

# “University Admissions Management Systems – How the College Admissions Process has Changed”

Project by Caleb A. Street

February 14, 2021

The University of Virginia’s College at Wise



This project is for the completion of the Bachelor of Science  
degree in Computer Science / Software Engineering at  
The University of Virginia’s College at Wise

## Acknowledgements

*I would like to thank God for helping me get through my college education and this project, and for everything else that He does for me.*

*I would like to thank my parents for supporting me throughout my life, my education, and this project.*

*I would like to thank the professors that I have had throughout the years for teaching me how to do this kind of work and for being patient with me and helping me when I had questions.*

*I would like to thank all the staff at the college who have helped me on my journey throughout my college years.*

## **Abstract**

Paper application forms for colleges and universities are becoming very outdated and difficult to use, both for the applicants and the admissions staff. Students are becoming more and more used to using technology and may even decide not to apply to a college or university that does not accept electronic applications thinking that the college or university itself is too outdated. Additionally, large numbers of applicants can create a major burden for admissions staff if they have to search through and sort every application by hand. There is, however, a solution to these problems, an electronic application system.

This project included conducting research on college admissions and how to create a system that can sort through all those applications and determine which applicants are the most likely to get accepted into the college or university. Once these topics were well researched, an actual system was implemented. The system included an electronic application form, application sorting software, and multiple databases to store all the information needed by the system and college personnel.

This project concluded that an efficient electronic system could be implemented that would accept applications and sort through them to determine which applicants are more likely to be accepted by the college or university. It also showed that the system made a large reduction in the pool of applicants that needed to be searched through by the admissions staff.

## Table of Contents

<b>CHAPTER 1 - INTRODUCTION .....</b>	<b>6</b>
1.1 INTRODUCTION TO THE PROBLEM .....	6
1.2 AIMS AND OBJECTIVES OF THE STUDY .....	7
1.2.1 Aims .....	7
1.2.2 Objectives .....	7
1.3 SIGNIFICANCE OF THE STUDY .....	7
1.4 DEFINITION OF TERMS .....	7
1.4.1 University Admissions Management System .....	7
1.4.2 Python .....	8
1.4.3 SQLite3 .....	8
1.4.4 Electronic Application Form .....	8
1.4.5 Database .....	8
1.4.6 Application Sorting Software .....	8
<b>CHAPTER 2 – LITERATURE REVIEWS .....</b>	<b>9</b>
2.1 INTRODUCTION .....	9
2.2 ARTICLE ONE REVIEW .....	9
2.2.1 Why was the Study Conducted? .....	9
2.2.2 Were there any Similar Works Discussed in the Paper? .....	9
2.2.3 What were the Procedures or Methods Used? .....	10
2.2.4 What Conclusions did the Authors Reach? .....	10
2.2.5 How could this Work be Improved? .....	10
2.3 ARTICLE TWO REVIEW .....	11
2.3.1 Why was the Study Conducted? .....	11
2.3.2 Were there any Similar Works Discussed in the Paper? .....	11
2.3.3 What were the Procedures or Methods Used? .....	11
2.3.4 What Conclusions did the Authors Reach? .....	12
2.3.5 How could this Work be Improved? .....	12
<b>CHAPTER 3 – SYSTEM ANALYSIS AND DESIGN .....</b>	<b>13</b>
3.1 INTRODUCTION .....	13
3.2 METHODOLOGY .....	13
3.3 SYSTEM ANALYSIS .....	14
3.3.1 Analysis of Existing System .....	14
3.3.2 Analysis of the Proposed System .....	14
3.3.3 Advantages of the Proposed System .....	15
3.4 SYSTEM DESIGN .....	15
3.5 INPUT REQUIREMENTS AND LAYOUT .....	16
3.6 OUTPUT LAYOUT .....	17
3.7 DESCRIPTION OF THE ALGORITHMS .....	17
3.8 PROGRAM FLOWCHART (APPLICATION SORTING SOFTWARE) .....	18
<b>CHAPTER 4 – SYSTEM IMPLEMENTATION AND DOCUMENTATION .....</b>	<b>19</b>
4.1 INTRODUCTION .....	19
4.2 SYSTEM DESIGN DIAGRAM .....	19
4.3 CHOICE OF PROGRAMMING LANGUAGE .....	21
4.4 ANALYSIS OF MODULES IN SOURCE CODE .....	21
4.4.1 tkinter .....	21
4.4.2 pyautogui .....	22
4.4.3 sqlite3 .....	22
4.4.4 os.path .....	22
4.5 DESCRIPTION OF PROGRAMMING ENVIRONMENT .....	22
4.6 SYSTEM IMPLEMENTATION .....	23

<b>CHAPTER 5 – SUMMARY, CONCLUSION, AND RECOMMENDATION.....</b>	<b>27</b>
5.1 INTRODUCTION.....	27
5.2 CONSTRAINTS OF THE STUDY .....	27
5.2.1 <i>Time Constraints</i> .....	27
5.2.2 <i>Single Person Development Team</i> .....	27
5.2.3 <i>Small Test Cases</i> .....	27
5.2.3 <i>No Functionality for Remote use of Form</i> .....	28
5.3 SUMMARY .....	28
5.4 CONCLUSION .....	29
5.5 RECOMMENDATION .....	29
<b>REFERENCES .....</b>	<b>31</b>
<b>APPENDIX A: PROGRAM FLOWCHART (APPLICATION SORTING SOFTWARE) .....</b>	<b>32</b>
<b>APPENDIX B: SOURCE CODE LISTING .....</b>	<b>33</b>
<b>APPENDIX C: FIGURES .....</b>	<b>38</b>

## Chapter 1 - Introduction

### 1.1 Introduction to the Problem

It is getting harder and harder for students to be accepted into the universities in which they would like to attend. As more students begin to apply to college, the competition to get accepted into college gets harder and harder. From 1992 to 2004, the number of students that applied to a four-year college increased by 44 percent (Bound, Hershbein, & Long, 2009). This large increase in applicants creates a large pool of students for universities to cypher through and decide who gets admitted and who does not get admitted. How do universities search through large numbers of applications? Are the handwritten applications always readable? Do they miss any important information that might have made a difference as to whether a student gets accepted or declined? Is this process completely fair to all, or does the college make unfair decisions without even realizing those decisions were unfair? These are questions many people are asking themselves.

Imagine if you were a student who had worked hard throughout your high school years to get good grades and to be able to get into the school of your dreams, just to be rejected unfairly. This has happened to many students throughout the years. Take the recent college admissions scandal that involved several wealthy families bribing college employees to get their children into the best colleges as an example. According to Jaschik (2019),

What many are calling the worst admissions scandal in higher education emerged Tuesday, with federal authorities announcing 50 indictments in a scheme that allegedly involved faux athletes, coaches who could be bribed, cheating on the SAT and ACT, million-dollar bribes and “guarantees” that certain applicants would be admitted to highly competitive colleges (p. 1).

All of these wealthy students that are getting into colleges based on bribes and falsified test results are preventing other, more deserving students from being admitted into those colleges.

Now, imagine that you are an admissions employee at a large college that uses paper application forms. The college through which you are employed gets thousands of applications each year. It is your job to sort through each of those applications and determine which students meet the requirements for the college and which of those students are the best fit for the college. How do you accomplish this massive task, and more importantly, how do you accomplish it fairly? This is a very daunting task that can be difficult for one or two people to complete. However, if several people sort through the applications, there is a higher chance of varying opinions on which students should be admitted to the college. These varying opinions from several different people can lead to unfair results in the admissions process. For example, if one admissions employee is more lenient than another admissions employee, then less qualified students may get admitted by the lenient employee and more qualified students get rejected by the stricter admissions employee.

Each of the problems listed above are good reasons as to why colleges should have an electronic university admissions management system. This electronic system could help sort through the thousands of applications that colleges receive each year and help make the whole process much quicker and fairer.

## **1.2 Aims and Objectives of the Study**

### **1.2.1 Aims**

1. To research and obtain a well-founded understanding of university admissions.
2. To explain why an electronic university admissions management system is needed.
3. To create an electronic university admissions management system.

### **1.2.2 Objectives**

1. Research other peer reviewed, scholarly articles to obtain insight on university admissions.
2. Create an electronic college application form using the python programming language.
3. Create a database to store applicant information using SQLite3.
4. Create software using the python programming language that can sort applications and select a small pool of applicants from which admissions officials can hand select applicants to be admitted.

## **1.3 Significance of the Study**

This study will help the college admissions process become much more efficient. Colleges will be able to use the electronic system to sort out the majority of the applications and have one or two admissions employees hand select students to be admitted from those selected by the electronic system. Not only would this electronic system make it much easier for the admissions employees to sort through the applications, but it would also make the process fairer. Students would be much more likely to get accepted based fully on merit instead of lying, cheating, and bribery.

An electronic admissions system will also make the college application process easier for the students. Students today are becoming more and more accustomed to working on computers, and less and less accustomed to filling out paper forms. This is especially true in our current situation of many schools and universities participating in remote instruction. Therefore, it would be much easier and quicker for many students to fill out the electronic application. There is also the possibility of making the application available remotely.

## **1.4 Definition of Terms**

### **1.4.1 University Admissions Management System**

A university admissions management system is a system for managing university or college admissions. This management system can be a paper or electronic system. The purpose of this study is to make a transition from the paper system to an electronic system.

### **1.4.2 Python**

Python is a very powerful, high-level, interpreted programming language. It is especially useful for analyzing and sorting data. It is being used in this study to create the electronic application form and the application sorting software. Python is available to be used cross-platform and can be downloaded for free from the python website.

### **1.4.3 SQLite3**

SQLite3 is software that allows for the creation and editing of databases. It is available to be used cross-platform and comes preinstalled on most machines. It is a lightweight and very popular database editing software. According to the SQLite website (SQLite, 2021), there are over 1 trillion active SQLite databases. The source code for SQLite is listed in the public domain and is free to use.

### **1.4.4 Electronic Application Form**

An electronic application form for college admissions would be very similar to the paper application form. It still includes many of the same fields that the paper application contained, but the responses can be typed instead of written. It also allows the applicant to enter lists of clubs he/she was a member of in high school, list of extracurricular activities he/she participated in during high school, and even an additional comments section that allows the applicant to enter any comments in which he/she would like the admissions employees to be aware. It is being implemented using the python programming language.

### **1.4.5 Database**

A database is a software structure that can organize and store large amounts of data. They are sectioned into tables, columns, and rows. A table is a closely related section of data. A column is a specific type of data within the table. A row contains specific pieces of data that relates to the column in which it is listed. Therefore, searching for a table, a column, and a row in a database will give a very specific piece of data. Databases are being used in this study to organize, store, and sort applications, admitted students, and rejected students. It is being implemented using SQLite3.

### **1.4.6 Application Sorting Software**

The application sorting software is a program that will be able to search through the applicant database and make decisions based on the information provided in each of the applications. It will be able to sort out which students will be a good fit for the college or university, and those that will not be a good fit for the college or university. The software will then return the applicants that it believes could be admitted into the college so the admissions employee can sort through them and make the final decision. It will be implemented using the python programming language.



## **Chapter 2 – Literature Reviews**

### **2.1 Introduction**

This chapter will discuss a couple articles that I used to help me get started with the study. The first article is on college admissions. The second article is on decision support systems. This chapter will discuss topics such as why those studies were conducted, what steps the authors of those studies took to reach a conclusion, and the conclusions of each of those studies.

### **2.2 Article One Review**

The article that is under review in this section is titled “*Playing the Admissions Game: Student Reactions to increasing College Competition*” (Bound, Hershbein, & Long, 2009). This article is on how college admissions and enrollment rates have changed over the years. It breaks down the information to discuss the differences in large universities and small colleges. The specifics of this report will be discussed in the following sections.

#### **2.2.1 Why was the Study Conducted?**

This study was conducted to determine and show how the application and admission rates to different types of universities are changing over time and how those changes are affecting the students. In particular, this study discusses the differences in the changes of application and admission rates between private and public universities, universities in different regions, and all universities as a whole. It also discusses how students are changing their plans and activities throughout their high school years to accommodate for these changing rates.

#### **2.2.2 Were there any Similar Works Discussed in the Paper?**

There were several similar works referenced in this paper, but they were not discussed in detail. Instead of discussing other studies, this article brings up points and then uses the other studies as concrete evidence to support those points. This article used these other studies as evidence many times throughout the entirety of this report. There are about two pages of references at the end of the report. For example, this article states, “Other work has also documented this increasing segmentation within higher education (Hoxby, 1997, this issue; Bound, Lovenheim, and Turner, 2008).” (Bound, Hershbein, & Long, 2009). This allows the author to give support for his/her statements by saying that other reports have found the same conclusions without having to go into the details of those reports. If the reader is interested, he/she can then read the other articles to get an understanding of their results.

### **2.2.3 What were the Procedures or Methods Used?**

This article used a lot of data that had been published in other reports and compiled it all together to support its topic and claims. In addition to this data, it also used information obtained from different National Center for Education Statistics surveys. The authors of this article compiled all the related information into various tables (which can be found at the end of the article). The data from all the different surveys were also placed into tables. These tables were then referenced throughout the article to give evidence for the article's claims. As the article moved from one subtopic to the next, it would just reference a different table. It allowed the authors to reference concrete support without going into a lot of detail and potentially confusing the reader or making the reader lose his/her train of thought or even the point of the article. It helped the article stay on track without losing focus on the main target. The authors, however, did not obtain their own information. They just used survey results and data obtained from other sources. In my opinion, this article would have been better had they conducted their own surveys and showed that other reports had similar data to its own results. If the authors had conducted the surveys themselves, they could have been surer of the results of their report. By using only external data, they cannot be completely sure the results were accurate. However, this problem was somewhat lessened by them using multiple other sources that had similar results.

### **2.2.4 What Conclusions did the Authors Reach?**

The authors of this article concluded that colleges and universities have greatly changed over the past many years. They state that both the advanced students and the average students were suffering from more competition over getting into their selected college. They also state that competition has grown faster in California and in the Northeast (than in the more rural regions for example). Furthermore, they state that this competition might be good if it causes students to learn more, but that might not always be the case based on their findings. Additionally, they conclude that many students are at a disadvantage due to where they are from or where their parents went to college. Finally, they state that private colleges spend a much larger amount of money than public colleges, and that there is a large gap between the two types of colleges.

### **2.2.5 How could this Work be Improved?**

This article could be improved if the authors had conducted their own surveys and gathered their own data instead of using data and surveys from other sources. If they had done it on their own, they could have been even more confident in their results and conclusion. It would have allowed them to have their results supported by the results of other reports. Instead, they are just using the other reports as supporting evidence. Additionally, while they did keep the article to the point by referencing the tables located at the bottom of the page instead of having all the information scattered throughout the report, I feel that it would have been better if the article had been even more concise than what it is already. At some point, the article almost becomes confusing with so many different year ranges that switch back and forth every few sentences.

## 2.3 Article Two Review

The article that is under review in this section is titled “*Intelligent Decision Support Systems for Admission Management in Higher Education Institutes*” (Vohra & Das, 2011). This article discusses different types of decision support systems, what each of them are used for, and how they can be used together to create a powerful system for admissions in higher education. The specifics of this report will be discussed in the following sections.

### 2.3.1 Why was the Study Conducted?

This study was conducted to explain different types of systems and why they are used. It also discusses how these systems can work together to create a larger, more powerful system for college admissions. The authors of this study are proposing this combined system as a possible solution for the decision-making process of an electronic university admissions management system.

### 2.3.2 Were there any Similar Works Discussed in the Paper?

The authors of this report did not specifically discuss any other similar works within their paper. They did, however, reference several other works throughout the entirety of their report. These references were just used as supporting materials instead of a discussion of other works. It would have been good if the authors of this report could have provided more information on the studies they referenced, but referencing several other sources allows the reader to know the authors of the report at least conducted research on the topic in which they are discussing.

### 2.3.3 What were the Procedures or Methods Used?

The authors of this report did not complete an actual project. They merely researched the topic and provided information on the systems needed to create a decision system for a university admissions management system. The types of systems discussed include Enterprise Resource Planning System (ERP), which is used for business management; Decision Support System

**Table 1: Decisional Situations (Vohra & Das, 2011)**

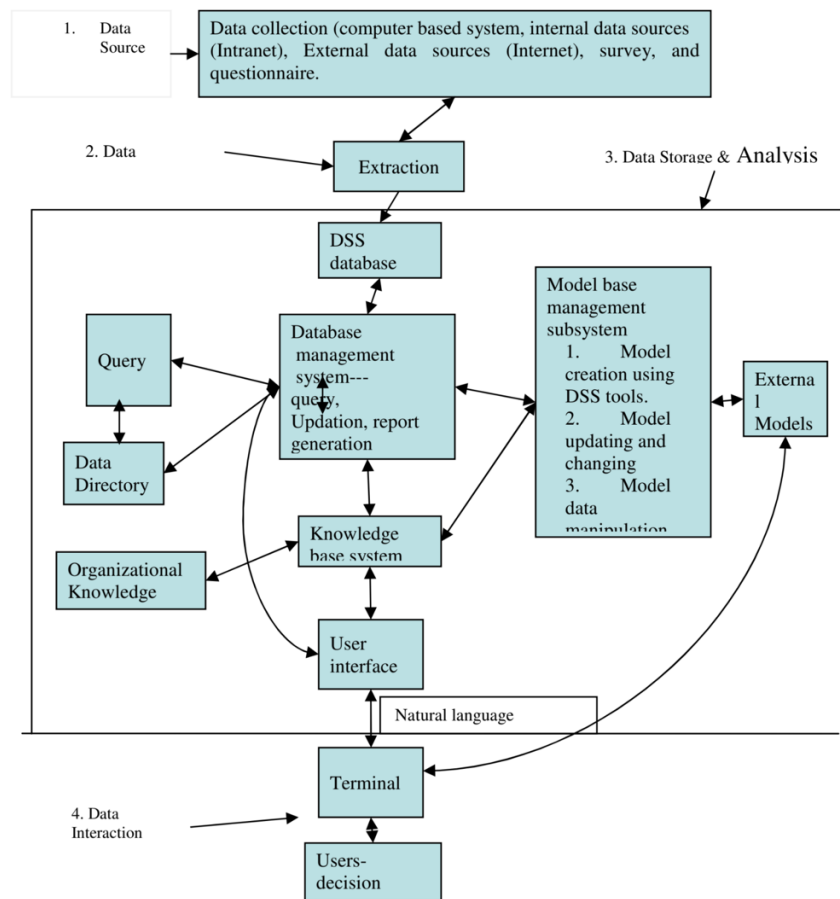
DSS MODULE

STUDENT MODULE

DECISIONAL  
SITUATION

Students' enrollment	Faculty
Choosing a Specialization (Course/ Branch)	Scholarship
Studies reclassification	Career guidance
Recognition	Web Information and announcements
Tuition fees	Hostel facility
Ranking of college	Transportation
Infrastructure	Library
Interruption of studies(BUNK)	Extension of studies
Tutorial activities	Parking facility
Health facility	Research facility

(DSS), which is used for decision making; Intelligent Decision Support System (IDSS), which is the DSS system with Artificial Intelligence (AI) included; and several other smaller systems. Additionally, they described several situations in which each of these systems would be needed. They also provided a few figures and tables that helped explain how the systems work and when they need to be used. A couple of these figures can be seen in Table 1 and Figure 1.



**Figure 1: Schematic View of DSS (Vohra & Das, 2011)**

### 2.3.4 What Conclusions did the Authors Reach?

Since the authors did not conduct their own experiment in this study, they did not have specific results. However, they did discuss how to combine all the smaller systems into one larger, more powerful, and more effective system. They also discussed how they may actually implement this system in a future study.

### 2.3.5 How could this Work be Improved?

This work would have been much more effective if the authors of the report had implemented the system they described. This would have shown readers that creating the system they discussed is possible and that is indeed an effective solution to the problem they were trying to solve. The way it currently stands, readers may still have some doubt that the system will work the way the authors say it will work. However, since they conducted research on the topic and used several sources as support, the system they describe should be an effective solution that is possible to implement, but their work still would have been more effective if they had proven their statements to be true.

## **Chapter 3 – System Analysis and Design**

### **3.1 Introduction**

This chapter will discuss the design of how the system will function as well as give an analysis of both the current and the proposed systems. It will also provide insight as to why the proposed system is needed and what advantages it will have over the current system. A detailed description of the proposed system can be found in the following sections of this chapter.

### **3.2 Methodology**

The system will consist of multiple databases, python programs, and a text file. Everything will be created using macOS but is expected to run on any major operating system (macOS, Linux, Windows, etc.). It will be created to run on a single machine but could be expanded upon to include a client-server architecture to allow applicants to submit their application remotely.

The system will begin by opening the electronic application form. Applicants will be allowed to fill out the first page of the application. They will then need to click the submit button at the bottom of the page. This will submit the first page of the application to the Applicant database. The form will then display the second page of the application. The user will then have to fill out this page and click the submit button at the bottom of the page once again to submit this information to the Applicant database. The form will automatically go back to the beginning of the first page for the next applicant. This entire process will be repeated for each applicant.

