate view.

DEPT	Class #	Section	Class Times	Actions
MATH	165	1	Monday 07:45-08:35 Wednesday 07:45-08:35 Friday 07:45-08:35 Tuesday 08:00-08:50	
MATH	165	2	Monday 07:45-08:35 Wednesday 07:45-08:35 Friday 07:45-08:35 Tuesday 09:00-09:50	
MATH	165	3	Monday 07:45-08:35 Wednesday 07:45-08:35 Friday 07:45-08:35 Tuesday 10:00-10:50	
MATH	165	4	Monday 14:15-15:05 Wednesday 14:15-15:05 Friday 14:15-15:05 Tuesday 14:10-15:00	
MATH	165	5	Monday 14:15-15:05 Wednesday 14:15-15:05 Friday 14:15-15:05 Tuesday 14:10-15:00	

Figure 10: Class table screen

We additionally added a CSV export function in the final product after being recommend by our advisor. The CSV export function can be triggered by the “Export to CSV” button on the schedule time grid.

Tab Title	Day	Time	Avallability
CPRE 2222	Monday	7:45 am	81.8
CPRE 2222	Monday	8:50 am	100
CPRE 2222	Monday	9:55 am	100
CPRE 2222	Monday	11:00 am	100
CPRE 2222	Monday	12:05 pm	72.7
CPRE 2222	Monday	1:10 pm	72.7
CPRE 2222	Monday	2:15 pm	72.7
CPRE 2222	Monday	3:20 pm	100
CPRE 2222	Monday	4:25 pm	0
CPRE 2222	Monday	5:30 pm	100
CPRE 2222	Wednesday	7:45 am	81.8
CPRE 2222	Wednesday	8:50 am	100
CPRE 2222	Wednesday	9:55 am	100
CPRE 2222	Wednesday	11:00 am	100
CPRE 2222	Wednesday	12:05 pm	72.7

Figure 11: CSV Example

Additional considerations had to be made on the Import section. The departments are pulled from the Iowa State classes.iastate.edu API because Iowa State will likely add departments over time. Additionally, an import setting had to be added that allows a user to select the semester that the classes are pulled from.

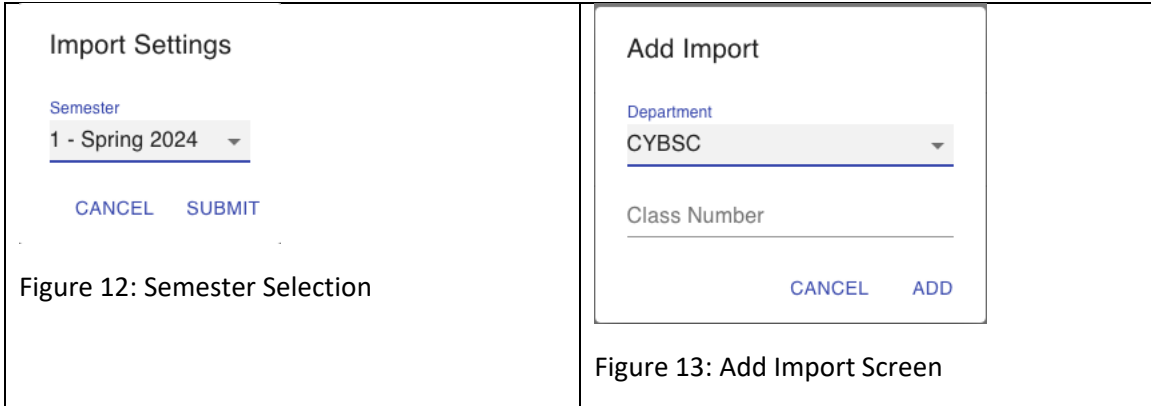


Figure 12: Semester Selection

Figure 13: Add Import Screen

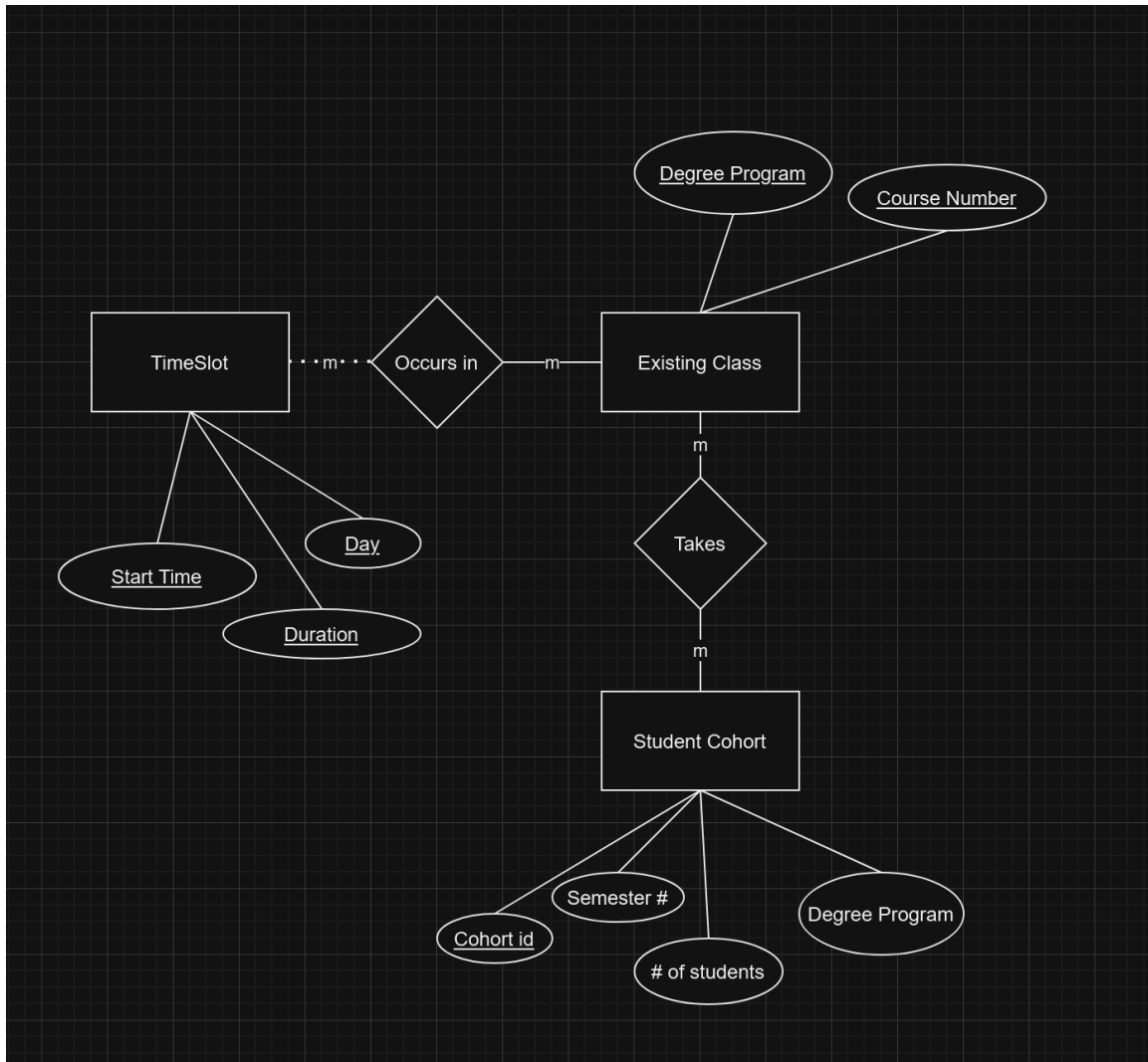
#### 4.7 SECURITY CONCERNS

Our program does not store or interact with any private student data or use logins for different users so there is no need for encryption or special security precautions. The project uses publicly available data so there is no need to ensure additional precautions. A stretch goal for the project could be putting in place measures to track the software supply chain security. We could have measures to scan the open-source packages and how they are used to ensure they will not harm the client computer. However, this was deemed low priority by our team and not implemented in this iteration of the project.

#### 4.8 APPENDIX A – TIME SLOTS

50 Minute Classes	75-Minute Classes
7:45am - 8:35am	8:00am - 9:15am
8:50am - 9:40am	9:30am - 10:45am
9:55am - 10:45am	11:00am - 12:15pm
11:00am - 11:50am	12:40pm - 1:55pm
12:05pm - 12:55pm	2:10pm - 3:25pm
1:10pm - 2:00pm	3:40pm - 4:45pm
2:15pm - 3:05pm	4:10pm - 5:25pm
3:20pm - 4:10pm	
4:25pm - 5:15pm	
5:30pm - 6:20pm	

## 4.9 APPENDIX B – DATABASE DESIGN DIAGRAM



## 5 Testing

### 5.1 UNIT TESTING

Unit testing will be used to test critical functions within our code. For functions that could easily be messed up, a unit test will be put into place to ensure that other code is not unintentionally broken. While we don't know every specific function that will be in place, we have identified some known functions this will be good for:

- The scheduling algorithm functions – ensure parts of the scheduling algorithm produces the desired input and output.
- Test the input of core classes for different majors to ensure that the course title, time requirement, and available spots are correctly stored in the database (Req 2.2.1.1)
- Verify that the application handles different time formats correctly (Req 2.2.1.1)

- Test the creation of a new course to ensure that it is correctly added to the database (Req 2.2.1.2)
- Verify a rule is correctly enforced during scheduling (Req. 2.2.1.6)
- Test the handling of cross listed courses to ensure they are not treated separately (Req. 2.2.1.8)

For a tool, we will likely use Jest as it is the most popular unit testing platform for JavaScript programs. Hopefully our team can enable it to run on a CI/CD pipeline.

## 5.2 INTERFACE TESTING

Our program is not a client and server communication-based application, so there is not a need to ensure that the API is matching a standard protocol. The only API we could check would be ensuring that classes.iastate.edu is accessible and matches the previously utilized data standard.

## 5.3 INTEGRATION TESTING

A potential integration test for our project would be ensuring that a standardized CSV format is read properly and saved to the database. This would ensure that the data is not harmed in the process and verify it can be accurately retrieved. Another integration test could be the integration between the user interface and the core application logic. Verify that user actions in the UI (e.g., creating a new course, setting rules) correctly trigger the corresponding functions in the backend.

We plan to use [Selenium and Jest](#) in order to test that the changes are accurately reflected.

## 5.4 SYSTEM TESTING

We plan on creating groups of multiple unit tests and then running them together to create a test to ensure full system functionality. There then can be integration tests that create sample user inputs and then test the output of the program to ensure that the desired output is produced.

The CI/CD pipeline on Gitlab will be used to ensure that the code is working as intended. Upon pushing code to Gitlab, all code will be tested to verify that it passes the implemented tests. This way we can confirm that code modifications doesn't negatively impact the results. The individual unit tests align to specific requirements so a group of smaller units or integration tests will work to ensure requirements are being met.

