Jed Davies

CS103 Group Project

School Information System

With Caitlin Hart

**Contents:**

**Research ………………………………………………………………………………………………………3**

**Requirements …………………………………………………………………………..………………….5**

**Algorithm …………………………………………………………………………………………………….6**

**Structure Diagram …………………………………………………..……………………………………7**

**Use Case Diagrams ………………………………………….……………………………………………8**

**Planning …………………………………………………………………………………………………….10**

**Coding Style Guide ..……………………………………………………………………………………10**

**Testing ……………………………………………….………………………………………………………11**

**Challenges Faced ………………………………………………………………………….……………12**

**Limitations …………………………………………………………………………………………………13**

**Links ………………………………………………………………………..………………..………………14**

**Information for Using the Program….………………………..………………..………………14**

**References …………………………………………………………………………………………………14**

**Research**

We conducted SWOT analyses of four similar products, based on lists of features and user reviews as we were not able to directly access the software.

**Alma** (Gartner, 2015)

Strengths

Has a district system and a per school system, meaning that people working with one will have greater familiarity and connectivity with the other

Enormous functionality makes it a ‘one-stop shop’ – handles everything from bus routes to report cards

Modern design

Weaknesses

Incomplete features rushed to market

Opportunities

ESSER Funding approved means schools have a financial motive to prefer Alma

Possibility of becoming a district/national standard

Threats

Standardization to another system

**Teach ‘n Go** (Gartner, 2018)

Strengths

Clean, appealing design

Weaknesses

Lack of export features

Poor mobile optimization

Opportunities

Prominence in Eurozone

Threats

Globalization of American competitors

Increasing focus on mobile computing in the world

**CANVAS** (Gartner, 2013)

Strengths

Open-source code allows for modularity and community support

Ever-expanding functionality

Weaknesses

Customer support and documentation is limited and arcane

Large and complex functionality undermines accessibility to low tech competence users

Opportunities

Can seize upon the vast amount of goodwill education has to harness volunteer coders

Threats

Feature bloat can lead to unwieldiness

**EDUHAPPY** (Gartner, 2018)

Strengths

User friendly interface

Weaknesses

High bandwidth use

Lengthy initial setup

Opportunities

Ease of use is essential as covid forces the technologically hesitant online

Threats

Competitors offer more features

**Requirements**

From our research we determined that our software would have to meet the following requirements:

Ease of use: teachers are a broad and varied group with many different levels of technological competence. It is essential that our software be simple to understand and use.

Clarity around errors: some users of other software reported that it was hard to figure out how to solve problems, so our error messages need to be clear and noticeable, offering clear direction on what to do.

Dependability: student information and grades can be hugely important, and our software needs to be thoroughly tested to ensure it is storing and accessing that data correctly.

**Algorithm**

I have not provided a complete algorithm due to the size and branching nature of the program. As a demonstration of algorithmic writing, here is an algorithm for the code that runs once an admin has chosen to update a student’s record.

1. Dereference arguments into vectors of parent, child and teacher structs and admin instance of Admin struct
2. Take inputs for child’s first name, last name, and classroom
3. Loop through children vector checking each instance for matching first name, last name, and classroom
4. If match found, increase flag and set match variable to match’s position in the vector
5. If flag is 0, alert user no such child found and offer retry (return to 1) or return to menu(run function adminMenu passing the memory address of the children, teachers, and parents vectors, and the memory address of the admin instance of the Admin struct)
6. If flag is greater than 1, alert user that administrator repairs are required to remove conflicting/redundant child records and offer retry (return to 1) or return to menu
7. Set update bool variable to the return of askUpdate function passing “first name”
8. Ask user if they want to update the child’s \**argument*\*(first name)
9. If yes, return true, if no return false
10. If update is true, instruct user to input new first name and receive input
11. Repeat from 7 for other members of Child (last name, preferred name etc)
12. Open 201.csv file
13. Loop through entire children vector, outputting each member of each instance, members separated by commas, instances separated by line breaks
14. Read 201.csv into children vector
15. Alert user that the update succeeded
16. Ask user if they want to update another child’s record
17. Return to 1 or return to menu

**Structure Diagram**

This diagram shows what screens lead where. The long column of records access can be read using the direction from which the arrows arrive – pointers on the left indicate that teacher accounts can access a functionality, the top for parents, the right for the administrator.

Diagram

Description automatically generated

**Use Case Diagrams**

For readability these are split up into several different diagrams

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated Diagram

Description automatically generated

**Planning**

We made a plan on Trello, dividing responsibilities based on our strengths and availabilities. Caitlin would focus on design, visual elements like spacing and colours, presentation and some UX elements. Jed would focus on feature development. We would both do some of everything, naturally, but these would be our areas of responsibility.

*Our Trello board at time of writing:*

Graphical user interface, application

Description automatically generated

**Coding Style Guide**

In our coding we chose to use camel case, plainly distinguishable function names that got to the purpose of the function, descriptive variable names except for loop iterators (the traditional i and j), indents based on what was within which curly braces, and comment headings for pieces of the code.