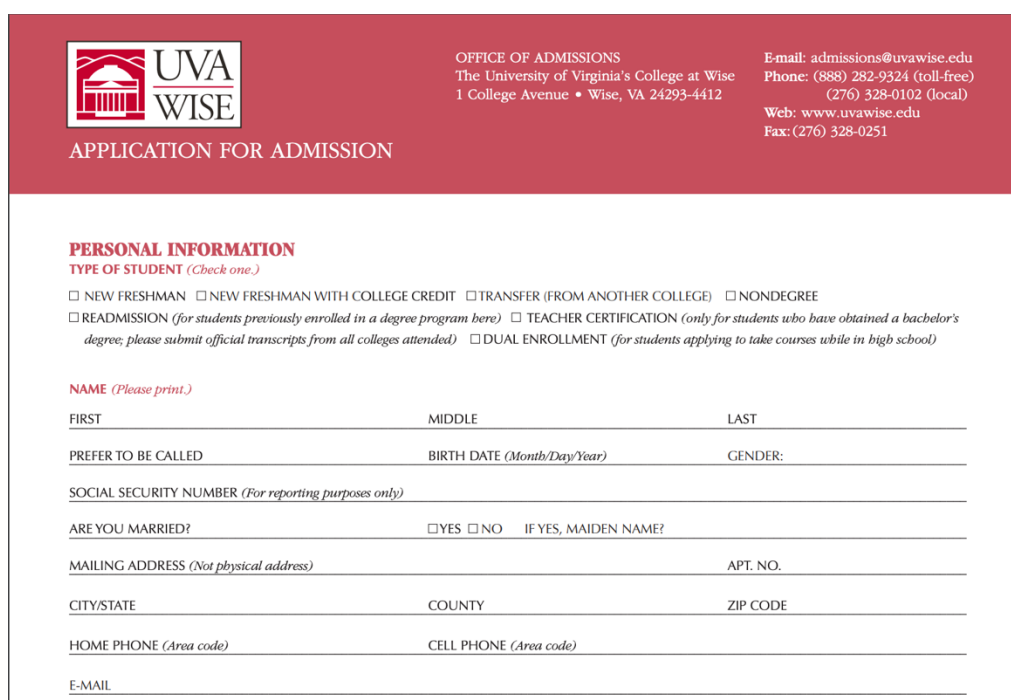
Once every applicant has submitted their application. The application sorting software will need to be executed. This software will query the Applicant database to retrieve data on each applicant. It will then calculate a score for multiple judging categories (ex. GPA, class rank, clubs, etc.) and an average or overall score and enter those scores into a Scores database (Clinedinst & Koranteng, 2017; Rigol, 2003). This database will help to organize and store the scores of each applicant. Once the scores have been calculated for each applicant, the software will query the Scores database to retrieve a predefined number of applicants with the highest overall average score. Those applicants will be entered into the Approved database. The remaining applicants (those with the lowest overall average score) will be entered into the Rejected database.

Finally, an admissions staff member will search through the Approved database to determine which of those applicants will be accepted and which ones will not be accepted. The Rejected database (along with the remaining applicants not accepted from the Approved database) can be used to notify applicants that they have not been accepted to the college or university. The accepted applicants from the Approved database can be notified that they have been accepted to the college or university.

### 3.3 System Analysis

#### 3.3.1 Analysis of Existing System

Many colleges are still accepting paper applications. Paper applications are much more difficult for students to fill out than electronic forms, and much more difficult for university admissions staff to search and sort through. This can create major problems for the staff. They will have to make fair decisions on who should be accepted and who should not be accepted, and paper applications make that decision process more difficult because it takes more time to sort through. Paper applications, however, give applicants an opportunity to give more details than many of the electronic applications. As an example, part of a paper application can be seen in Figure 2.



The image shows a paper application form for UVA WISE. The header is red with the UVA WISE logo on the left, contact information in the center, and email/phone/web/fax details on the right. The main body is white with a red section for 'PERSONAL INFORMATION'. It includes checkboxes for student type (New Freshman, Transfer, etc.), name fields (First, Middle, Last), birth date, gender, social security number, marital status, mailing address, city/state/county/zip code, home/college phone, and email.

**UVA WISE**

OFFICE OF ADMISSIONS  
The University of Virginia's College at Wise  
1 College Avenue • Wise, VA 24293-4412

E-mail: admissions@uvawise.edu  
Phone: (888) 282-9324 (toll-free)  
(276) 328-0102 (local)  
Web: www.uvawise.edu  
Fax: (276) 328-0251

**APPLICATION FOR ADMISSION**

**PERSONAL INFORMATION**  
*TYPE OF STUDENT (Check one.)*

☐ NEW FRESHMAN ☐ NEW FRESHMAN WITH COLLEGE CREDIT ☐ TRANSFER (FROM ANOTHER COLLEGE) ☐ NONDEGREE  
☐ READMISSION (for students previously enrolled in a degree program here) ☐ TEACHER CERTIFICATION (only for students who have obtained a bachelor's degree; please submit official transcripts from all colleges attended) ☐ DUAL ENROLLMENT (for students applying to take courses while in high school)

**NAME (Please print.)**

FIRST MIDDLE LAST

PREFER TO BE CALLED BIRTH DATE (Month/Day/Year) GENDER:

SOCIAL SECURITY NUMBER (For reporting purposes only)

ARE YOU MARRIED? ☐ YES ☐ NO IF YES, MAIDEN NAME?

MAILING ADDRESS (Not physical address) APT. NO.

CITY/STATE COUNTY ZIP CODE

HOME PHONE (Area code) CELL PHONE (Area code)

E-MAIL

**Figure 2: UVA WISE Paper Application Form**  
(The University of Virginia's College at Wise)

#### 3.3.2 Analysis of the Proposed System

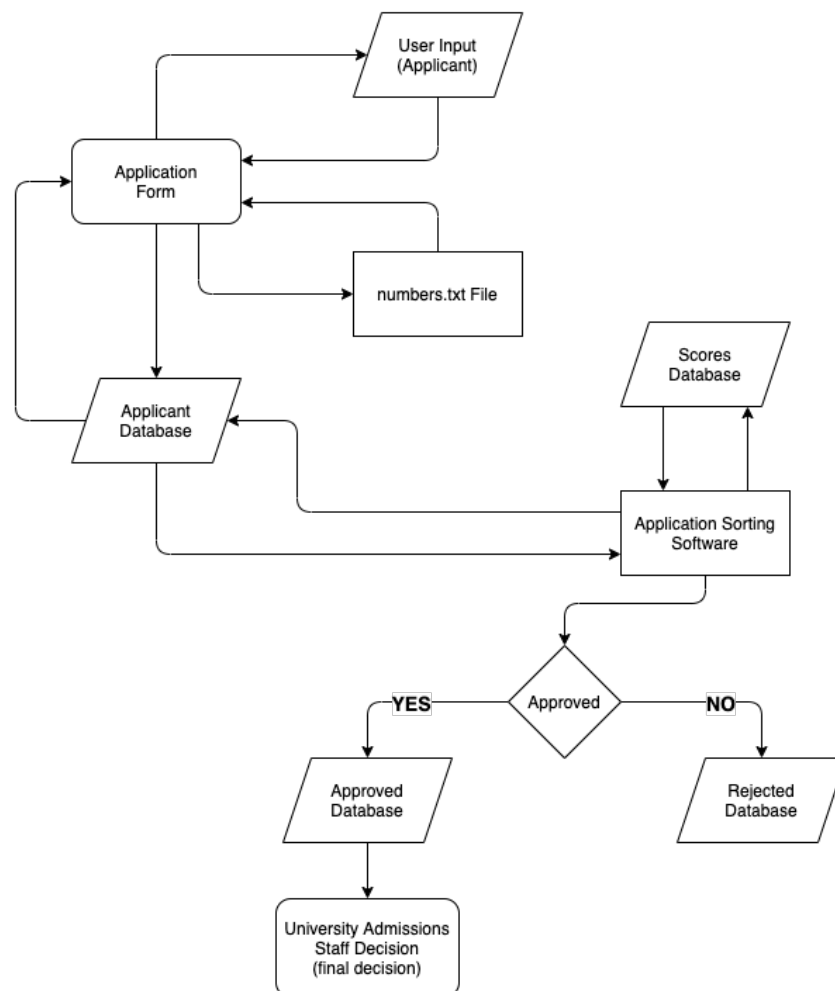
The proposed system will include an electronic application form. This electronic form will submit the applicant's responses to a database that can be queried to get the applicant's responses. There would then be a separate program that would be able to sort through all the applications in the database to determine which students are more likely to get accepted into the college or university. The system will give a larger number of students than what can be accepted to the college. The admissions staff will then be able make the final decision between the top students without having to manually search through all the applicants.

### 3.3.3 Advantages of the Proposed System

The proposed system would be advantageous because:

- It would be easier for students to fill out electronic form than the paper form.
- It would be easier for admissions staff to sort through the applications.
- It would make the outcome of the admissions decision fairer.
- It could be expanded to allow the applicants to fill out the application remotely.

### 3.4 System Design



**Figure 3: System Flowchart**

#### Steps:

1. Open the Application Form
2. Request user input (allow user to fill out the form)
3. Get user input (answered fields in the form)

4. Send user input to the Applicant Database
5. Store person\_ID field in the numbers.txt file
6. Return to the Application Form
- \* Repeat steps 2 and 3 to get input for second page of the form
6. Read in the last entry in the numbers.txt file to get the current person\_ID field
7. Send user input to the Applicant Database
- \* Repeat steps 2 – 7 until all applicants have applied
8. Application Sorting Software gets applicant information from the Applicant Database
9. Application Sorting Software calculates a score for multiple fields
10. Application Sorting Software calculates an average score for each applicant
11. Application Sorting Software sends the scores to the Scores Database
- \* Repeat steps 8 - 11 until scores have been calculated for every applicant
12. Application Sorting Software queries the Scores database to get a predefined number of applicants with the highest overall average scores
13. Application Sorting Software stores those applicants' information in the Approved database
14. Application Sorting Software queries the remaining applicants (lowest average scores)
15. The remaining applicants' information is added to the Rejected database
15. University Admissions Staff makes the final decision from the Approved Database

### 3.5 Input Requirements and Layout

Input to the system will consist of applicants filling out multiple fields on the application form. These fields will consist of personal information, education history, etc. The form will also consist of multiple pages. Once each page of the form is filled out, the user will have to click a submit button to submit the information to the database. An example of the form can be seen in Figure 4.

The screenshot shows the UVA WISE Electronic College Application Form. The form is organized into two main sections: 'Personal Information' and 'Prior Education'. The 'Personal Information' section contains fields for Personal ID Number, First Name, Middle Name, Last Name, Social Security Number, Date of Birth, Gender, Are you a US Citizen, What is your Ethnicity, What is your Primary Language, Email Address, Home Phone Number, Call Phone Number, Street Address, City, State, and Zip Code. The 'Prior Education' section contains fields for High School Code, High School Name, Class Rank, GPA, SAT Score, Average Number of Total Courses Taken per Semester, Total Number of College Credits Earned, and Average College Courses Grade (100 point scale). At the bottom, there is a 'Submit and go to the next page:' button.

