CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
DIT345	Fundamentals of Software Architecture			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
DIT345 -0001-0XR		4-01-2024	16	

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	✓	14.5	
2	✓	24	
3	✓	37	
4	✓.	18	
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen		93.5	

CHALMERS | Anonymous code: DIT345-0001-0XR | Points for question (to be filled in by teacher): | Consecutive page no. Löpande sid nr: 1
Anonym kod | Poäng på uppgiften (ifylles av lärare): | Question no. Uppgift nr: P1 A

~~QP1~~ QP1

A.

Functional Requirement:

The system must consist of several storage machines to ~~enable access to clients~~ provide storage space to clients. 1.5

Technical Constraint:

The system shall consist of such storage components that are ~~non~~ functional only when required (e.g. in case of failure of other components).

quality req. 0

CHALMERS | Anonymous code DIT345-0001-0XR | Points for question (to be filled in by teacher) | Consecutive page no. Löpande sid nr 2
Anonym kod | Poäng på uppgiften (ifylles av lärare) | Question no. Uppgift nr P1 B

P1. B

Availability and performance are the two most imporetant quality attributes for the GFS.

It can be seen in the figure above that the tactic of ~~active~~ passive redundancy or spare servers and databases have been applied on the GFS architecture. There are two servers with the same name 'GFS chunkserver' which might be there ~~so that~~ to ensure system availability even during peak loads when system is accessed by several clients as stated in the technical constraint that some components might not be functional at times it can be that they are utilised as spare components or to add redundancy. Eg if one server is down the other can be utilised. We can also call this load balancing as if both of them are functional at the same time the load can be distributed more equally ensuring better availability for users. ✓ 4p

Large systems like GFS have to really make sure that the system's performance is good as it directly affects the views of users about their system and so the customer satisfaction. Here in this figure by looking at multiple databases it is clear that GFS might have the tactic 'increase available

2

P1.B continued...

~~resourced~~ 4p

resources' applied. As GFS system is using hundreds ~~&~~ and thousands of storage machines it is therefore increasing their storage resources which ensures efficient utilisation of those resources and thus better performance.

Moreover, only few interactions ~~are~~ between the server and client are taking place through 'GFS master' and where ~~it~~ is not necessary they are communicating directly. This is ~~ensured~~ ensuring reduced computational overhead. Thus increasing the performance.

This is how I can see that availability and performance are both relevant and applied on GFS architecture.

2

P1. C

These are the Quality attribute Scenarios for both QAs. H+M

QAs →	Availability	Performance
Source	client	client
Stimulus	attempts to access system website	attempts to access a particular file by clicking
Artifact	System	File system GFS master!
Environment	During peak load	Under peak load
Response	Opens, or loads the webpage	loads & Opens the relevant file
Response Measure	Is at least available 99.8% of times	Within 0.8 seconds of the click

5P

2

P2-A

M, M H, M

QAs →	Modifiability	Security
Source	Frontend Developer	An intruder / unauthorised user
Stimulus	makes changes to the GUI of the application	tries to access the GFS system
Artifact	GUI client application	Security system
Environment	During maintainence phase	During normal operation
Response	Successfully modified GUI	Denying service and informs Security team to take action & recover system
Response measure	1 Component's modules or code is changed	Within 1 min of DOS attack