**Testing**

We asked our tester to sign in as administrator, register a child’s account, update the details for that child, log out and register a teacher, update that child’s grades, log out and register a parent for that child and view their report. We fed them incorrect inputs from some options/fields to test how the program handles those.

We observed the following issues and implemented these solutions:

“Please enter the year will you be teaching: ” was an ambiguous instruction. A response like “Year 9” was expected, but users input “2022.” An example has been added to combat this.

When encountering errors that output a new list of options, the error message could easily be lost on screen leaving the user confused. We added some system pauses to hold the program at the appropriate point.

Testing revealed several spacing issues that we adjusted to become more readable.

Redundant cin.ignore functions were removing letters from password inputs preventing them form matching. We removed the unnecessary ignores.

Some outputs used in debugging were still present. We removed those.

Not all file updates notified the user they had been successful. We included more such notifications.

At different window sizes the long output lines could become disrupted. We spaced them out with line breaks.

**Challenges Faced**

Our first challenge was coding a menu that would best handle bad inputs. I tried using a switch statement of an integer input, but if someone entered non-integer input the program wouldn’t handle it well. To account for any user input, not presuming the user would input only a number as instructed, I used string inputs – no need for getline as we are only looking for a single character. Switch statements won’t take strings, so I made our menus if/else if/else statements and enclosed them within while loops that would only terminate when one of the correct inputs had been set. This created ‘bulletproof’ menus that would offer clear instructions to users in error without disrupting the program.

This was our first experience with Github, and we had some teething problems with merging – not realizing that one has to push after only opening the program and closing it, for example. We learned to stash unwanted changes to allow pulling and force merges when necessary.

I wrote a lot of functions and didn’t test them until I had later built the menus that called them. In testing these I found a lot of logical errors that were harder to resolve because the work wasn’t fresh in my mind. One simple example is the code I was using to overwrite a file. I had enclosed the file closure within the loop that should output each instance of a struct from a vector, meaning that I could register any number of students, but when I updated their details, only the first one would still be in the file. I’ve learned to test everything as you build it, working with agility, as I believe I would have found this error more easily if I had tested the code immediately after writing it.

**Limitations**

While our product is viable in its current form, I would be remiss not to admit some possible improvements.

Firstly, the program is hardcoded with only one classroom, using the ‘201.csv’ file. While our tutor has approved this, it is worth mentioning that some if statements to direct the program to the file based on the specified classroom would need to be added to handle multiple classrooms in separate files.

The build does not support associating multiple children with the same parent account, nor multiple parents with the same child record. While this allows full functionality, it would be more convenient and secure for users if they didn’t have to share account details with other caregivers or manage multiple accounts for their children.

Outputting to a comma-separated-value file carries the risk of users inputting a comma, which will store their data in the wrong cells and consequently the wrong members of the structure when read.

Passwords are visible while being typed in, posing a security risk.

When searching for a record, users must input exactly what is stored – for example ”Room 201” will not find a classroom stored as “201”. An ideal system would have some capacity to find results similar to the user’s inputs.

More precise validation would make the registration process more likely to succeed – for example, the current email validation checks for an @ symbol somewhere in the string. A more sophisticated system might send them an email with a code, to confirm all at once that is a valid email address and the user can access it.

A more feature complete version is possible. It may be better to allow admin to edit and delete teacher records, for example. Our code features several ways to edit records, and we did not feel it necessary to demonstrate this again in more instances for this assessment, but it bears mentioning that if we were to take this product to market it would be worthwhile.

A version of this made for a specific school could have better-tailored datatypes. Many fields are left as strings, for example grades, as we don’t know if a school will be entering letter grades, percentages, number of marks, or NCEA style Achieved, Not Achieved, Achieved with Merit, Achieved with Excellence. If we knew, for example, they were using percentages, we could use float variables and reject and less than 0 or greater than 100, as well as give automatic averages and totals.

**Links**

**Github Repository**

<https://github.com/Gumption-and-Gusto/GroupProject>

**Trello board** (Beula has access, email me at [jed.davies@gmail.com](mailto:jed.davies@gmail.com) if a moderator needs access)

<https://trello.com/b/iPzokT6A/school-information-project-cs103>

**Google Blog**

<https://jedandcaitlingroupproject.blogspot.com/>

**Information for Using the Program**

The admin account has hardcoded username “PrincipalAdmin” and password “AdminAdmin123+”. Several accounts are preregistered in the files – Teacher account MmeCurie (password: P1erre<3), parent account MarcoR (password: Eliseo@School1) and the details of a few children.

# **References**

Gartner. (2013). *CANVAS Pricing, Reviews and Features*. Retrieved from Capterra New Zealand 2021: https://www.capterra.co.nz/software/127214/canvas

Gartner. (2015). *Alma Pricing, Reviews and Features*. Retrieved from Capterra New Zealand 2021: https://www.capterra.co.nz/software/133283/alma

Gartner. (2016). *EDUHAPPY Pricing, Reviews and Features*. Retrieved from Capterra New Zealand 2021: https://www.capterra.co.nz/reviews/150575/eduhappy

Gartner. (2018). *Teach n Go Pricing, Reviews and Features*. Retrieved from Capterra New Zealand 2021: https://www.capterra.co.nz/reviews/149259/teach-n-go