**Figure 4: Electronic Application Form Example**



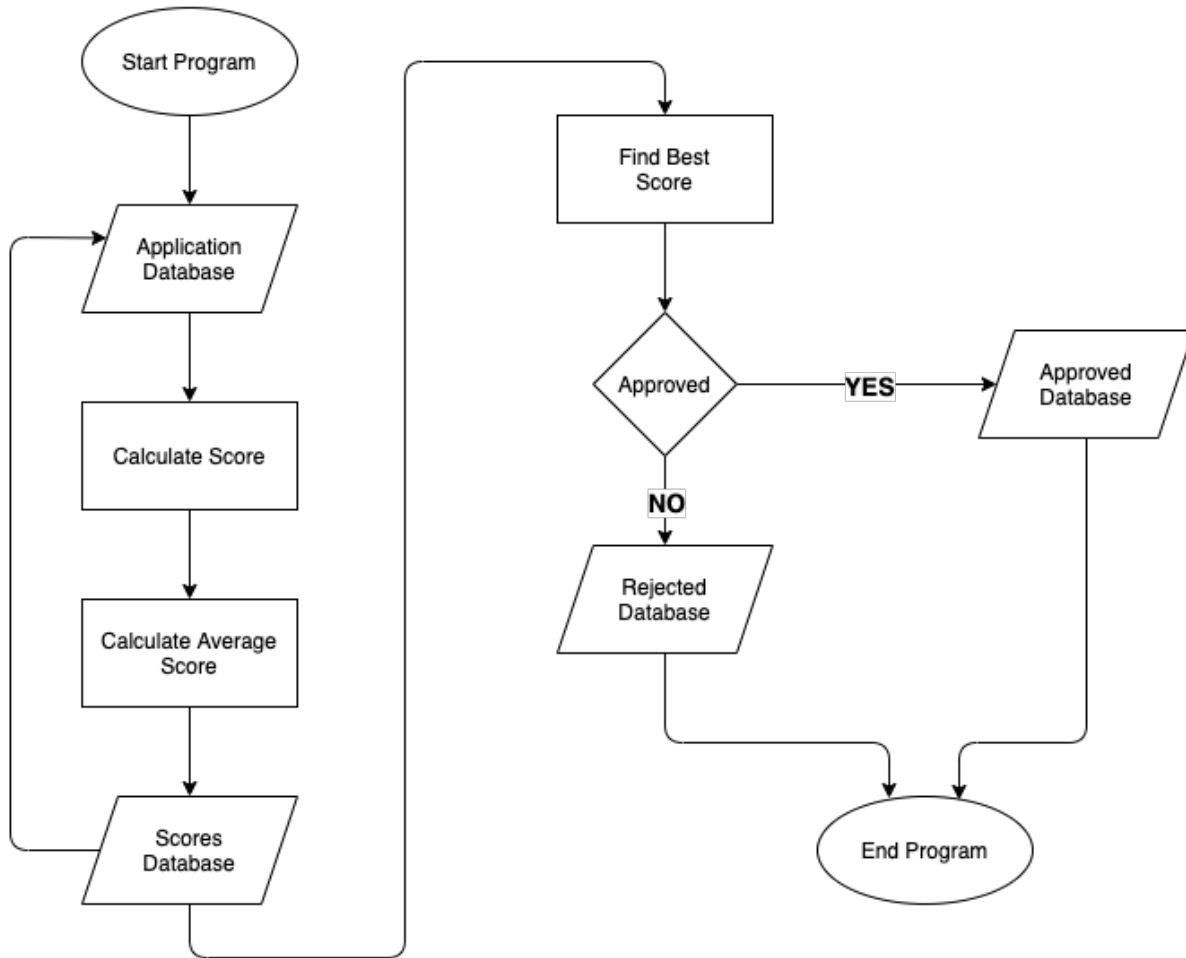
### **3.6 Output Layout**

The output of this system will include four databases, the Approved database, the Rejected database, the Scores database, and the Applicant database. The Approved database will contain information on each of the applicants that have been approved by the system. The Rejected database will contain information on each of the applicants that have not been approved by the system. The admissions staff will then have access to the Approved database to make the final decision on which applicants get accepted. The college or university will also be able to use the Rejected database (and the applicants from the Approved database that did not get accepted by admissions staff) to notify those applicants that they are not being admitted to that college or university. The Scores database and the Applicant database will be stored for recording purposes in case of a later dispute.

### **3.7 Description of the Algorithms**

This system will make use of a grading rubric for several different judging categories, such as GPA, class rank, clubs, etc. It will assign a numeric score to each applicant for each of the categories. The system will then calculate the average score for each applicant. Once all the averages have been calculated, it will pick a predetermined number of applicants (determined by the college or university) with the highest scores to be entered into the Approved database. All the applicants that were not chosen by the system will be entered into the Rejected database.

### 3.8 Program Flowchart (Application Sorting Software)



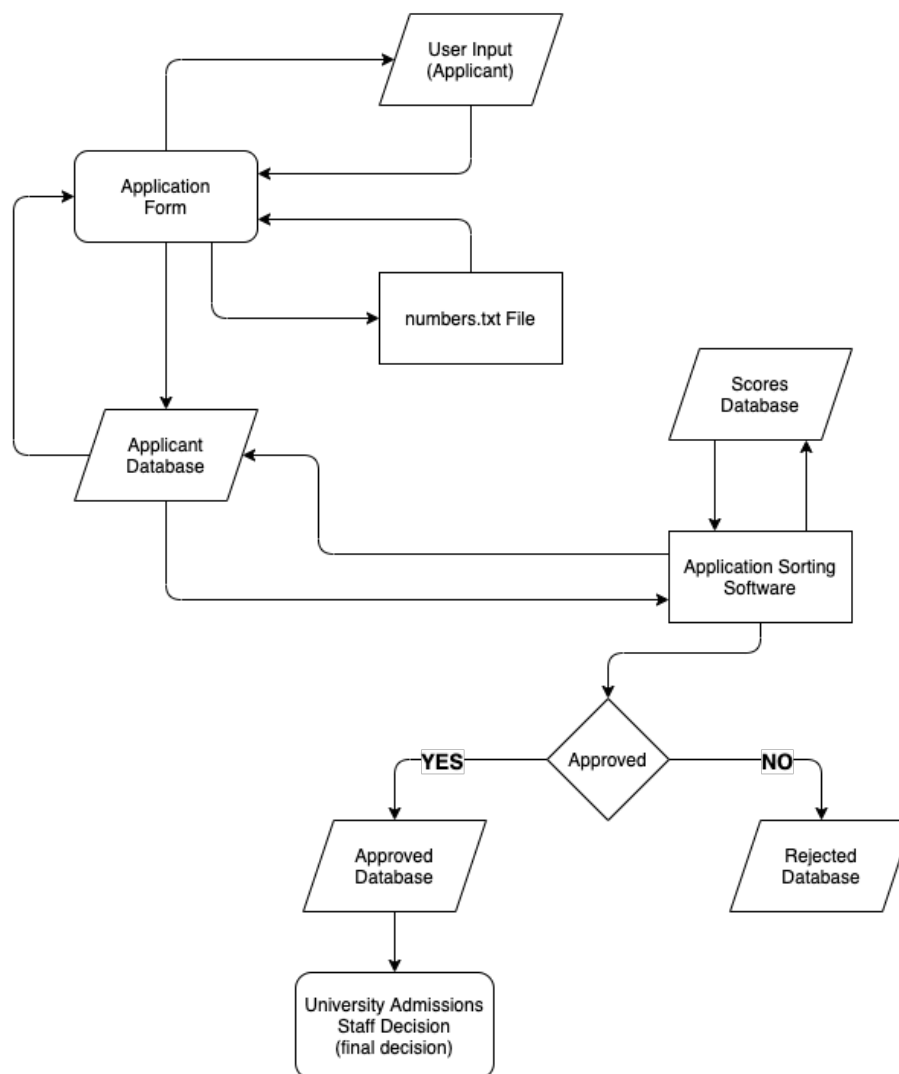
**Figure 5: Program Flowchart (Application Sorting Software)**

## Chapter 4 – System Implementation and Documentation

### 4.1 Introduction

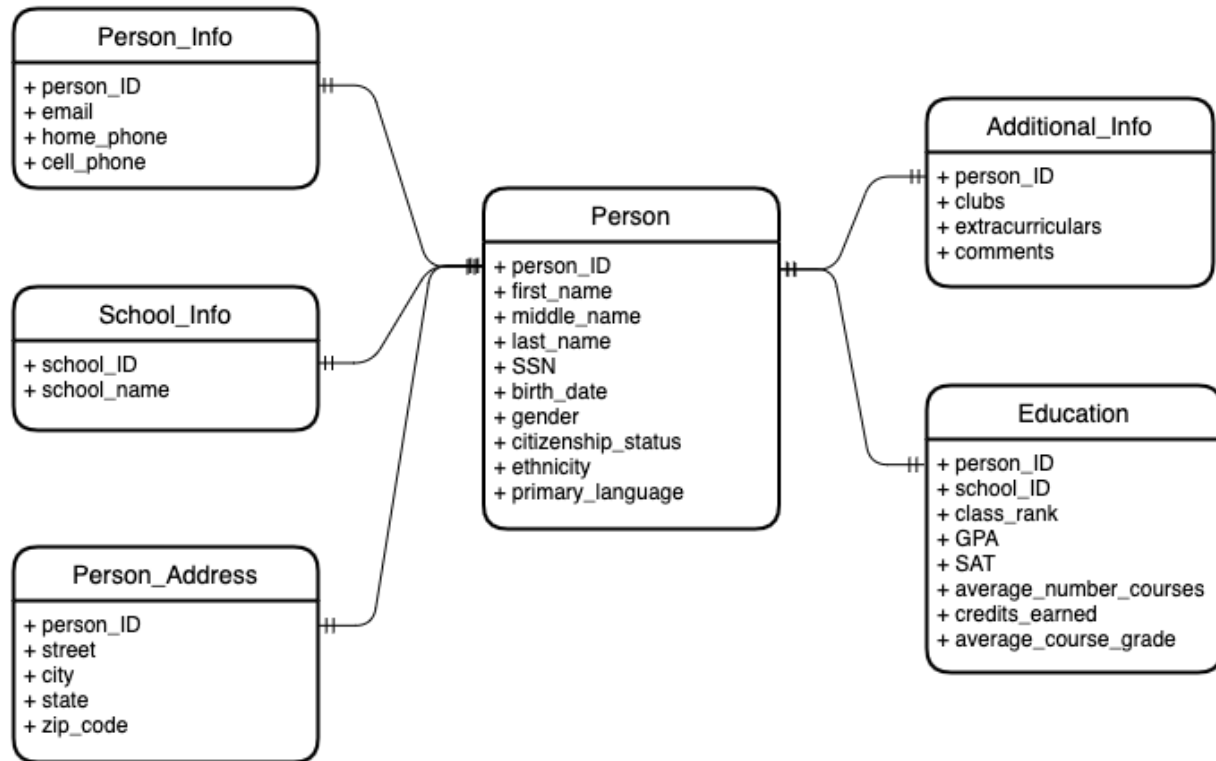
This chapter will discuss the specifics of how the system was implemented. It also contains several models and diagrams of the system. Additionally, it will discuss the source code, the environments used, and the programming language used to create the system. A very detailed description of the system can be found in the following sections.

### 4.2 System Design Diagram



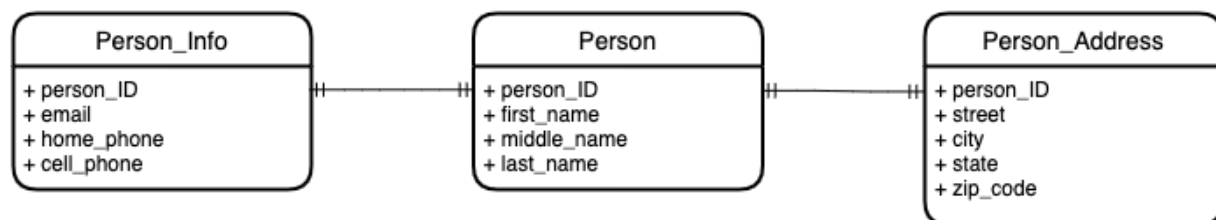
**Figure 6: System Design Diagram**

A detailed description of the steps in the above diagram can be found in section 3.4.



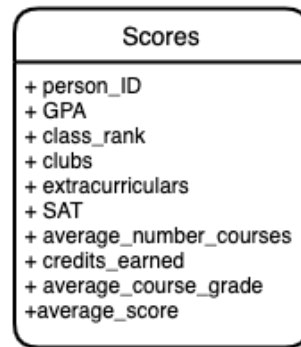
**Figure 7: Applicant Database Model**

The applicant database is used to store important information for each applicant. This information includes personal information, prior school information, and prior education. The information gathered can be viewed in detail in Figure 7 above.



