TASK 2

- Since each visitor registration system may have its own API, authentication method and technology, the architecture software I would develop is the “event-driven architecture”. The event-driven architecture helps manage this by building a central unit that accepts all data and then delegates it to the separate modules that handle the particular type. This handoff is said to generate an “event,” and it is delegated to the code assigned to that type. In opposition to layered architecture where all data will typically pass through all layers, at event-driven architecture modules interact only with the events that concern them, they scale, adapt and extend to new event types easily. So each time a visitor uses a specific API or authentication method and technology there will be a specific event which will deal with it accordingly.

- We will occasionally have to store credentials somewhere for when some of these visitor registrations need authentication credentials to retrieve data. these plain text in the applications source code doesn't provide any true security since it isn't too much hassle to reverse engineer a program. Gathering the credentials from a server won't do the trick either since hackers easily can perform requests themselves. Neither will encryption of the stored credentials make any difference since the application will need access to the decryption key to be able to use the credentials at all. I would store the credentials on a user’s machine associated with that user, e.g., credentials that enable that user to log into his/her account.

- I believe the best way to deal with this kind of problem is to have certain monitor tools, such as Exhaustive logging to analyze a failure allowing us to discover where the problem lies. Also we could do integration testing before unit testing, the later you discover a defect in the development cycle, the more expensive it is to fix.