## 5.5 REGRESSION TESTING

The unit tests will be run when code is being built to determine if changes have broken something else (related or unrelated). The requirements are being used to develop the tests, so a failure of a test will mean a violation of a requirement. We will use internal code interfaces to provide some separation of concerns which will help later in our development when we wish to add or change code but do not want to break our existing functionality. This also means that our job of mocking and testing will be easier later and our individual components will not affect each other.

## 5.6 ACCEPTANCE TESTING

We plan to show the client working demonstrations in the development process. Even though specific parts of the code will not be functioning, we can determine if the specific parts of the project under test are aligning to what the client desires. Since our project is just used by a specific client, we can ensure it is exactly what they want.

## 5.7 SECURITY TESTING

Security testing will not be applicable for this project since we are not storing any private information. Because everything we will be working on for the class scheduler is already available to the public, we won't need to worry about any type of security testing.

## 5.8 RESULTS

Our team was able to successfully create and pass unit tests for the business logic that calculates the availability for courses. These tests were helpful in ensuring that additional algorithm changes did not break previously developed code. Additionally, tests were developed for the backend endpoints for the availability calculations and successfully passed them as well. Our team spent an extensive amount of time trying to get unit tests created for the frontend but was unfortunately unable to get the tests to compile correctly.

The acceptance testing was successful. We met with the clients and asked them to provide a difficult scenario and other examples to run through the program. Tina and Vicky brought a complex scenario regarding scheduling a CPRE lab and other requirements. The program was able to meet and exceed their requirements. The clients did not have any other requirements that needed to be met after the meeting. If the project were bound by a formal contract, this would be deemed a complete performance.

# 6 Implementation

## 6.1 DETAILED DESIGN

Our team developed a class scheduler program to follow the requirements given by the client. We decided to utilize React as a frontend library and NodeJS as a backend library to power the program. It is being run on a desktop currently with both the server and frontend executed from a single command, however this could very easily be moved to a server and then be deployed as a web application. Additionally, ElectronJS was utilized in order to give our UI a wrapper that makes it appear as a desktop application and not a website.

The frontend was developed to get input from the user, call the backend endpoints and then display results in an easy-to-view manner. Additionally, table libraries were utilized to allow for pagination and search of classes, cohorts, and imports. The time grid / schedule screen accepts parameters from a user and then sends the parameters to the backend for calculation.

The backend has an algorithm to take the parameters provided by a user and then return the availability of each time slot and the class conflicts. For each conflict, a conflict percentage is also included. The backend has endpoints and logic to interact with the SQLite database and call the conflict logic. The conflict algorithm works by iterating over the class(es) and cohort(s) that the request was made for and for each class assume that the students are divided by sections equally. This allows us to estimate how many students will have a conflict during each timeslot.

## 6.2 DESCRIPTION OF FUNCTIONALITY

A user begins by first creating the classes that they will want to schedule around. This can be done in two ways. One method is to manually create it in the Classes section by filling out the class department, section, meeting times, and dates of the week. The other option is by going to the

Import section and then adding the class and then clicking on “Refresh.” This will pull the latest sections and then populate the Classes table reducing the amount of time a user needs to spend inputting all the class sections and meeting times.

Next, a user can create a Cohort. A Cohort in our program is a grouping of courses based on department and semester. The classes inside of a Cohort are based off a semester of the four-year plan. An example of a Cohort would be electrical engineering semester one or computer engineering semester seven. Cohorts are used to group classes so a new potential course can be scheduled around a target group of students like third semester electrical engineering students.

Finally, the best fit for the new course can be determined. A user begins by clicking on the Schedule tab and then on Class Settings. From there, a potential class name can be entered. A day and time sequence needs to be selected (Monday/Wednesday/Friday or Tuesday/Thursday). Then, the classes to avoid need to be selected as well as any cohorts to avoid. When submit is pressed, a calculation is performed, and each time slot has a percentage value on it as well as a color. If you click on a time slot, it will tell what conflicts there are as well as a percentage availability for each.

### 6.3 NOTES ON IMPLEMENTATION

Much of the code was written between the start of the second semester of the project and spring break. Due to some team members starting jobs or being involved in other classes, we agreed to frontload development. The second half of the semester was spent on implementing additional features like the CSV export and persistent time grid as well as expanding testing. The final part of the semester was spent on documentation and preparing for the presentation.

## 7 Professionalism

This discussion is with respect to the paper titled “Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment”, *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

### 7.1 AREAS OF RESPONSIBILITY

If we take the Software Engineering code of ethics and compare it to Table 1 from the paper discussed above, then we can see that many of the principles seen in the SE code of ethics are able to relate to the items seen in the table. For example the first item in the table is Work Competence which is defined as “Perform work of high quality, integrity, timeliness, and professional competence.”, this can be closely compared to Principle 1: Product from the SE code of ethics, which summarily states that software engineers must assure that the product they work on is both useful and acceptable to the public, client, and/or user and should be completed on time and with little to no error.

### 7.2 PROJECT SPECIFIC PROFESSIONAL RESPONSIBILITY AREAS

The first item of the table Work Competence does apply to our profession context, especially because we are the second group to have touched this project, but our predecessors were not able

to deliver a completed product, thus adding more pressure on us to deliver a quality software product to our client. Our team is performing quite well in being able to deliver quality work, so far only in reports, on a timely basis and it's expected to continue going forward in the second semester.

### 7.3 MOST APPLICABLE PROFESSIONAL RESPONSIBILITY AREA

The most applicable professional responsibility area for our team would be the SE code of ethics since our project is exclusively a software project.

#### 7.4.1 Team Contract

Team Members:

- 1) Brian Schomer 2) Lewis Callaway
- 3) Simeon Steward 4) Isaiah Ortiola
- 5) Michael Less 6) Carter Everts

Team Procedures

Day, time, and location (face-to-face or virtual) for regular team meetings:

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
3. Decision-making policy (e.g., consensus, majority vote):
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
3. Expected level of communication with other team members:
4. Expected level of commitment to team decisions and tasks:

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
2. Strategies for supporting and guiding the work of all team members:
3. Strategies for recognizing the contributions of all team members:

### Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
2. Strategies for encouraging and support contributions and ideas from all team members:
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

### Goal-Setting, Planning, and Execution

1. Team goals for this semester:
2. Strategies for planning and assigning individual and team work:
3. Strategies for keeping on task:

### Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
2. What will your team do if the infractions continue?

\*\*\*\*\*

- a) I participated in formulating the standards, roles, and procedures as stated in this contract.
- b) I understand that I am obligated to abide by these terms and conditions.
- c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

- 1) Brian Schomer DATE 12-3-23
- 2) Lewis Callaway DATE 12-3-23
- 3) Simeon Steward DATE 12-3-23
- 4) Isaiah Ortiola DATE 12-3-23
- 5) Michael Less DATE 12-3-23
- 6) Carter Everts DATE 12-3-23



## 8 Conclusions

### 8.1 REVIEW PROGRESS

Our team was able to build a software program that met our client's expectations and was able to handle the scenarios that were given to us by our client. Our team successfully worked together to develop an attractive user interface that runs on a desktop. A backend was developed that could calculate an availability for a timeslot given constraints. The frontend was successfully integrated with the backend. The project was a great experience for group members to be able to take client requirements gathered in a meeting and transform it into working software. Our implemented solution did not vary much from the design that was created the first semester.

### 8.2 VALUE PROVIDED BY DESIGN

The value provided by our design is the ability for the ECpE office to be able to now schedule new classes or class sections in a more streamlined manner. The ECpE office can now provide professors better data regarding when a new course would have the highest likelihood of successful enrollment. Additionally, the program can theoretically reduce the amount of the ECpE office staff have to spend doing manual scheduling work and increase their time supporting students directly.

### 8.3 FUTURE STEPS

The program should function without maintenance for a reasonable time period. The only limiting factor could be NodeJS packages becoming end-of-life. The project was developed while Iowa State is undergoing an ERP transition to Workday from ADIN so there is some uncertainty regarding the classes.iastate.edu endpoint that was used for the import. Our team evaluated other options, but there was no other suitable option for a publicly available classes API at the given time. Therefore, if classes.iastate.edu is eliminated in favor of a Workday solution, this program would need new code to integrate with a new API. The program was developed as modularly as possible in order to assist with future development. Additionally, it was developed so the software doesn't necessarily rely on the endpoint. A user could manually enter class information and still use the program albeit less efficiently.

## 9 Appendix 1 – Install/Operation Manual

### 9.1 INSTALLATION

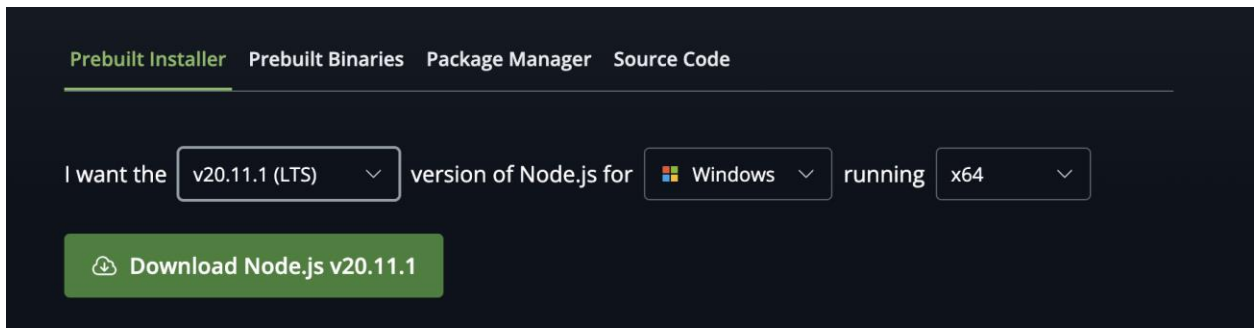
Download and unzip the project file that we provide to somewhere like the documents folder. When we reference files, it will come from this folder.

1. Download Node.js – this is the framework used to power the app.

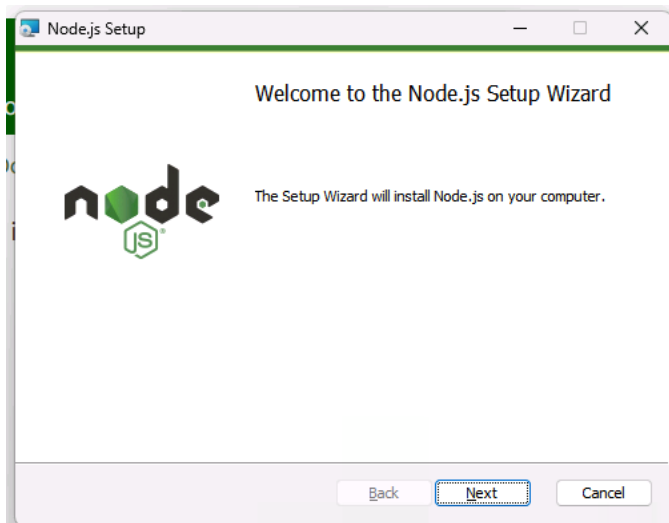
<https://nodejs.org/en/download/>

Set these settings to run on Windows and then click on “Download Node.js v20.11.1”

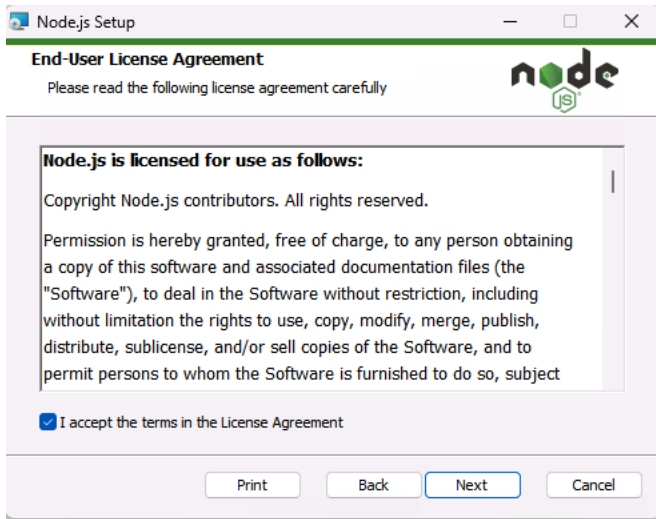
*\*It should run on newer versions in the future, but unfortunately, we can't predict the future*



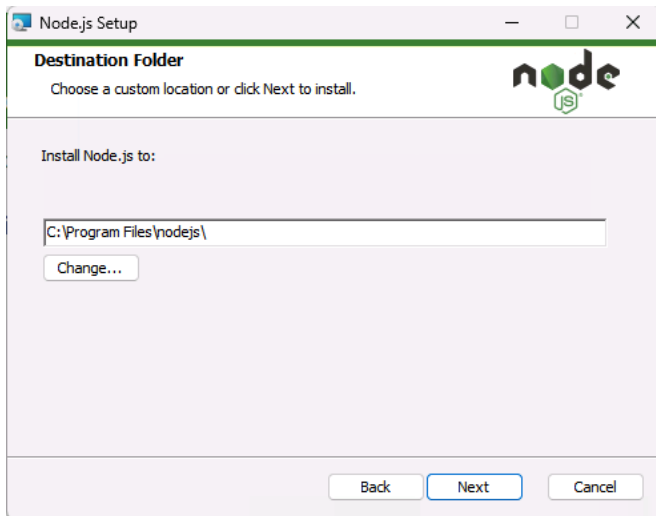
2. Run the executable to get the setup wizard.



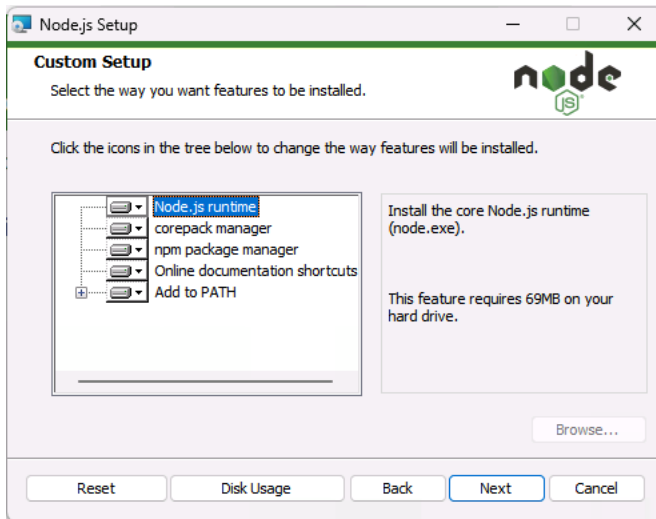
3. Accept the terms.



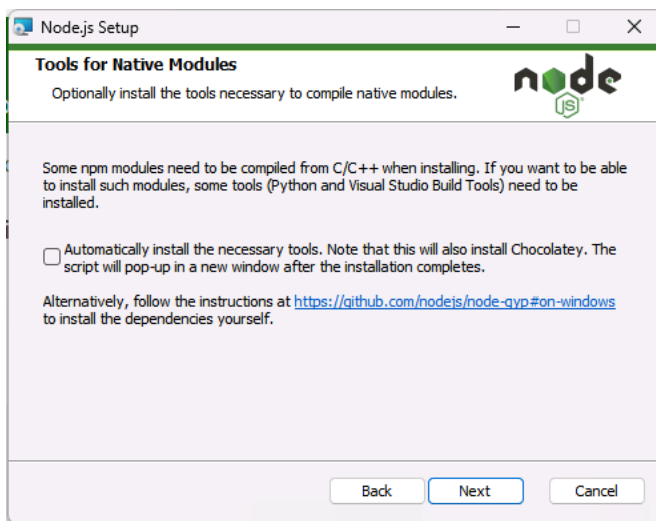
4. Default folder location is fine.



