Alex Strickland
CS 509

Final

Github link: https://github.com/strickland77/Final-CS-509

Codecov: https://app.codecov.io/gh/strickland77/Final-CS-509

AppVeyor: https://ci.appveyor.com/project/strickland77/final-cs-509

Software architecture:

I use a layered architecture for this ATM software application. It has 3 layers including user interface (UI), business logic, and data abstraction layer (DAL). The UI handles the aspects of the program the user interacts with such as input handling and user menus. The business logic contains functionality for the types of users and the main program execution. The DAL takes care of interactions with the database such as creating a connection, logging in, and retrieving user data.

VMs vs Docker vs Computer:

Each have their respective pros/cons and ideal situations to use each. I personally have experience using all three for software development throughout my academic and professional career. With some exception and a generalization of the comparison, I believe VMs are the best option in a production environment, Docker is best for continuous development, and a plain old computer is great for one off programs/small projects.

VMs are in my opinion the best choice to use in a production environment for a few reasons. They are able to run different OS's to support any applications that may be needed with no additional hardware required. Each instance of a VM is isolated from one another ensuring separation of resource allocation (CPU cores, memory, storage). This also improves security preventing access to all instances if only one VM is breached. They are extremely portable across machines making deployment and scaling relatively easy if production needs grow. VMs can also have snapshots taken for easy backup and restore or recovery after system failure. Most VM software also has functionality to manage and maintain an entire cluster or group of VMs with ease.

Using VMs does have some downsides that make them less than ideal for all situations. For one, they are more complicated to setup and manage than Docker containers or using a computer. They can often require a skilled professional to utilize properly if it is a complex and large system. Another downside is most of the best VM software requires paid licensing to use often only afforded by companies. Given these downsides, they could still be used at small scale or even a single instance for free. Meaning they are still an option for active development or one off personal projects if desired.

Docker is a phenomenal option for a continued development setting. One benefit being it is cross platform and can therefore be run on basically any system. This make it convenient if you or your team are constantly switching and working on different machines. Docker containers/images are also extremely lightweight and easy to use. It takes seconds to load an image and run a container making deployment and scalability even easier than VMs. They also take up significantly way less system

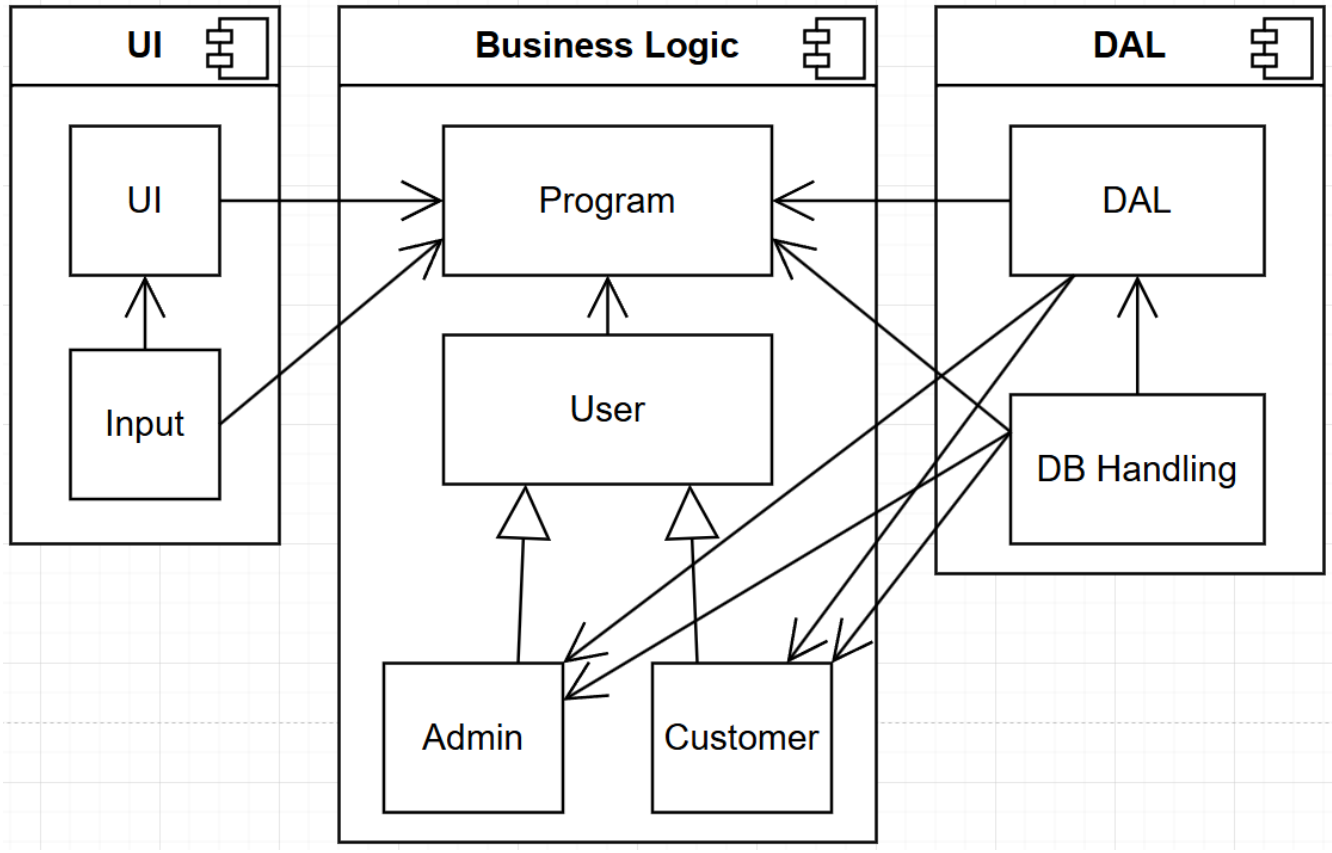
resources since it runs on top the hardware's existing not needing its own. Docker also includes convenient features for version control and reusability. Simple commands allow for saving containers to images and tagging them however you'd like. This is perfect rolling back to a previous setup is ever desired by you or your team.

While Docker is extremely convenient it is not perfect. There is a slight learning curve to Docker albeit not as large as VMs. While there are some options that exist, managing a lot of containers at once is not nearly as easy as VMs running with the right software. Another major downside is no direct GUI support while using Docker containers. There are workarounds to display GUI applications on your host running from Docker but otherwise it is all command line usage. Another pain is managing persistent storage as the main way to store things from a container is virtually mounting directories from your host machine. Finally, Docker shares your host OS kernel meaning you can't necessarily run apps requiring a different OS than the host.

Finally we have using a plain old computer. I think this is best in situations where you are working on your own or on a small team. Ideally on a short term project or some testing/practice utilizing some new software. The biggest benefit is there is absolutely zero learning curve using your own machine as you are likely already extremely familiar with it. Likely already has your preferred OS that will work for your project. There is also no additional setup required to get started. No performance overhead from running emulation software on your existing hardware. No additional cost required and no increased security risk incurred. As simple and straightforward as it gets.

The downsides of using your computer normally is really just missing out on all of the pros from using VMs or Docker. No extra portability or OS options by default. No fancy features for simple deployment or scalability as there really isn't any to deploy or scale. No extra reusability of your environment for rolling back or restoring other than saving your work. Limited in what you can run based on whatever OS/Software you currently have. No easy way to share your exact environment/setup with someone else to work with it. Lacking of all fancy features.

System design:



Class Diagram: