CS 3240 - Fall 2021 / Software Architecture Quiz

Computing ID	Name
cas8ds	Chloe Smith

On my honor as a student, I have neither given nor received unauthorized assistance on this assessment.

Signature: _________Chloe Smith_______________________________

For all questions, **we expect you to answer in full sentences** unless specifically directed otherwise.

The best answers to questions will:
- Use good grammatical structure with complete sentences
- Be succinct and direct
- Show understanding of the material
- Not contain superfluous information
- Be professionally stated

Instructions: Please make a copy of this document into your own personal Google Drive folder. Then type your answers underneath the associated question. Export your final submission as a PDF and upload to the appropriate assignment in Gradescope.

When you upload, you MUST choose the correct page(s) for each question! If you do not do this, we may miss your questions and not grade them!

This quiz is open notes/slides/videos. You may reference any material from the course that you like. Indicate on *each question at the bottom of the page* what material you referenced, if any. There is no time limit, just a submission deadline, but please don't go overboard and spend hours on this when you have other things to do. The quiz is intended to take around one hour. If you do not do well on the quiz, remember that you will have a chance to retake these questions as a part of the final exam to earn any XP you did not get on this quiz.

You are on your honor not to discuss this quiz with any other students or share answers in any way! ***Copying any text from any source word for word (including course slides or any website) will earn 0 XP and a professionalism penalty.***

You may post questions privately on Piazza, but note that there are VERY few questions that we will answer. We need to make sure that everyone has the same information to be fair to all. Further, no late work is accepted since quizzes can be retaken as a part of the final exam.

Good luck!

1. The most important reason that we learn about software architecture and design is so that when we are working on a team we can have/emphasize ______________________ *(one phrase)*

organization of the project in the midst of flexible requirements.

(2000 XP)

2. Discuss what we mean when we say "separation of concerns" when considering a software system's architecture. Consider the *PepsiPrime* architecture diagram in the Development Scenario. Why might "separation of concerns" be important here and what advantages could it give? *(2000 XP)*

When we say "separation of concerns" we are referring to the breakdown of specific tasks of our project to different areas. For example, we can separate the controller from the view. The "controller" is concerned with the actual interactions between the user and the program (eg, sending data back and forth), whereas the view is concerned with what the user sees displayed on the screen (eg, loading an html file). This sort of separation makes it easier to make sure that all these different concerns are addressed, and that everything is organized and understood.

For the PepsiPrime example, there are multiple levels of things happening. Over to the left, we have different sorts of displays for the client to view. To the bottom right we have various APIs stored and in the top right we have a database server. We also notably have authentication servers in the middle top. This is a lot of different pieces with different functionalities. It would be easy to miss pieces or to connect them in an inefficient way. Separation of concerns is important here because it could organize this project in a fairly intuitive and cohesive way. We could have a controller running along the diagram arrows and reacting to button clicks. We could have views to display the common api to the various applications. We could have models holding various other characteristics of our users and the tools they need. In a similar way to the benefit of having multiple people on a team with varying specialties, the separation of concerns helps the whole team to operate more clearly and to know "where to go" to address a certain issue.

3. Why has MVC become a key architectural design pattern for web applications? Would an MVC framework make sense for the *PepsiPrime* system? Why or why not? Provide specific examples. *(2000 XP)*

MVC has become a key architectural design pattern for web applications because it separates concerns in a way that makes sense and is fairly simple and effective. The "model" of the MVS framework refers to the actual data stored. It is more backend, database type stuff. The "view" displays things to the user. It is more html, UI type stuff. Lastly, the controller aids the interaction between the user and the program. It can take a request, find the right view and model, and return necessary outcomes to the user.

I think an MVC framework would work really well for the PepsiPrime system. The model can store information on the users and the tools (such as current number of hours worked by a user). The view can display needed information on the tools and the users (such as the number of hours worked being written to the screen in a blue tool logo). The controller can take a request and run through Model and View (such as the user adding 8 more hours, the controller updating the count in model, and updating the view to display the new total). This would be a very cohesive and good way to manage the system in my opinion.