**Figure 8: Approved and Rejected Database Models**

Both the Approved and Rejected databases contain the same tables and fields. The Approved database is used to store the contact information of each of the applicants that are accepted by the system. The Rejected database is used to store the contact information of each of the applicants that are not accepted by the system. These databases were created separately to make it easier for the university to search through and share information about different sets of applicants. Each database can be shared to another department, if necessary, without having to share all the data on each applicant. Only the necessary information (contact information) will be shared.



**Figure 9: Scores Database Model**

The Scores database contains each of the different fields that are used to determine which applicants are approved and which applicants are not approved. It also contains the overall average score. This information is stored in a database to make it easier for the system to search through each of the applicants and scores. Additionally, it is being stored in the database as a record of how the score was calculated. This could be very important in case of a later dispute or lawsuit against the college or university.

### **4.3 Choice of Programming Language**

The system (both the application form and the sorting software) were implemented using the Python programming language. This language was chosen for several reasons. To begin with, Python is well suited for numeric calculations and creating desktop GUIs (Python Software Foundation, n.d.). Python is also a good programming language for sorting data. Additionally, there are several useful modules that make python a little quicker and easier to use to create complex systems. These features of Python made it stand out from other similar programming languages as the best choice to create this system.

### **4.4 Analysis of Modules in Source Code**

#### **4.4.1 tkinter**

“The tkinter package (‘Tkinter Interface’) is the standard Python interface to the Tk GUI toolkit” (Python Software Foundation, 2021). It is used for creating GUI applications. It usually comes packaged with the Python download, so it is not necessary to download any new packages to use this module. It was used in this project to help create the GUI for the application form. It allowed for the creation of the logo, labels, textbox entry fields, drop-down menus, etc. An example of the application form can be seen in Figure 4 from section 3.5.

#### 4.4.2 pyautogui

“PyAutoGUI lets your Python scripts control the mouse and keyboard to automate interactions with other applications” (Sweigart, 2019). However, while this module has the ability to do quite a bit more, it is only being used to get the height and width of the screen. Getting the screen size allows the Application Form window to resize itself to be full screen no matter the screen size.

#### 4.4.3 sqlite3

“SQLite is a C library that provides a lightweight disk-based database that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language” (Python Software Foundation, 2021). It was used in this system to create, query, and insert information into all four of the system’s databases. SQLite3 can be used in the terminal as well as a module that can be imported into python.

#### 4.4.4 os.path

“This module implements some useful functions on pathnames” (Python Software Foundation, 2021). “Unlike a unix shell, Python does not do any *automatic* path expansion” (Python Software Foundation, 2021). Therefore, this Python module is being used to get the path of the current directory. The correct file name is then added to the directory path to obtain the full path of the appropriate file.

### 4.5 Description of Programming Environment

The operating system used to create this system was macOS. However, it should be able to run on any of the current major operating systems (macOS, Linux, and Windows). It was also implemented to be run on a single system. However, the system could be expanded to include a client-server architecture. This would allow applicants to fill out and submit the application form remotely. This could be accomplished by turning the current Application Form into a mobile application or a web application. The databases could then be stored on a server instead of personal machine. This would allow the remote applications to submit the forms to the database.

Visual Studio Code (VS Code) was also used to create this system. “Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS, and Linux” (Microsoft, 2021). This software was used to create and test both the Application Form and the Application Sorting Software.

Finally, the macOS terminal was also used to create, query, delete, and test the databases created in this system. To use SQLite3 with the terminal, go to the directory where the database is located and type “sqlite3 <name of database>”. This will open the database or create a new one if there is no database with that name. Once the database is open, use normal SQL syntax to make changes to or query the database.

## 4.6 System Implementation

Initial implementation included creating the Application Form. The form was created with the Python programming language and the tkinter module. The form includes the college or university logo, entry fields, drop-down menus, labels, etc. There are two pages to the form. The first page collects personal information, such as the applicants' name, ethnicity, citizenship status, contact information, etc. It also collects prior education information, such as GPA, SAT scores, course load information, etc. The second page collects the applicant's high school clubs, extracurricular activities, and any additional comments the applicant may have. Once the applicant has the first page filled out, he/she will click the submit button. The system will try to connect to the applicant database. If the connection fails, the system will print an error message. If the connection is successful, that page of information is added to the Applicant database, the applicant's ID number is added to an external file, and the form displays the second page of the application. This page then gets filled out and the submit button is pressed once again and that page of information is added to the Applicant database. The system checks the external file for the last ID number added and adds that ID to the database for the second page. This process prevents the applicant from having to enter the ID number twice. The layout of the Application Form can be seen in Figure 10 below.

A. Page One

B. Page Two

**Figure 10: Application Form**

The Applicant database was also created during the early stages of implementation. It was created by using the macOS terminal and SQLite3. This database must be created before the Application Form can work, so it must be created in order to test the Application Form. The Applicant database contains six tables and multiple fields in each table. It collects all the information provided from the Application Form. A model of this database can be seen in Figure 7 in section 4.2.

The next step was to create the Approved and Rejected databases. These databases are used to store information on the applicants that are accepted by the system and the applicants that are not accepted by the system respectively. They were created using the macOS terminal as mentioned above. These databases contain a subset of the information provided in the Applicant database. The purpose of these databases is to make it easier for the college or university to contact the applicants and notify them if they have been accepted or not accepted. It also allows the college or university to share the contact information of the applicants with other departments, if necessary, without sharing all the applicants' information. These databases could also be created

after the beginning stages of the implementation of the Application Sorting Software, but they must be created before that software can be fully tested.

Additionally, the Scores database should be created at this point in the project, or it could be created after the beginning stages of the implementation of the Application Sorting Software. It just needs to be created in order to fully test the Application Sorting Software. Like the previous databases, it was also created using the macOS terminal as mentioned above. This database is used to organize and store each of the values calculated to determine the applicants overall score as well as the overall score itself. It is to be kept as a record in case of a later dispute or in case of a lawsuit against the college or university.

Once the Application Form was created and the Applicant Database was created, that portion of the system was tested to ensure it was working correctly. Once it was confirmed that the software was working as it should and the information was being stored in the database correctly, the Application Sorting Software was then implemented. It was important to make sure the information was being stored in the database correctly first because the sorting software would not work correctly if the information in the database was not correct.

The Application Sorting Software was created to query the databases using the sqlite3 module in Python. It connects to all four databases (Applicant, Scores, Accepted, and Rejected databases). If the connection to any database fails, the system prints out an error message stating which connection failed. It then queries the Applicant database to obtain the applicants' information. The system then uses this information to calculate scores for each field. Scores are calculated as:

- **GPA:**
  - Score = GPA x 10
- **Class Rank:**
  - If rank is top 10:
    - Score = 10
  - Else if rank is top 50:
    - Score = 6
  - Else if rank is top 100:
    - Score = 3
  - Else:
    - Score = 1
- **Clubs:**
  - If applicant was in 10 or more clubs:
    - Score = 10
  - Else if applicant was in 5 or more clubs:
    - Score = 6
  - Else if applicant was in 3 or more clubs:
    - Score = 3
  - Else:
    - Score = 1
- **Extracurricular Activities:**
  - If applicant participated in 10 or more extracurricular activities:
    - Score = 10



- Else if applicant participated in 5 or more extracurricular activities:
    - Score = 6
  - Else if applicant participated in 3 or more extracurricular activities:
    - Score = 3
  - Else:
    - Score = 1
- **Average Number of Courses:**
  - If applicant had an average of 5 or more courses:
    - Score = 5
  - Else if applicant had an average of 4 or more courses:
    - Score = 4
  - Else if applicant had an average of 3 or more courses:
    - Score = 3
  - Else if applicant had an average of 2 or more courses:
    - Score = 2
  - Else:
    - Score = 1
- **Number of College Credits Earned:**
  - If applicant had earned 60 credits or more:
    - Score = 60
  - Else:
    - Score = number of credits earned
- **Average Course Grade (100-point scale):**
  - If applicant had an average course grade of 100 or more:
    - Score = 100
  - Else:
    - Score = average course grade
- **Overall Average Score:**
  - Score = sum of all previous scores divided by total possible points

These scores are then entered into the Scores database after they are calculated for each applicant. Once the scores for all the applicants have been entered into the database, the system queries the database for the applicants' ID number and the overall average score. It then adds these two values for each applicant into a dictionary. The system then sorts through the scores to determine which applicants are to be accepted by the system and which applicants are not to be accepted by the system. This is accomplished by the system selecting the applicants with the highest overall average scores. The system will chose the top \_\_ number of students, where the blank is determined by the college or university using the system. However, if there are multiple applicants with the same overall average score at that cutoff point, the system will accept all the applicants with that score. That means there could be a few extra students accepted by the system than the number supplied by the college or university. The accepted applicants' contact information is then added to the Approved database and the non-accepted applicants' contact information is added to the Rejected database.

Finally, the college or university admissions staff queries the Approved database to see the information for the applicants that have been approved by the system. They can also query the Scores database to see the scores calculated for each applicant, or the Applicant database to see

the exact information provided by the applicant or the applicants' comments. The admissions staff is then able to make the final decision as to which applicants get accepted and which applicants do not get accepted into the college or university.

## **Chapter 5 – Summary, Conclusion, and Recommendation**

### **5.1 Introduction**

This chapter provides a general discussion of the entire project. A discussion of the constraints of this study can also be found in this chapter. Finally, it will discuss what was accomplished by the project and also how it was accomplished.

### **5.2 Constraints of the Study**

This section discusses the constraints of this study. It breaks each constraint down into its own section. Each section will state what the constraint is and why it is a constraint in this particular project.

#### **5.2.1 Time Constraints**

This project was created and completed for the Computer Science Senior Seminar Course at The University of Virginia's College at Wise. This project had to be completed within a single semester (16 weeks). That included researching, planning, designing, implementing, testing, and creating this report of the project. It was a very limited amount of time for a project of this size. That limited the project to simpler software. Had the project been given more time, the GUI could have had a much better appearance, there could have been a client-server architecture created to allow the form to be filled out remotely, and some details of the project could have been improved with more planning and designing.

#### **5.2.2 Single Person Development Team**

This project only had a single person working on it. That one person had to do all the research, planning, designing, implementing, testing, and creating the report. When there are multiple people working on the same project, they can split up the amount of work needed to complete the project. This allows tasks to be completed much more quickly. It also helps create better designed and better tested software. When there is only one person working on a project, not as much can be accomplished as if there were a whole team of people working on it (especially when the project is already on a strict time-frame).

#### **5.2.3 Small Test Cases**

This project uses the application form software to gather information on all the applicants. The process of filling out all the required information for a large number of applicants is very time consuming. This time constraint combined with the tight schedule for this project limits the number of applicants that can be added to the system for testing. The system, however, was tested using different numbers of applicants, but they were all relatively small in number.

### 5.2.3 No Functionality for Remote use of Form

Ideally, this project would have made use of a client-server architecture which would have allowed the application form software to be used remotely. The databases and the application sorting software would have been stored on a server and the application form software would have ran on the remote devices (likely in the form of a web application or a mobile application). This design would have made the application process even easier on the applicants. The current design has all the software being stored on campus, which means the applicants must come to the college to fill out the application. Had the project been given more time, the improved client-server architecture design could have been implemented for this project.

