You should produce a final project report. That report ***must*** *include the final version of your project:*

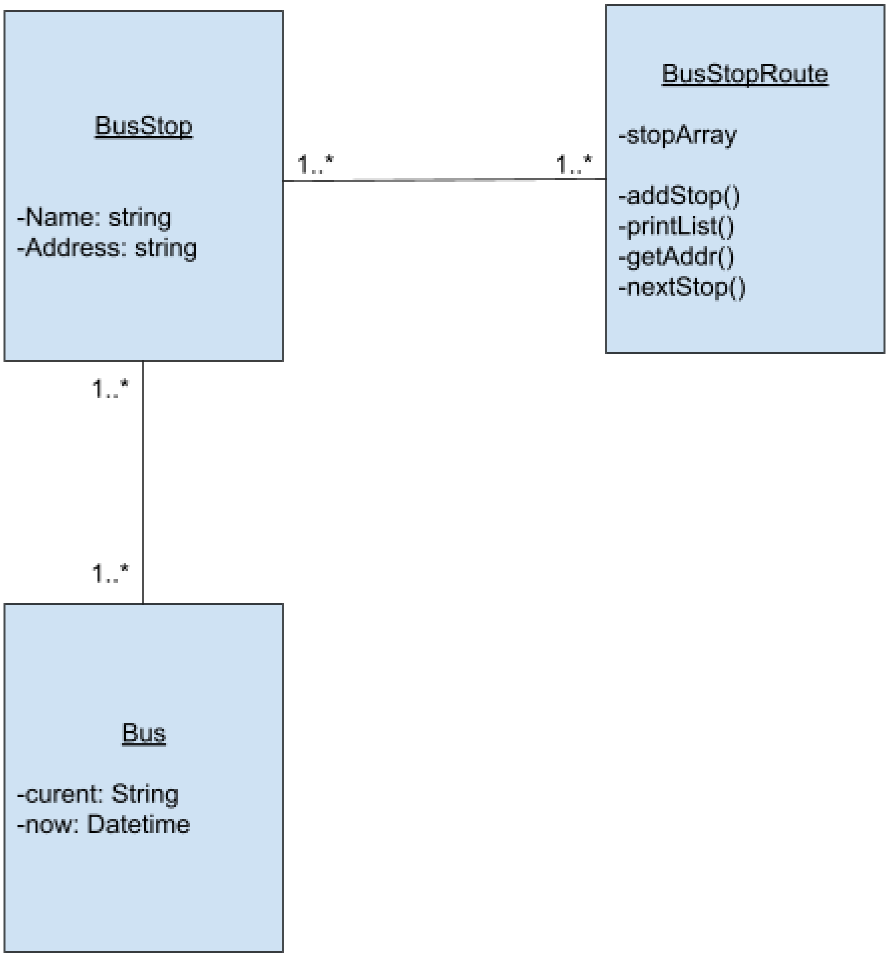
- functional/nonfunctional requirements table
- at least the UML sequence, activity and class diagrams
- system architecture diagrams.

Requirements Table

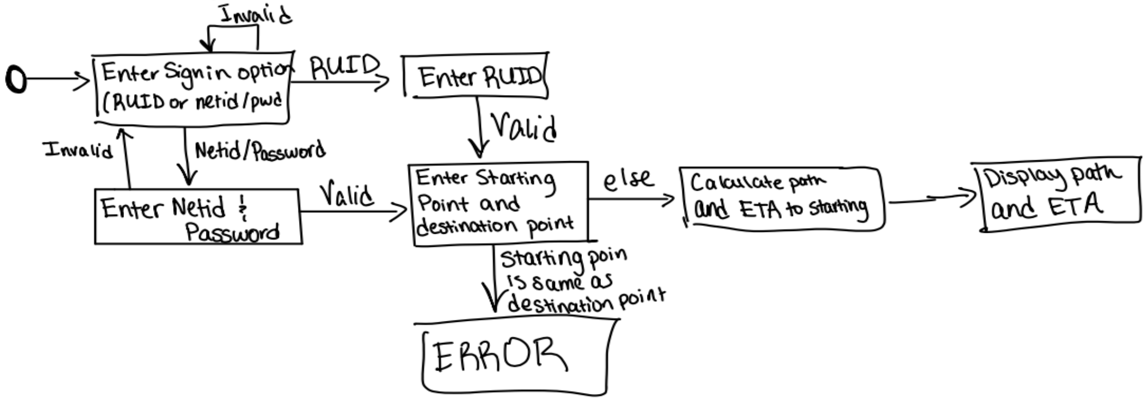
Requirements Description	Requirement Type
The user should be able to view all buses in route chosen via Google API map	Functional
The user should be able to view the ETA of the buses in route chosen	Functional
The user should be able to set an alert that notifies them when a bus is within 5 minutes of arrival	Functional
The user should be able to input their starting point location and the destination bus stop they choose	Functional
Utilizes the Google API	non-functional
To prevent any hacking into the system, authorization via netid/password or RUID is required before using system	non-functional
In the event of an accident or anything that causes route blockage or danger to any drivers or passengers, the system should notify them.	non-functional
The system should be able to withstand immense throughput from the users and execute in a timely manner.	non-functional
The system should regulate bus scheduling to be at most 30 min apart per bus per route	non-functional
Users need to authenticate themselves by signing in with netid and password or RUID number.	non-functional