4. Does Django follow the three-tier architectural design pattern? Why or why not? Be specific with your examples. *(2000 XP)*

Yes, Django uses the MVC, which is a form of three-tier architectural design pattern. There are three main separations of concerns in this structure: Models, View, and Controller. Django utilizes all three. I know this first hand because I myself have been writing model and view files for my project, and I have seen this process of updating via the controller. I have utilized this structure within the Django framework. To be more specific, my group's project has a Housing model in models.py that contains attributes of off-grounds housing. It has a name, address, housing type, etc. The project also has a FilterView in views.py that queries the houses and displays the ones that match a given filter search.

5. Describe the purpose of REST as an architectural design pattern. What problem is it trying to solve? Do you believe that the *PepsiPrime* system should emphasize a RESTful architecture design? Why or why not? *(2000 XP)*

REST stands for "Representation State Transfer," and it is a Stateless architectural design pattern that avoids storing State information in active memory on the web server. It tries to solve the State problem that comes with the internet being inherently asynchronous. I believe that we should use REST for the PepsiPrime system, because it is a pretty standard, tried-and-true way to run these sorts of internet applications. I could definitely see us using GET and POST for our sessions as our users login, to store things that they input at the time. Plus, it is completely normal to have both MVC and REST for the same program, and I think in this PepsiPrime case both make sense and would work well together.

Referenced
https://docs.google.com/presentation/d/1bUBh0M0SqhITou4ijoi-7EJjk_nwwqPIIlSHTydwK5E/edit#slide=id.g348c754b90_0_349

APPENDIX: Development Scenario *(Same scenario from Quiz 1 and 2, but nothing new added.)*

AwesomeSoft is a mobile and web app development company located in Reston, VA. The company was founded in 2010 by three recent graduates from UVA and JMU who found a passion for making iOS apps in their spare time. The idea for the company was to focus solely on building iOS apps for other companies that needed a mobile presence.

Fast forward to 2021 and the company has grown steadily. In fact, even before COVID, the company had a forward-thinking work-from-home policy, where almost two-thirds of the developers worked from home most days and only came into the office on rare occasions. During COVID, the company easily transitioned to a fully-remote, virtual working environment and may never go back. Currently, the company employs 40 total developers, split across several different projects. Over half of the developers have been in the industry for at least five years, with a burst of hiring of new graduates over the summer of 2019 and 2020.

AwesomeSoft has landed the biggest contract they have ever gone for - with PepsiCo! Pepsi drink machines are found all over the country - from can and bottle dispensers to fountain machines. PepsiCo employs hundreds of technicians that work out of regional offices to go around and repair machines, order new parts, order more product for the machines, etc. Currently, these technicians use specialized barcode scanners along with huge binders full of information about parts to find what they need and then orders are placed on a website when they get back to their office, introducing a delay in getting what they need.

AwesomeSoft has pitched the idea to PepsiCo to replace their outdated system with a sleek, user-friendly iOS iPad app for technicians! The app will do everything from letting the technician record their hours worked, to submitting order requests, to communicating with customers. All in a much more easy to use app!

The managers at AwesomeSoft are diverting half of the development personnel to work specifically on this project, while the other developers finish up with other existing projects. In order to help the developers learn exactly what the needs of the technician are, several developers are going to ride along with the technicians for a week to see how they do their jobs! And, in fact, you, as the product owner and Scrum Master, are heading out next week! After you see how things work for the technicians, you will draw up a set of requirements for the system.