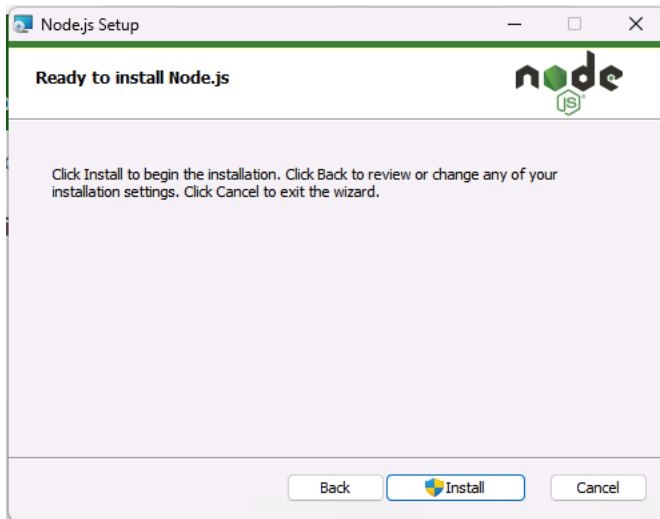
5. Leave this as default



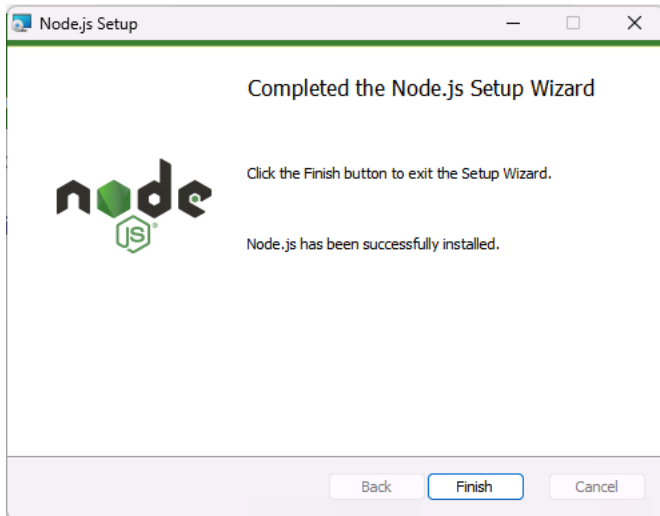
6. This doesn't need to be selected.



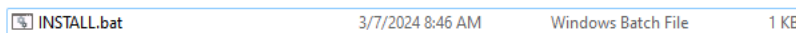
7. Click "Install"



8. Wait for it to install then click “Finish”



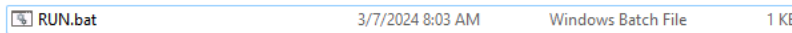
9. In the project folder, double click on INSTALL.bat. Leave this window open while it installs the required packages.



It will look like this:

```
ted
warning react-scripts > workbox-webpack-plugin > workbox-build > workbox-google-analytics > workbox-strategies@6.6.1: this package has been deprecated
warning react-scripts > workbox-webpack-plugin > workbox-build > workbox-recipes > workbox-strategies@6.6.1: this package has been deprecated
warning react-scripts > workbox-webpack-plugin > workbox-build > workbox-precaching > workbox-strategies@6.6.1: this package has been deprecated
warning react-scripts > workbox-webpack-plugin > workbox-build > workbox-strategies > workbox-core@6.6.1: this package has been deprecated
warning react-scripts > workbox-webpack-plugin > workbox-build > workbox-sw@6.6.1: this package has been deprecated
warning react-scripts > jest > @jest/core > jest-config > jest-environment-jsdom > jsdom > data-urijs > abab@2.0.6: Use your platform's native atob() and btoa() methods instead
warning react-scripts > jest > @jest/core > jest-config > jest-environment-jsdom > jsdom > domexception@2.0.1: Use your platform's native DOMException instead
warning react-scripts > jest > @jest/core > jest-config > jest-environment-jsdom > jsdom > w3c-hr-time@1.0.2: Use your platform's native performance.now() and performance.timeOrigin.
warning react-scripts > @svgr/webpack > @svgr/plugin-svgo > svgo > stable@0.1.8: Modern JS already guarantees Array#sort() is a stable sort, so this library is deprecated. See the compatibility table on MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort#browser_compatibility
warning react-scripts > css-minimizer-webpack-plugin > csso > csso-preset-default > postcss-svgo > svgo > stable@0.1.8: Modern JS already guarantees Array#sort() is a stable sort, so this library is deprecated. See the compatibility table on MDN: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort#browser_compatibility
warning react-scripts > workbox-webpack-plugin > workbox-build > @rollup/plugin-replace > magic-string > sourcemap-codec@1.4.8: Please use @ridgewell/sourcemap-codec instead
warning sqlite3 > node-gyp > make-fetch-happen > cacache > @npmcli/move-file@1.1.2: This functionality has been moved to @npmcli/fs
warning @babel/plugin-proposal-private-property-in-object@7.21.11: This proposal has been merged to the ECMAScript standard and thus this plugin is no longer maintained. Please use @babel/plugin-transform-private-property-in-object instead.
resolve-alpn@1.0.0
```