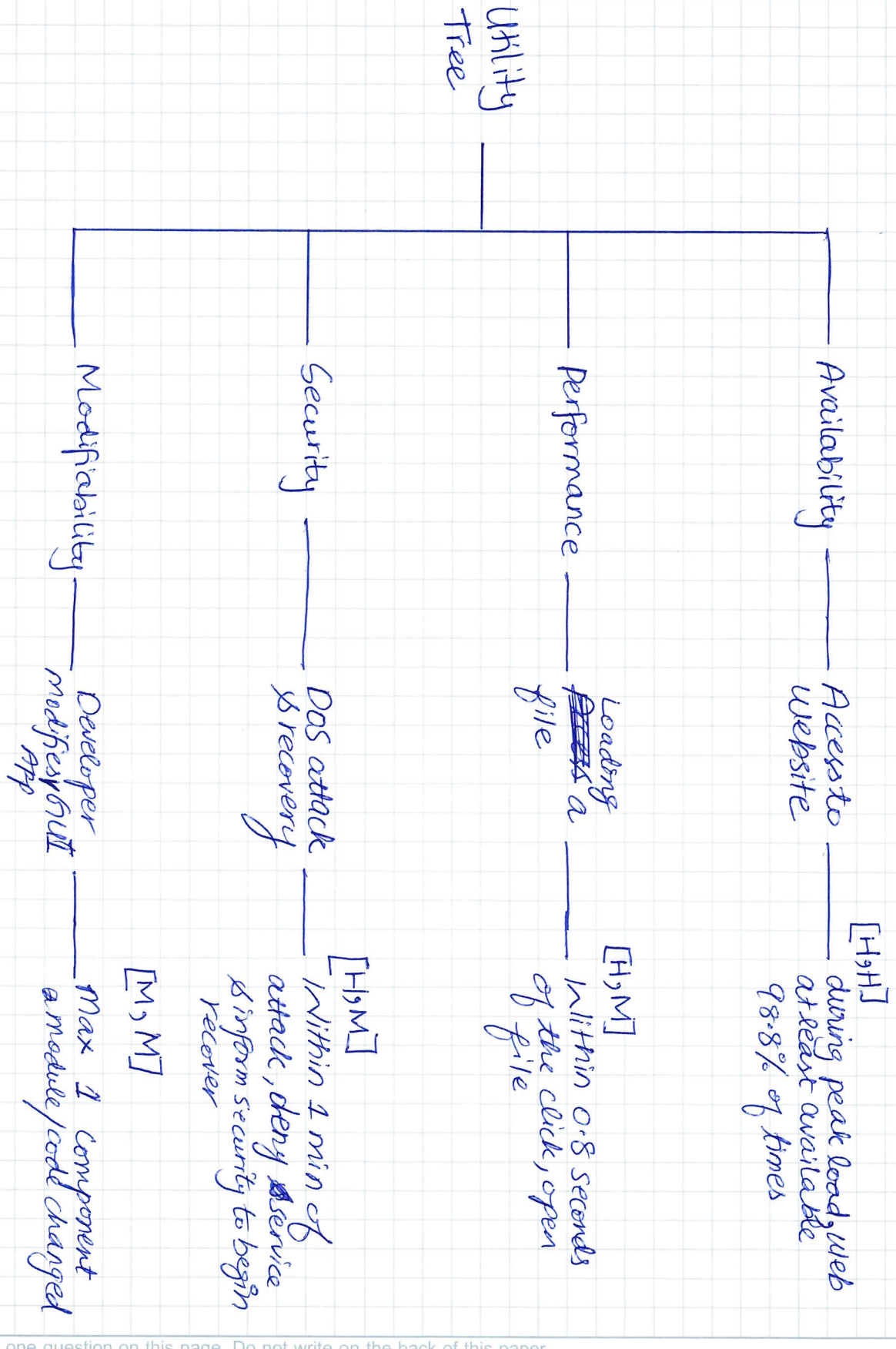
12P

2

P2.B

Utility Tree

Please look at it by turning it horizontal :)



12

2

QP3

A. The junior software architect seems to be using a service-oriented style. It can be seen through breakdown of the system into several component but all of them using the same database. It can also be likely that the junior software architect also tried to include peer to peer style in the architecture partially as some components like UI and BPS bds tracker are directly communicating, they both might be utilising and providing services to each other

I do not think that the architecture is satisfactory. Firstly, due to high complexity and communication difficulties over the network ~~it would be~~ microservice styles affect performance of the system adversely. In fact in this architecture diagram the architect have not even maximised the decoupling of components as it is supposed to be in microservices style. But there is instead a lot of dependencies and interactions with several components shown by the arrows. This ~~can leads to~~ can lead to poor performance. Secondly, service-oriented style can be good for ~~easy~~ good modifiability of the system. But according to this diagram components are not decoupled enough to enable least effect of modifications on the other components therefore this architecture can also cause

7P

2

OP3-A continued ....

poor modifiability.

Moreover, the peer-to-peer style kind of approach would not be suitable at all as the sequence of orders, commands and events is not clear in this style and it can cause huge problems for a time-intensive system like app Västtrafik. These issues would adversely affect the ~~perfo~~ availability and performance too.

2

Q P3-B

The junior software architect's solution followed some design principles such as information hiding and cohesion.