## 5.3 Summary

This project's main goal was to try to find a solution to the problems faced by upcoming college students and college admissions staff. It is much easier for applicants to fill out an electronic form than it is for them to fill out a paper form (especially if the electronic form is available to be filled out and submitted remotely). An electronic application form and electronic application sorting software also make the admissions process much easier on the admissions staff. It can reduce the number of applicants that the admissions staff must sort through to a much more reasonable and manageable amount.

In an effort to solve this problem, a system was designed that would create an electronic application form and application sorting software. It also included multiple databases that stores the applicants' information as well as important information created by the application sorting software. Finally, the application sorting software was designed to sort through each of the applicants using the information stored in the database. It was designed to be able to select the top applicants from the entire pool of applicants.

The application sorting software was created using the python programming language. It included a GUI desktop application that would allow the applicants to fill in their information and submit it to the database. This application also checked the applicants' information to make sure it was valid before it submitted it to the database.

The application sorting software was able to get the information for each applicant and to calculate scores for each judging field from the application. The sorting software would then calculate an overall average score for each applicant. These scores were then added to a database. The software then queried that database for the specified number of top applicants. It then added the information for those applicants into another database. Finally, it added the information of the remaining applicants into a different database.

There were four databases created and used for this system (Applicant, Scores, Approved, and Rejected). The Applicant database was used to store the information provided by each applicant. The sorting software could query this database to obtain information for each applicant. The sorting software then calculated the scores for each applicant and store them in the Scores database. The sorting software then added the information of the accepted applicants to the

Approved database and the information of the applicants that were not accepted to the Rejected database.

Having multiple separate databases allowed the college or university to separate, store, and share information without having to send it all together. This would allow the admissions staff to share information with other departments in the college without having to share all the information. For example, if a certain department within the college wanted to reach out to the rejected applicants and tell them how they could improve and encourage them to apply again in the future, admissions could share only the Rejected database instead of all the information that was stored in all the databases.

## **5.4 Conclusion**

In conclusion, this project created a solution to the problem presented in the beginning of this report. That solution included a system that can accept electronic applications and sort them electronically as well. This system reduced the number of applicants that admissions staff must sort through to make their final decision and made the whole process fairer at the same time. Additionally, it allowed applicants to fill out an application electronically, which is preferred by many applicants in today's time.

The software created during this project was fully functional and worked as expected. All the information entered into application form was stored in the Applicant database successfully. The application sorting software was able to query the Applicant database for information on each applicant. It then calculated scores for each category of information and an average overall score for each applicant and stored those scores in the Scores database. It then queried the Scores database for the selected number of top applicants, and successfully stored the required information in the Approved database. It then successfully added the required information for the remaining applicants to the Rejected database.

Since the software was able to obtain applications electronically, sort those applications electronically, and make a decision on who should get selected and who should not get selected, it was a success. Additionally, the system was able to help make the admissions process easier on both the applicants and the admissions staff. Therefore, since this project created a successful solution to the problems, the project was a success.

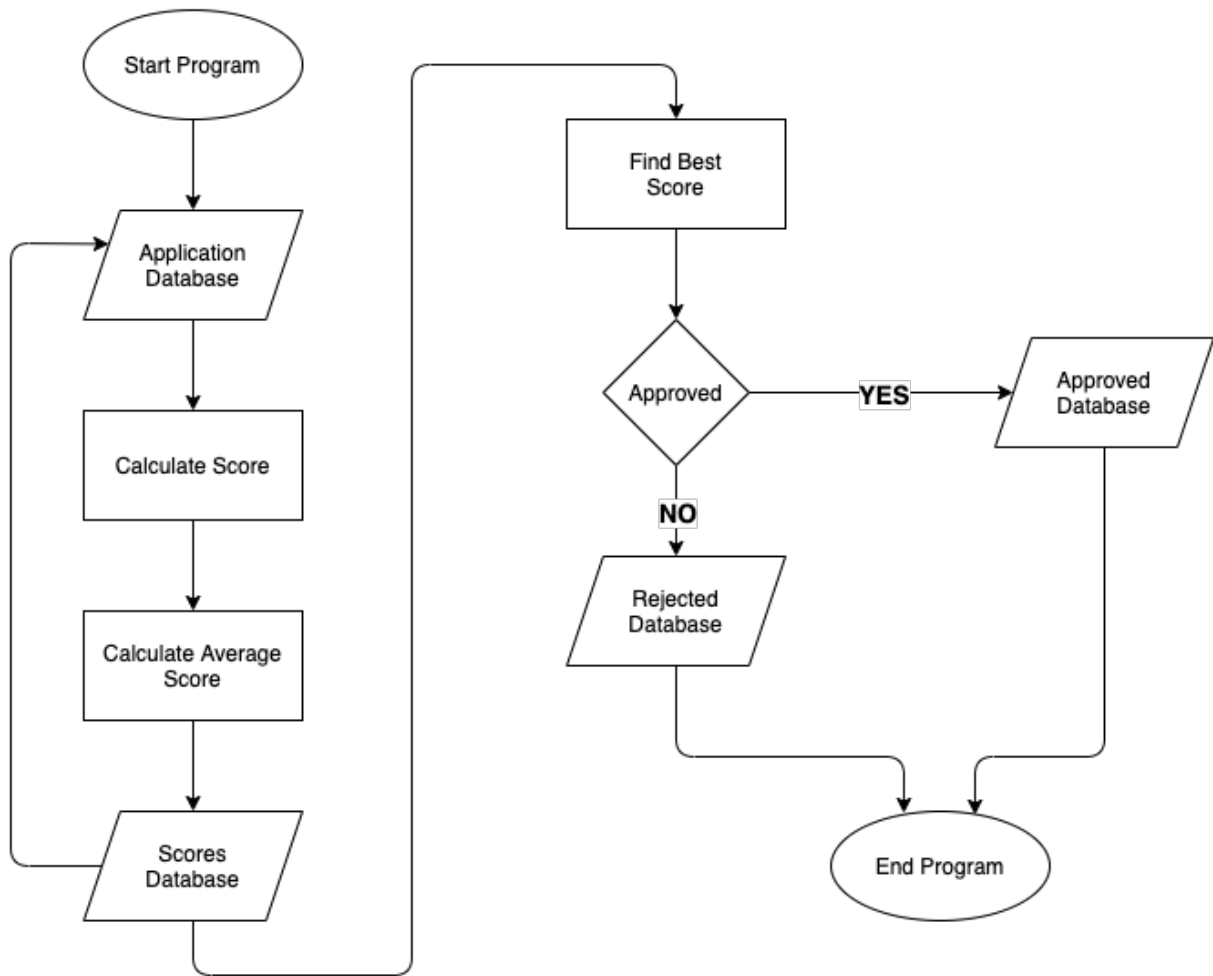
## **5.5 Recommendation**

Recommendations to others who may decide to work on this project further or who may work on a similar project would include making the application form available to be filled out remotely, fixing scaling problems, and allowing yourself enough time to thoroughly test the software. This project would have been more effective if it could have included the client-server architecture that would have allowed the application form to be remote. That remote form would prevent applicants from having to actually be at the college or university to submit their application.

There were also some problems with the application form software not scaling correctly on a Windows or Linux device. It would be better if the form would scale correctly to work on any operating system. It is also possible, however, that the problems are due to a large screen size and 5K resolution. It could also be because of the 4:3 aspect ratio of a Mac screen and a 16:9 ratio of the Windows and Linux screen. Additionally, this project could have been improved by having better and larger test cases. Small test cases are a good indication that the software is working correctly. However, to fully test the system, both large and small amounts of applicants need to be tested. For example, test cases should be created for five, ten, fifteen, or twenty people to begin with, but it should also be expanded to include hundreds or thousands of applicants. This may uncover errors that the smaller test cases do not uncover. This also helps to stress test the system to make sure it can handle and store large numbers of applicants. However, while these larger test cases are beneficial, it can consume a large amount of time to create them. Make sure to allow enough time in the schedule to create these larger test cases, or make sure to assign someone else on the team to that task to make sure it gets done within schedule.

## References

- Berg, K. L., Seymour, T., & Goel, R. (2012, December 31). History of Databases. *International Journal of Management & Information Systems*, 17(1), 29-36.
- Bound, J., Hershbein, B., & Long, B. T. (2009). Playing the Admissions Game: Student Reactions to Increasing College Competition. *J Econ Perspect*, 23(4), 119-146.
- Clinedinst, M., & Koranteng, A.-M. (2017). *2017 | State of College Admission*. National Association for College Admission Counseling (NACAC).
- Jaschik, S. (2019, March 13). Massive Admissions Scandal. *INSIDE HIGHER ED*, pp. 1-15.
- Microsoft. (2021). *Getting Started*. Retrieved March 2021, from Visual Studio Code: <https://code.visualstudio.com/docs>
- Python Software Foundation. (2021, March 2021). *os.path -- Common pathname manipulations*. Retrieved March 2021, from Python: <https://docs.python.org/3/library/os.path.html>
- Python Software Foundation. (2021, March 27). *tkinter - Python interface to Tcl/Tk*. Retrieved March 2021, from Python: <https://docs.python.org/3/library/tkinter.html>
- Python Software Foundation. (n.d.). *Applications for Python*. Retrieved March 2021, from Python: <https://www.python.org/about/apps/>
- Python Software Foundation. (2021, March 27). *sqlite3 -- DB - API 2.0 interface for SQLite databases*. Retrieved March 2021, from Python: <https://docs.python.org/3/library/sqlite3.html>
- Rigol, G. W. (2003). *Admissions Decision-Making Models How US Institutions of Higher Education Select Undergraduate Students*. College Entrance Examination Board.
- SQLite. (2021, January 20). *SQLite*. Retrieved March 2021, from SQLite: <https://www.sqlite.org/index.html>
- Sweigart, A. (2019). *Welcome to PyAutoGUI's documentation*. Retrieved March 2021, from Read the Docs: PyAutoGUI: <https://pyautogui.readthedocs.io/en/latest/>
- The University of Virginia's College at Wise. (n.d.). *APPLICATION FOR ADMISSION*. Retrieved March 2021, from UVA WISE: [https://www.uvawise.edu/sites/default/files/attachments/pages/2020-10/UVAWISE\\_Application\\_Final.pdf](https://www.uvawise.edu/sites/default/files/attachments/pages/2020-10/UVAWISE_Application_Final.pdf)
- Vohra, R., & Das, N. N. (2011, October). Intelligent Decision Support Systems for Admission Management in Higher Education Institutes. *International Journal of Artificial Intelligence & Applications (IJAlA)*, II(4), 63-70.