10. Now click on the “RUN.bat”



And leave the terminal window open (you can minimize it) as long as you want the program to run.

```
app - electronmon
[0] Compiled with warnings.
[0]
[0] Failed to parse source map from 'C:\Users\User\Documents\SeniorDesign\sdmay24-01\node_modules\react-double-scrollbar\dist\DoubleScrollbar.js.map' file: Error: ENOENT: no such file or directory, open 'C:\Users\User\Documents\SeniorDesign\sdmay24-01\node_modules\react-double-scrollbar\dist\DoubleScrollbar.js.map'
[0]
[0] Search for the keywords to learn more about each warning.
[0] To ignore, add // eslint-disable-next-line to the line before.
[0]
[0] WARNING in ./node_modules/react-double-scrollbar/dist/DoubleScrollbar.js
[0] Module Warning (from ./node_modules/source-map-loader/dist/cjs.js):
[0] Failed to parse source map from 'C:\Users\User\Documents\SeniorDesign\sdmay24-01\node_modules\react-double-scrollbar\dist\DoubleScrollbar.js.map' file: Error: ENOENT: no such file or directory, open 'C:\Users\User\Documents\SeniorDesign\sdmay24-01\node_modules\react-double-scrollbar\dist\DoubleScrollbar.js.map'
[0]
[0] webpack compiled with 1 warning
[0] No issues found.
[2] Executing (default): SELECT `id`, `title`, `dayTimeSequence`, `avoidClasses`, `avoidCohorts`, `availabilityData`, `createdAt`, `updatedAt` FROM `Schedule` AS `Schedule`;
[2] Executing (default): SELECT `classDepartment`, `classNumber`, MIN(`id`) AS `id` FROM `classes` AS `Class` GROUP BY `classDepartment`, `classNumber` ORDER BY `Class`.`classDepartment` ASC, `Class`.`classNumber` ASC;
[2] Executing (default): SELECT `id`, `program`, `semesterNumber`, `classes`, `createdAt`, `updatedAt` FROM `cohorts` AS `Cohort`;
[2] uniqueClasses
[2] Executing (default): SELECT `classDepartment`, `classNumber`, MIN(`id`) AS `id` FROM `classes` AS `Class` GROUP BY `classDepartment`, `classNumber` ORDER BY `Class`.`classDepartment` ASC, `Class`.`classNumber` ASC;
[2] uniqueClasses
[2] Executing (default): SELECT `id`, `program`, `semesterNumber`, `classes`, `createdAt`, `updatedAt` FROM `cohorts` AS `Cohort`;
```

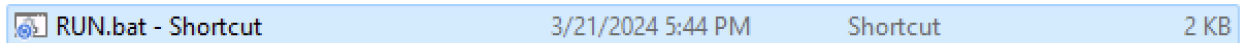
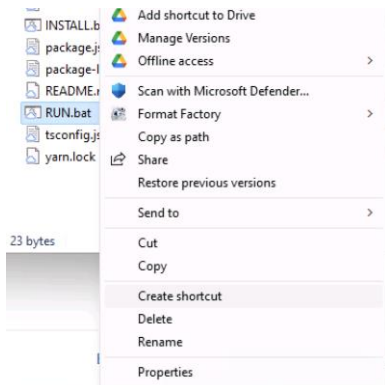
*Terminal: Leave this window open*

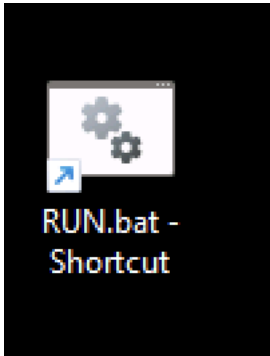
The app will now run!

The screenshot shows a web application window titled "React App" with a menu bar (File, Edit, View, Window, Help) and a navigation bar with buttons for "Scheduler", "Classes", "Cohort", "Schedule", and "Import". Below the navigation bar is a tab labeled "Tab 1" and a "CLASS SETTINGS" button. The main content is a table with columns for "Monday", "Wednesday", and "Friday", and rows for various times from 7:45 am to 5:30 pm. The table cells contain numerical values representing class settings.

	Monday	Wednesday	Friday
7:45 am	100	100	100
8:50 am	85.71	85.71	85.71
9:55 am	71.42	71.42	71.42
11:00 am	85.71	85.71	85.71
12:05 pm	100	94.74	100
1:10 pm	85.71	80.45	85.71
2:15 pm	85.71	80.45	85.71
3:20 pm	85.71	80.45	85.71
4:25 pm	100	94.74	100
5:30 pm	100	94.74	100


When done, close out program window and terminal window. You just need to click on "RUN.bat" every time you want to run the program. You can create a shortcut to this file and then move it to the desktop if you don't want to move the folder.





Finally, if you want to maintain separate databases for each semester, rename the database.sqlite file to something else to make a copy. An example would be “fall.sqlite”

Then the program will create a new database when launched if no database.sqlite exists. If you want to reuse an existing database, then just rename the existing database to database.sqlite and place it in this folder. So you could have fall.sqlite and spring.sqlite and then rename to database.sqlite when you want to use it.

 database.sqlite	3/7/2024 1:02 PM	SQLITE File	28 KB
----------------------------------------------------------------------------------------------------	------------------	-------------	-------

## 9.2 OPERATION

Once opening the program, begin by first creating the classes that they will want to schedule around. This can be done in two ways. One method is to manually create it in the Classes section. Start by clicking on **Classes** and then **Add Class**. Fill out the class department, section, meeting times, and dates of the week.



### Add Class

Class Department  
CPRE

---

Class Number  
1234

---

Section  
A

---

Monday ▾ Start Time 09:00 AM End Time 09:50 AM

---

Wednesday ▾ Start Time 09:00 AM End Time 09:50 AM

ADD TIME

CANCEL SUBMIT

The other option is by going to the **Import** section and then adding the class and then clicking on **Refresh.** This will pull the latest sections and then populate the Classes table reducing the amount of time a user needs to spend inputting all the class sections and meeting times.