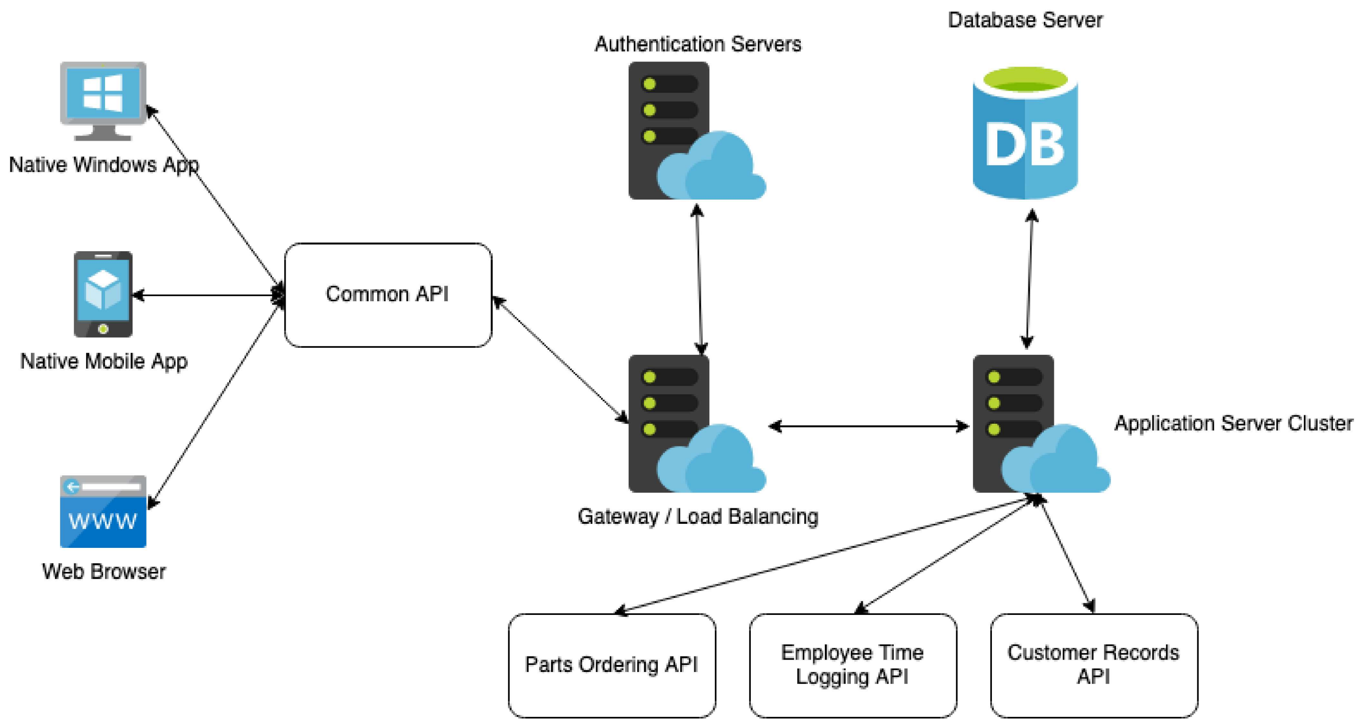
The contract from PepsiCo says that the new app has to be ready in 9 months or else you forfeit a lot of money. The contract doesn't say if they will be providing any further technical or testing support beyond the ride-alongs with technicians. You may be able to request more support, but you can't count on it and may have to find other reliable ways to test the system. PepsiCo has provided detailed specs of the various API endpoints that the current systems use, such as the format of the data that needs to be sent to the parts ordering system. You do have a contact at PepsiCo you can reach out to who can answer "big picture" requirements questions, but they may not have detailed technical answers. The docs will have to do. Good luck!

Update from Quiz 2 (imagine this is two weeks later in the context of the scenario…)
Your team has finally finished all the ride-alongs with the technicians from PepsiCo. At the latest update meeting, you walked away with some key notes from everyone:

- The technicians definitely do not want to have to put in a password over and over in order to use the app, but they need it to be locked down so no one else can access their account.
- They know most of the parts by sight, but not all of them - particularly some of the smaller parts in the older machines. They want some way to identify the parts effectively so they can place the correct orders.
- They don't want to be bothered with putting in overly specific hours if possible… can the app somehow do that for them based on what they are doing at that time?
- It would be nice if they didn't have to swap between the new app and their GPS/mapping app when they are travelling to different sites.

You also met with managers at PepsiCo. They want things as simple as possible for their technicians, while also generating all the info that they need, such as the hours technicians worked, how long they were at each job, if they were slacking off excessively, and what sorts of parts were being ordered. They would like this information to be easily accessible so they can run reports on individual technicians and job sites. Charts, graphs… you get the idea.

Update for Quiz 5
Your team has begun creating system architecture and high-level system designs for your solution, which you are calling *PepsiPrime*. Below in the current diagram that your team is working with to help communicate your design decisions to the rest of the team.


Native Windows App
Native Mobile App
WWW
Web Browser
Common API
Authentication Servers
Database Server
DB
Gateway / Load Balancing
Application Server Cluster
Parts Ordering API
Employee Time Logging API
Customer Records API