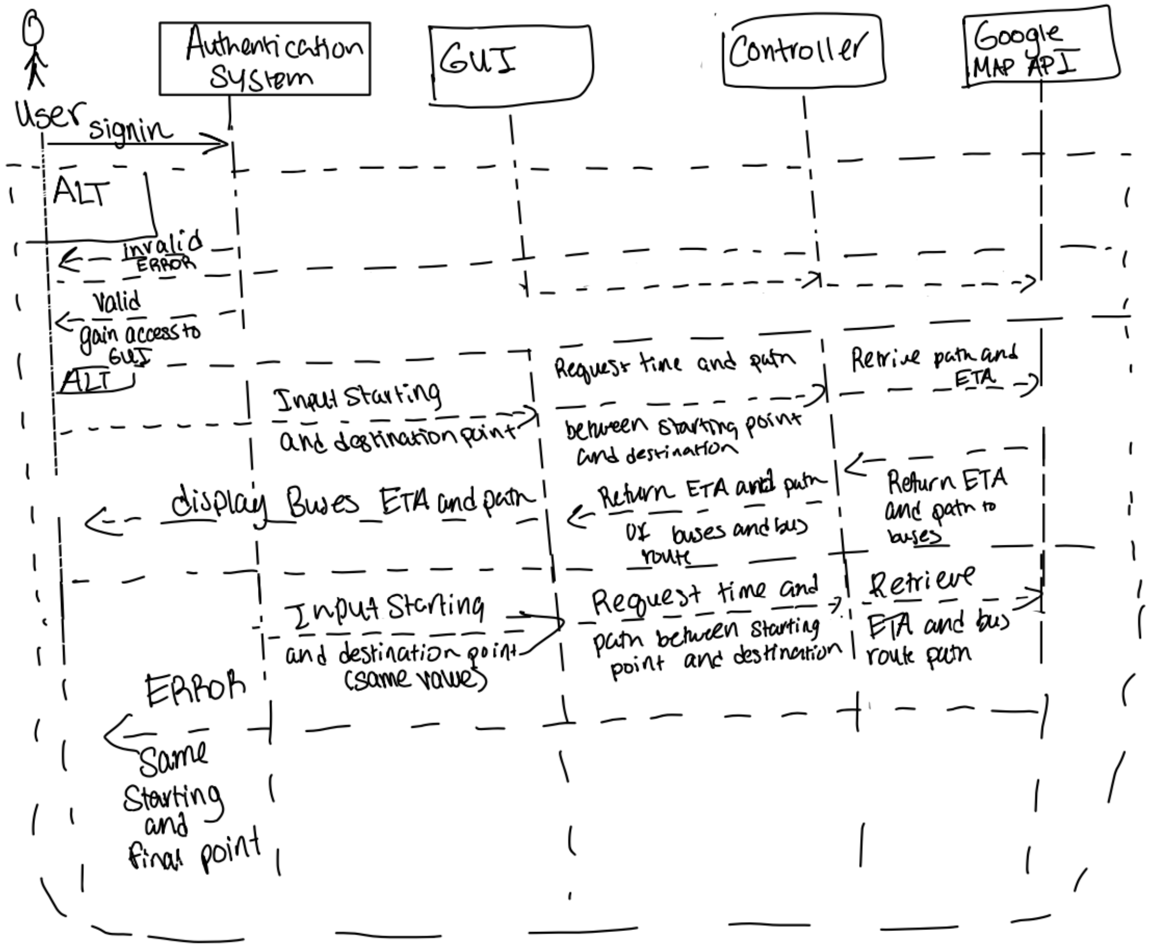
UML Diagrams

Class diagram

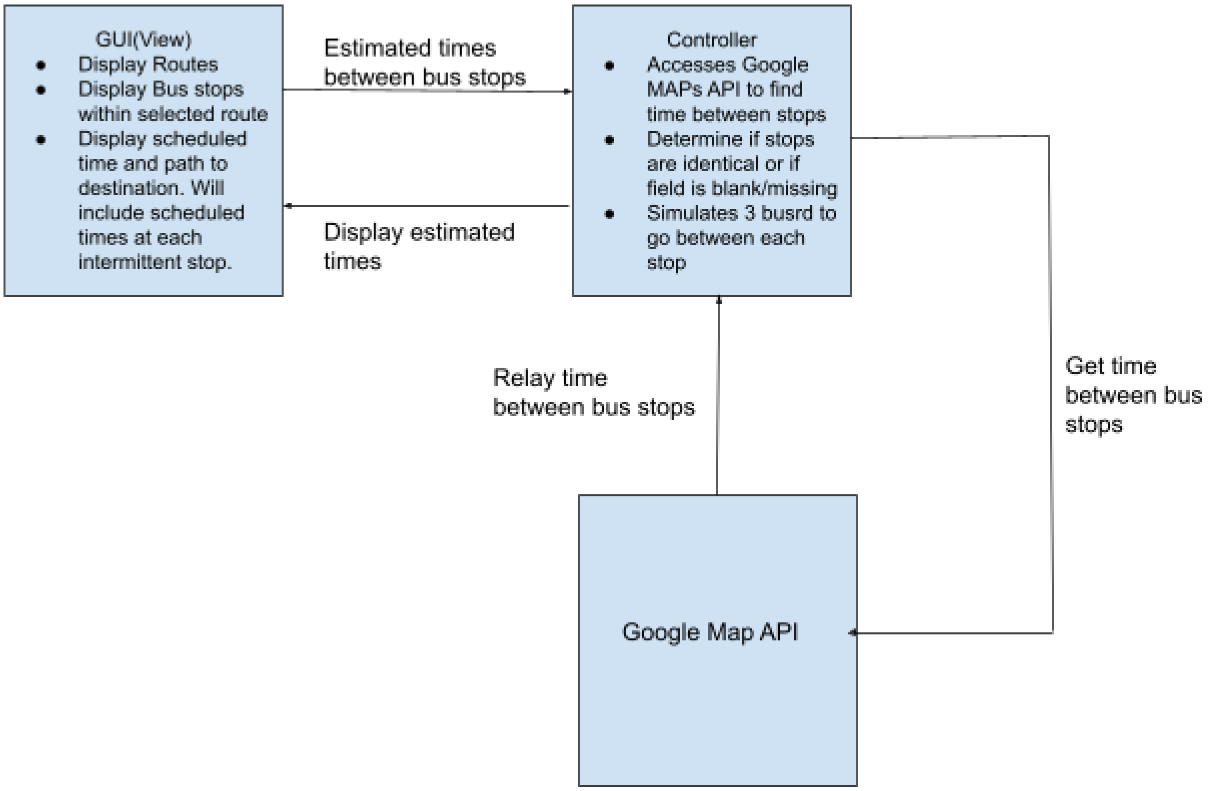


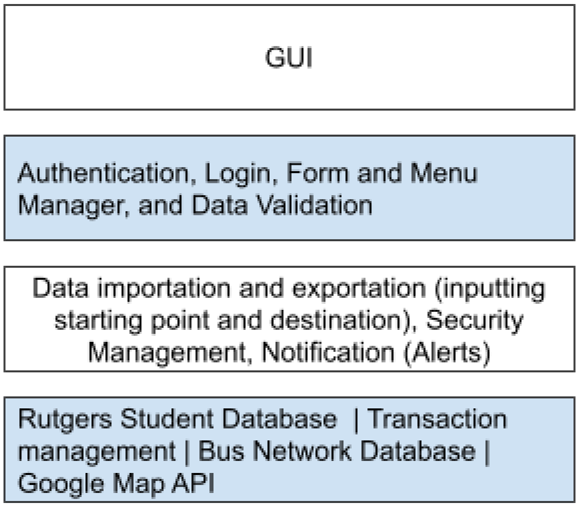
Activity diagram


Invalid
Enter Signin optio (RUID or netid/pwd
RUID
Enter RUID
Valid
Invalid
Netid/Password
Enter Netid & Password
Valid
Enter Starting Point and destination point
else
Calculate path and ETA to starting
Display path and ETA
Starting poin is same as destination point
ERROR


User
Authentication System
GUI
Controller
Google MAP API
signin
ALT
invalid ERROR
Valid gain access to GUI
ALT
Input Starting and destination point
Request time and path between starting point and destination
Retrive path and ETA
Return ETA and path to buses
Return ETA and path of buses and bus route
display Buses_ETA and path
Input Starting and destination point (same value)
Request time and path between starting point and destination
Retrieve ETA and bus route path
ERROR
Same Starting and final point

Architecture Diagrams


GUI(View)
Display Routes
Display Bus stops within selected route
Display scheduled time and path to destination. Will include scheduled times at each intermittent stop.
Estimated times between bus stops
Display estimated times
Controller
Accesses Google MAPs API to find time between stops
Determine if stops are identical or if field is blank/missing
Simulates 3 busrd to go between each stop
Relay time between bus stops
Get time between bus stops
Google Map API


GUI
Authentication, Login, Form and Menu Manager, and Data Validation
Data importation and exportation (inputting starting point and destination), Security Management, Notification (Alerts)