**Appendix A: Program Flowchart (Application Sorting Software)**



## Appendix B: Source Code Listing

The entire source code files can be found in the following GitHub repository:

<https://github.com/cas5ga/Computer-Science-Seminar>

### Sample Application Form Software Code:

```
#Main function
screenWidth, screenHeight = pyautogui.size()
backgroundColor = "red3"
foregroundColor = "white"
labelColor = "gray45"

#creates the window
window = tkinter.Tk()
window.title("College Application Form")
window.geometry(str(screenWidth)+"x"+str(screenHeight))
window.configure(background = "light gray")

#.....
#Left out a large section of the code here
#Missing code can be found on the GitHub site listed at the top of this appendix
#.....

#Second Form
pageLabel4 = tkinter.Label(window, text = "Additional Information", width = (int(0.04
* screenWidth)), font = ("bold", int(0.014*screenHeight)), bg = labelColor, fg =
foregroundColor)

#first row label fields (additional information section)
clubLabel = tkinter.Label(window, text = "Clubs (one per line): ", width = (int(0.025
* screenWidth)), font = ("bold", int(0.0112*screenHeight)), bg = "light gray")
extracurricularLabel = tkinter.Label(window, text = "Extracurricular Activities (one
per line): ", width = (int(0.025 * screenWidth)), font = ("bold",
int(0.0112*screenHeight)), bg = "light gray")
commentLabel = tkinter.Label(window, text = "Additional Comments: ", width =
(int(0.025 * screenWidth)), font = ("bold", int(0.0112*screenHeight)), bg = "light
gray")

#first row entry fields (additional information section)
clubEntry = tkinter.Text(window, width = (int(0.025 * screenWidth - 1)), height =
(int(0.035 * screenHeight)), font = ("bold", int(0.0084*screenHeight)))
extracurricularEntry = tkinter.Text(window, width = (int(0.025 * screenWidth - 1)),
height = (int(0.035 * screenHeight)), font = ("bold", int(0.0084*screenHeight)))
commentEntry = tkinter.Text(window, width = (int(0.025 * screenWidth - 1)), height =
(int(0.035 * screenHeight)), font = ("bold", int(0.0084*screenHeight)))
```

```

pageLabel5 = tkinter.Label(window, text = "Submit and finish:", width = (int(0.04 *
screenWidth)), font = ("bold", int(0.014*screenHeight)), bg = "light gray")

submitButtonTwo = tkinter.Button(window, text = "Submit", command = submitSecondForm,
padx = int(0.01 * screenWidth), pady = int(0.005 * screenWidth))

#error message for input validation
errorMessageLabel = tkinter.Label(window, text = "Please correct all errors marked
with red labels", width = (int(0.015 * screenWidth)), font = ("bold",
int(0.014*screenHeight)), bg = "light gray", fg = "red")

displayFirstForm()

window.mainloop()

```

### Sample Application Sorting Software Code:

```

#loops through each applicant
for row in rows:
    #calculates the GPA score
    #total points: 40
    GPAScore = int(float(row[1]) * 10)

    #calculates the rank score
    #total points: 10
    rankScore = int(row[2])
    if(rankScore <= 10):
        rankScore = 10
    elif(rankScore <= 50):
        rankScore = 6
    elif(rankScore <= 100):
        rankScore = 3
    else:
        rankScore = 1

    #calculates the club score
    #total points: 10
    clubValues = row[3]
    clubList = clubValues.split("\n")
    numberOfClubs = len(clubList)
    if(numberOfClubs >= 10):
        clubScore = 10
    elif(numberOfClubs >= 5):
        clubScore = 6
    elif(numberOfClubs >= 3):
        clubScore = 3
    else:

```

```
clubScore = 1

#calculates the extracurricular score
#total points: 10
extracurricularValues = row[4]
extracurricularList = extracurricularValues.split("\n")
numberOfExtracurriculars = len(extracurricularList)
if(numberOfExtracurriculars >= 10):
    extracurricularScore = 10
elif(numberOfExtracurriculars >= 5):
    extracurricularScore = 6
elif(numberOfExtracurriculars >= 3):
    extracurricularScore = 3
else:
    extracurricularScore = 1

#calculates the SAT score
#total points: 16
SATScore = int(int(float(row[5]) / 100))

#calculates the average number of courses score
#total points: 5
coursesValues = int(float(row[6]))
if(coursesValues >= 5):
    coursesScore = 5
elif(coursesValues >= 4):
    coursesScore = 4
elif(coursesValues >= 3):
    coursesScore = 3
elif(coursesValues >= 2):
    coursesScore = 2
else:
    coursesScore = 1

#calculates the credits earned score
#total points: 60
creditsValues = int(row[7])
if(creditsValues >= 60):
    creditsScore = 60
else:
    creditsScore = creditsValues

#calculates the average course grade score
#total points: 100
courseGradeValues = int(float(row[8]))
if(courseGradeValues >= 100):
    courseGradeScore = 100
else:
```

```

        courseGradeScore = courseGradeValues

    #calculates the overall average score
    total = GPAScore + rankScore + clubScore + extracurricularScore + SATScore +
coursesScore + creditsScore + courseGradeScore
    averageScore = (total / totalNumberOfPoints) * 100

    values = scoresConnection.execute('''SELECT person_ID FROM Scores''').fetchall()

    #determines if the provided person_ID is already in the Scores table
    flag = False
    for value in values:
        #sets the flag to true if the person_ID is in the table
        if(value[0] == row[0]):
            flag = True

    #adds the new information into the Scores table if it is not already in the table
    if(not flag):
        #adds each of the scores to the Scores database
        scoresConnection.execute('''INSERT INTO Scores (person_ID, GPA, class_rank,
clubs, extracurriculars, SAT, average_number_courses, credits_earned,
average_course_grade, average_score) \
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)''', (row[0], GPAScore, rankScore,
clubScore, extracurricularScore, SATScore, coursesScore, creditsScore,
courseGradeScore, averageScore))
        scoresConnection.commit()

#gets the top "schoolCapacity" applicants from the database
rows = scoresConnection.execute('''SELECT person_ID FROM Scores ORDER BY average_score
DESC LIMIT ''' + str(schoolCapacity)).fetchall()

for row in rows:
    values = approvedConnection.execute('''SELECT person_ID FROM Person''').fetchall()

    #determines if the provided person_ID is already in the Person table
    flag = False
    for value in values:
        #sets the flag to true if the person_ID is in the table
        if(value[0] == row[0]):
            flag = True

    #adds the new information into the appropriate table if it is not already in the
table
    if(not flag):
        #adds the information into the Person table

```

```
        PersonValues = applicantConnection.execute('''SELECT person_ID, first_name,
middle_name, last_name FROM Person WHERE person_ID = '' + str(row[0])).fetchall()
        for value in PersonValues:
            approvedConnection.execute('''INSERT INTO Person (person_ID, first_name,
middle_name, last_name) VALUES (?, ?, ?, ?)''', (value[0], value[1], value[2],
value[3]))

        #adds the information into the Person_Info table
        Person_InfoValues = applicantConnection.execute('''SELECT * FROM Person_Info
WHERE person_ID = '' + str(row[0])).fetchall()
        for value in Person_InfoValues:
            approvedConnection.execute('''INSERT INTO Person_Info (person_ID, email,
home_phone, cell_phone) VALUES (?, ?, ?, ?)''', (value[0], value[1], value[2],
value[3]))

        #adds the information into the Person_Address table
        Person_AddressValues = applicantConnection.execute('''SELECT * FROM
Person_Address WHERE person_ID = '' + str(row[0])).fetchall()
        for value in Person_AddressValues:
            approvedConnection.execute('''INSERT INTO Person_Address (person_ID,
street, city, state, zip_code) VALUES (?, ?, ?, ?, ?)''', (value[0], value[1],
value[2], value[3], value[4]))

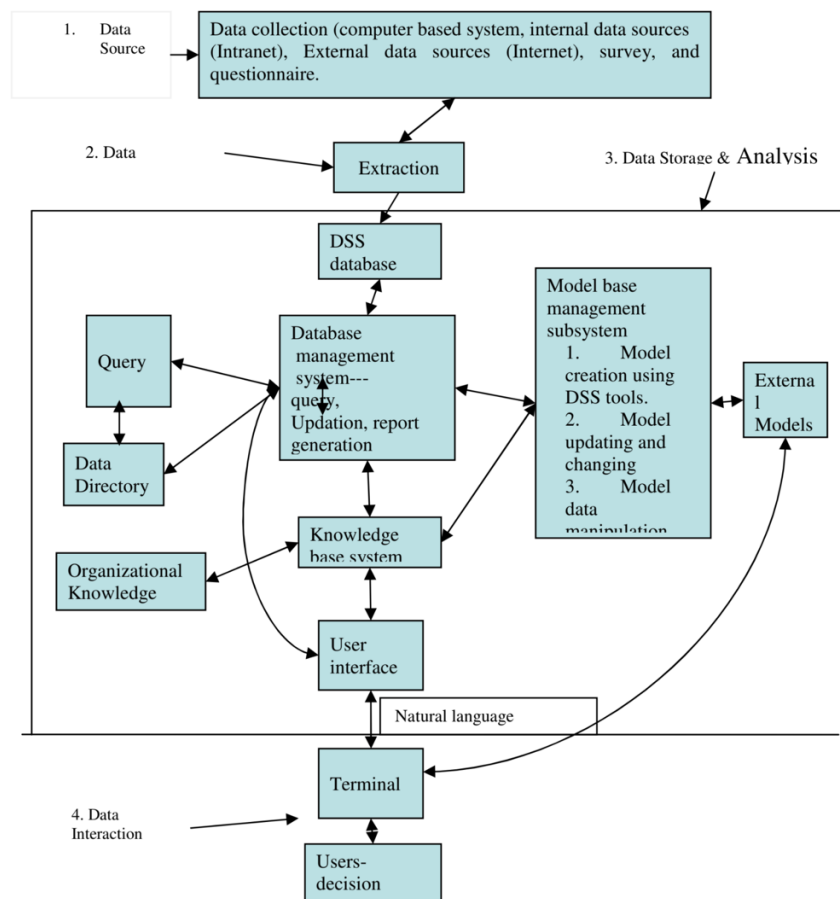
        approvedConnection.commit()
```

## Appendix C: Figures


**Table 1: Decisional Situations (Vohra & Das, 2011)**

DECISIONAL  
SITUATION

Students' enrollment	Faculty
Choosing a Specialization (Course/ Branch)	Scholarship
Studies reclassification	Career guidance
Recognition	Web Information and announcements
Tuition fees	Hostel facility
Ranking of college	Transportation
Infrastructure	Library
Interruption of studies(BUNK)	Extension of studies
Tutorial activities	Parking facility
Health facility	Research facility



**Figure 1: Schematic View of DSS (Vohra & Das, 2011)**



OFFICE OF ADMISSIONS  
The University of Virginia's College at Wise  
1 College Avenue • Wise, VA 24293-4412

E-mail: admissions@uvawise.edu  
Phone: (888) 282-9324 (toll-free)  
(276) 328-0102 (local)  
Web: www.uvawise.edu  
Fax: (276) 328-0251

**APPLICATION FOR ADMISSION**

**PERSONAL INFORMATION**  
TYPE OF STUDENT *(Check one.)*

☐ NEW FRESHMAN  
 ☐ NEW FRESHMAN WITH COLLEGE CREDIT  
 ☐ TRANSFER (FROM ANOTHER COLLEGE)  
 ☐ NONDEGREE  
☐ READMISSION *(for students previously enrolled in a degree program here)*  
 ☐ TEACHER CERTIFICATION *(only for students who have obtained a bachelor's degree, please submit official transcripts from all colleges attended)*  
 ☐ DUAL ENROLLMENT *(for students applying to take courses while in high school)*

**NAME** *(Please print.)*

FIRST \_\_\_\_\_ MIDDLE \_\_\_\_\_ LAST \_\_\_\_\_

PREFER TO BE CALLED \_\_\_\_\_ BIRTH DATE *(Month/Day/Year)* \_\_\_\_\_ GENDER: \_\_\_\_\_