Scheduler Classes Cohort Schedule Import

ADD IMPORT IMPORT SETTINGS REFRESH

Search

Actions	Department	Class Number
	EE	185
	MATH	165
	CPRE	281

5 rows |< < 1-3 of 3 > >|

Next, a user can create a **Cohort**. A Cohort in our program is a grouping of courses based on department and semester. The classes inside of a Cohort are based off a semester of the four-year plan. An example of a Cohort would be electrical engineering semester one or computer engineering semester seven. Cohorts are used to group classes so a new potential course can be scheduled around a target group of students like third semester electrical engineering students.

Create a **Cohort** by clicking on **Add Cohort**

Program	Semester #	Classes	Actions
No records to display			

Fill out the Add Cohort field to include program, semester, and the classes to schedule around.

**Add Cohort**

Program  
EE

Semester #  
Ex: 3  
1

Class  
ADD CLASS

**Cohort Classes:**

E E 185    DELETE

CANCEL    SUBMIT

Finally, the best fit for the new course can be determined. A user begins by clicking on the **Schedule** tab and then on **Class Settings**. From there, a potential class name can be entered.

**Class Settings**

Potential Class Name  
CPRE 1234

Select day and time sequence  
MWF - 50 Min ▼

Select classes to avoid:  
▼

ADD CLASS

Select Cohort  
▼

ADD COHORT

Avoiding:

CANCEL SUBMIT

A day and time sequence needs to be selected (Monday/Wednesday/Friday or Tuesday/Thursday). Then, the classes to avoid need to be selected as well as any cohorts to avoid.

## Class Settings

Potential Class Name

CPRE 1234

Select day and time sequence

MWF - 50 Min ▾

Select classes to avoid:



ADD CLASS

Select Cohort



ADD COHORT

Avoiding:

E E 285

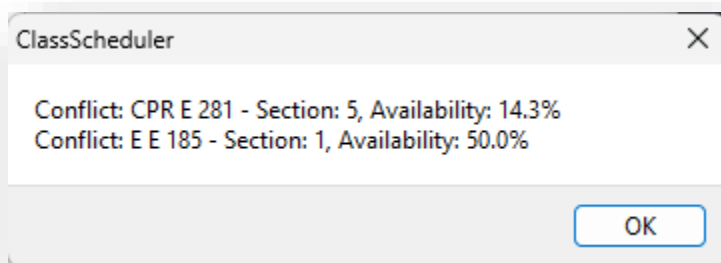
EE - Semester 1

CANCEL SUBMIT

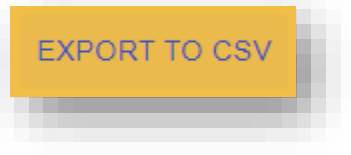
When submit is pressed, a calculation is performed, and each time slot has a percentage value on it as well as a color.

Scheduler			
Classes	Cohort	Schedule	Import
CPRE 1234 <span>x</span>	<span>+</span>	EXPORT TO CSV	CLASS SETTINGS
	Monday	Wednesday	Friday
7:45 am	100	100	100
8:50 am	100	100	100
9:55 am	100	83.33	100
11:00 am	100	83.33	100
12:05 pm	0	0	0
1:10 pm	100	100	35.71
2:15 pm	100	100	85.71
3:20 pm	100	100	100
4:25 pm	0	0	0
5:30 pm	100	35.71	100

If you click on a time slot, it will tell what conflicts there are as well as a percentage availability for each.



If you click on the **Export to CSV**, it will allow you to save to a CSV file that can be opened in a spreadsheet program.



The screenshot shows the Microsoft Excel interface with a warning banner that reads: "POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format." Below the warning is a data table with the following content:

1	Tab Title	Day	Time	Availability
2	CPRE 1234	Monday	7:45 AM	100
3	CPRE 1234	Monday	8:50 AM	100
4	CPRE 1234	Monday	9:55 AM	100
5	CPRE 1234	Monday	11:00 AM	100
6	CPRE 1234	Monday	12:05 PM	0
7	CPRE 1234	Monday	1:10 PM	100
8	CPRE 1234	Monday	2:15 PM	100
9	CPRE 1234	Monday	3:20 PM	100
10	CPRE 1234	Monday	4:25 PM	0
11	CPRE 1234	Monday	5:30 PM	100
12	CPRE 1234	Wednesd	7:45 AM	100
13	CPRE 1234	Wednesd	8:50 AM	100
14	CPRE 1234	Wednesd	9:55 AM	83.33
15	CPRE 1234	Wednesd	11:00 AM	83.33
16	CPRE 1234	Wednesd	12:05 PM	0
17	CPRE 1234	Wednesd	1:10 PM	100
18	CPRE 1234	Wednesd	2:15 PM	100
19	CPRE 1234	Wednesd	3:20 PM	100
20	CPRE 1234	Wednesd	4:25 PM	0
21	CPRE 1234	Wednesd	5:30 PM	35.71
22	CPRE 1234	Friday	7:45 AM	100
23	CPRE 1234	Friday	8:50 AM	100
24	CPRE 1234	Friday	9:55 AM	100
25	CPRE 1234	Friday	11:00 AM	100
26	CPRE 1234	Friday	12:05 PM	0
27	CPRE 1234	Friday	1:10 PM	35.71
28	CPRE 1234	Friday	2:15 PM	85.71
29	CPRE 1234	Friday	3:20 PM	100
30	CPRE 1234	Friday	4:25 PM	0
31	CPRE 1234	Friday	5:30 PM	100
32				