Rutgers Student Database | Transaction management | Bus Network Database | Google Map API

The report should describe and support your design, specification and implementation choices, outlining how you integrated principles of dependability, project management and advanced software engineering into your project. Finally, the report should contain a description of your testing and evaluation processes.

Our project Rutgers RoutePlanner System required us to design a student bus route management system that displays the time of arrival and path between two bus stops at Rutgers University. The user of this app should not have to wait more than 30 minutes to travel between two bus stops for class, events, etc. While doing so we were also instructed to make assumptions about budget, the time between campuses, frequency etc. when building this system. We used an agile test-driven and component based development approach when building this system. We used this to mitigate any future risks of having unknown errors or failure causes. Using a similar approach as our last project, we continued to use a GUI utilizing the tkinter module in Python and the Google Maps API utilizing the distance matrix. We built the system sort of layer by layer. First we implemented an authentication component to ensure dependability of the system via terminal by either inputting a Rutgers netid/password or RUID. Since the bus system is designed for Rutgers' students we saw it best fit to add this layer of security to our system so buses can remain available to as many students as possible. After that we implemented the skeleton of the GUI which would accept user inputs (starting point and destination). When constructing the GUI we reused the code we created from the previous project and adjusted it to fit the specifications of the system. Then finally, we reused the Google Map API distance matrix code to add functionality to our GUI. Throughout the time span of the first sprint to the deadline we kept track of any potential risks that we may encounter whether it pertains to the project or product. We kept constant and honest communication with each other pertaining to our workload and dependency to complete assigned tasks while being inclusive to each other's ideas during our sprints and motivating each other when needed during the last month of the semester. As to product and technology risks, we researched the Google Map API to see if it was compatible with Python, which was our preferred programming language to create the system with and what functionality it contains in the event where we could reuse the code instead of hard coding. To test the security and authentication of the system we created a list of RUIDs and a list of tuples (netid, password) and asked for the user to either input an RUID or netid/password to access the system/GUI. We used different test cases to ensure that faulty or non-existent RUID's or netid/password pairs gained access to the system. To test for the functionality of the system we inputted test cases to find errors such as inputting the same value for starting point and endpoint. Overall for our testing, we used component-based and unit-based testing and testing for any validations or defects through various use-cases.