SOCIAL SECURITY NUMBER *(For reporting purposes only)* \_\_\_\_\_

ARE YOU MARRIED? \_\_\_\_\_ ☐ YES ☐ NO   IF YES, MAIDEN NAME? \_\_\_\_\_

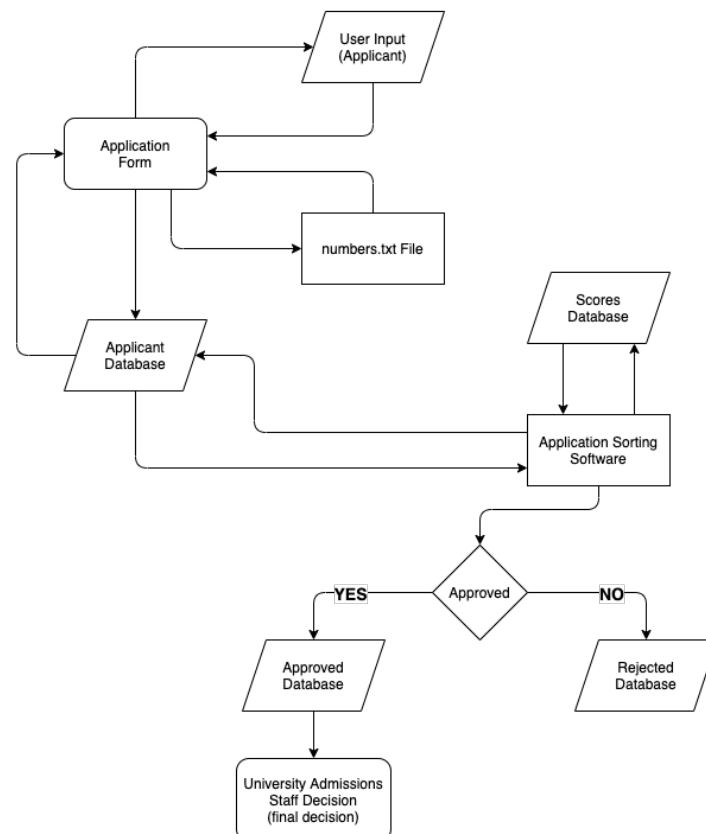
MAILING ADDRESS *(Not physical address)* \_\_\_\_\_ APT. NO. \_\_\_\_\_

CITY/STATE \_\_\_\_\_ COUNTY \_\_\_\_\_ ZIP CODE \_\_\_\_\_

HOME PHONE *(Area code)* \_\_\_\_\_ CELL PHONE *(Area code)* \_\_\_\_\_

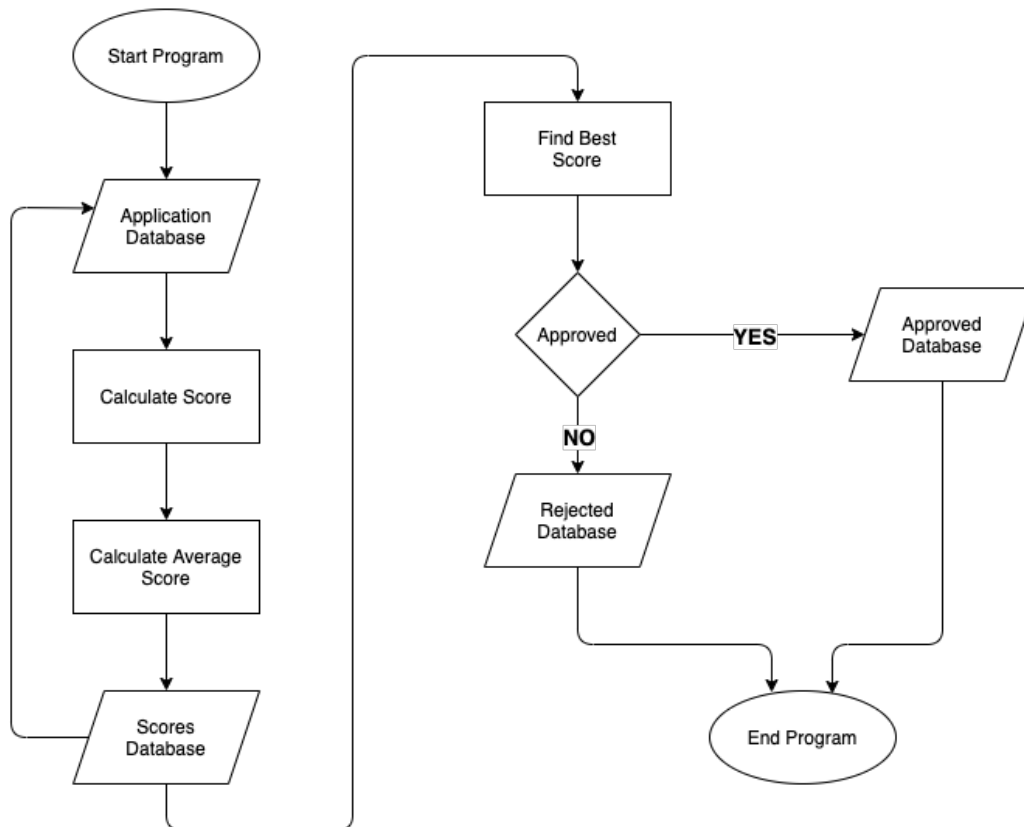
E-MAIL \_\_\_\_\_

**Figure 2: UVA WISE Paper Application Form**  
(The University of Virginia's College at Wise)



**Figure 3: System Flowchart**

**Figure 4: Electronic Application Form Example**



**Figure 5: Program Flowchart (Application Sorting Software)**



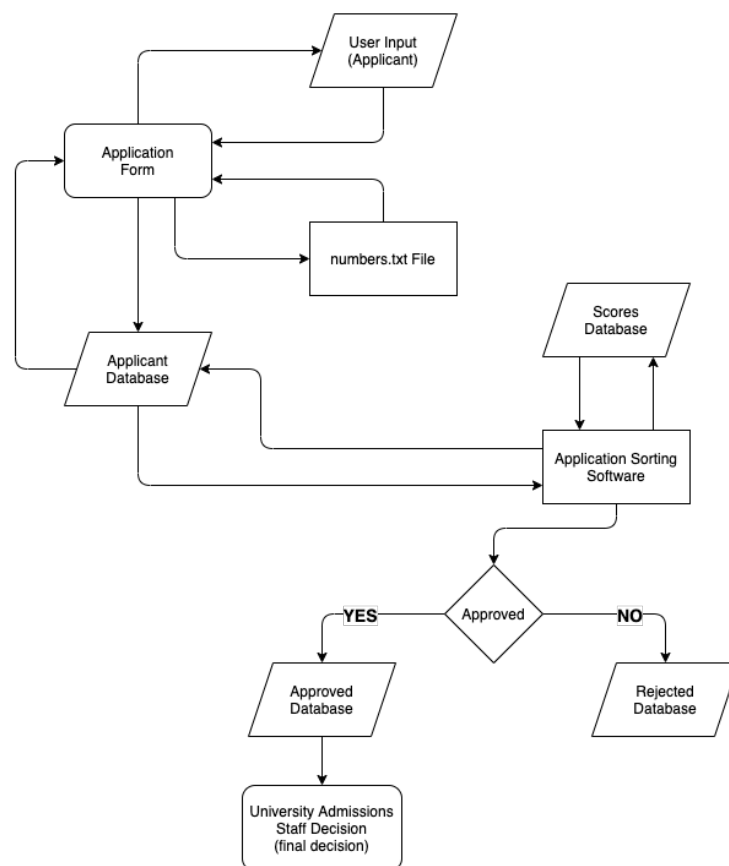


Figure 6: System Design Diagram

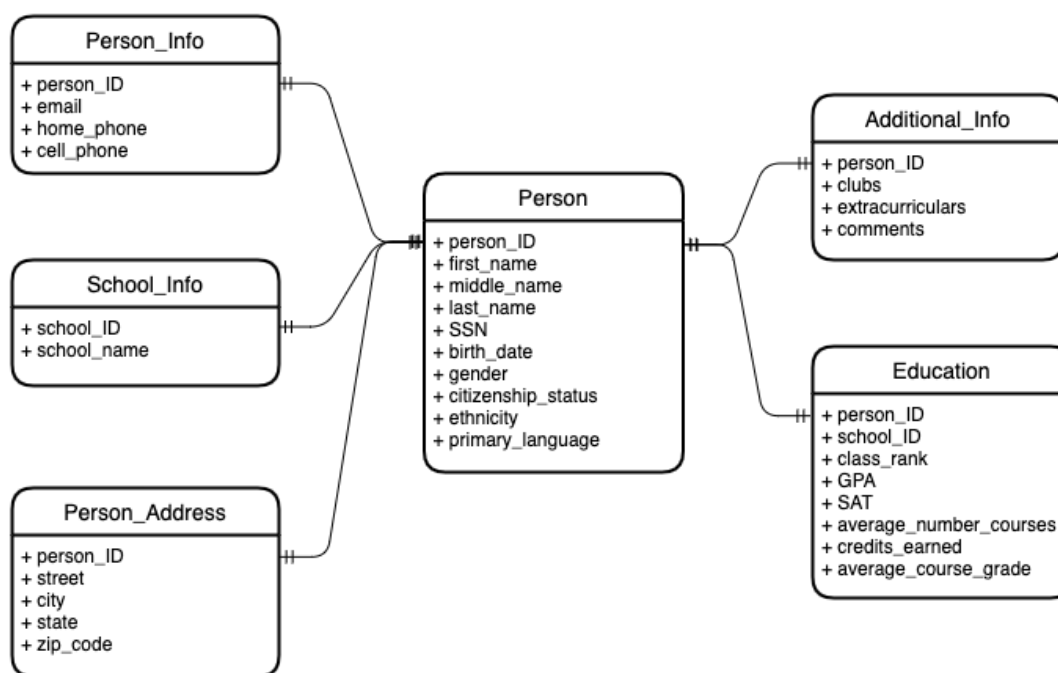
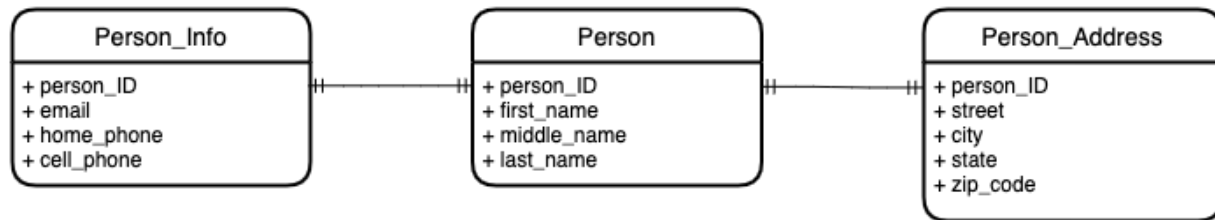
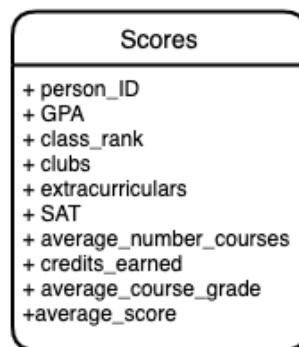


Figure 7: Applicant Database Model



**Figure 8: Approved and Rejected Database Models**



**Figure 9: Scores Database Model**

A. Page One

B. Page Two

**Figure 10: Application Form**