As we can see that the information from bus tracker GPS is only required by the app's UI to show it to the user therefore it is only interacting with the UI. The data access is only needed by the app and timetable organizer and account management so it is only connected to them. This shows that the relevant information is only given access to the components that necessarily require it.

The principle of what belongs together, is similar and relevant is tried to be followed also by keeping the timetable organiser and account management to have everything belonging to organising and management at one place.

Moreover, the ticket purchase information for example is only shared with a few relevant components in order to ensure security for the user accounts and maintain integrity of the system by following the information hiding principle

All the parts related to a particular travel, both search and ticket purchasing are kept together in the architecture. It might be an attempt to maintain coherance principle.

Lastly, even though the junior software architect tried to follow some design principles there is still room for improvement.

7P

2

-0.5p how does the user interact with the system? No path?

QP3-C

UIVästrafikTogo
«Subsystem»
client

User

«Component» TimeTable

schedules

«Component» TravelSearch

«Component» AccountInfo

«Component» PurchaseAndTicket

only admins can monitor Process

Accountdetail

web situation

«Component» ProcessMonitor

time table

routes

Purchase process, status, ticket Price

web status measure

Administrator

All of them are receiving data from server

16p

locationdata

«GPS Component» Bus Tracker

locationdata

Server
«sub system Server»

«component» TimeTableOrganiser

«Component» AccountManagement

«ServiceProvider» PaymentCompanies

«component» Travelsearch

«Component» TicketPurchasing

Payment service & verification

All user infos history

timetable updates, schedules

«database» AccountDatabase

«database» TimeTable& TravelDatabase

«database» PurchaseAnd Ticket Database

tickets & carddetail price & confirmation

2

Q P3 - C continued...

I ~~created~~ chose the client server, layered architecture style for the architecture of this system. It consists of the client side with a UI for the users. The UI fetches data from the system server where most of the processing takes place. The server is connected to some databases in order to update and fetch relevant data.

The system is also farther divided into more components. Reducing coupling and ensuring better coherance. External systems and actors interacting with the system are also shown.

I used the tactic process monitor ~~three~~ to tackle the issues of availability. This will ensure the admin can monitor the current status of app availabily and take measures according to that. Moreover by decoupling components like Travelsearch and ticket purchasing I have removed unnecessary 'interactions thus improving computational efficiency. I also ~~did~~ increased available resources by increasing storage resources (databases) shown by multiple data bases. Both of these ~~steps~~ will enhance performance of the system.

-0.5 p process monitor is a tactic where a monitor process observes the state of a

2

Q P3-D

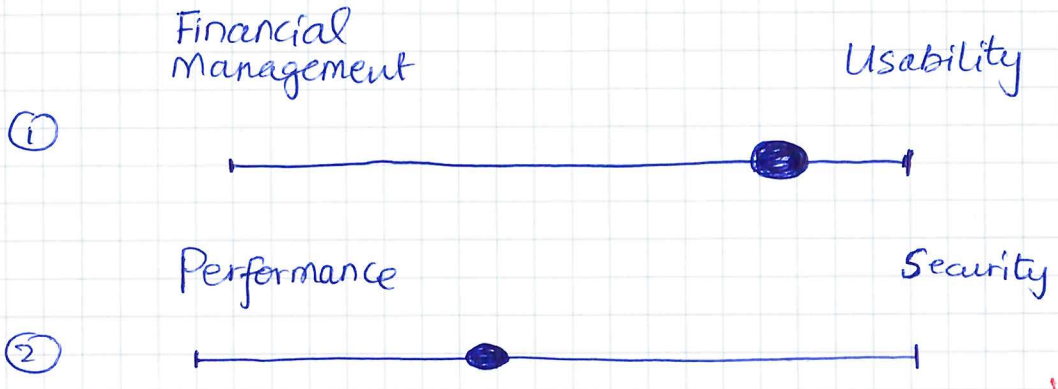
This solution is better than my coworker's solutions as it is an improved, simpler and cleaner architecture. My solution has followed separation of concerns design principle fully by breaking down the system into separate components. Where each component performs a single specific task. For example ticket purchasing is separate from travel search. This will ensure both components have only a specific purpose reducing confusion and complexity. This approach would dramatically increase the performance and improve modifiability of the system. As we can see in my co-worker's solution several interactions between all components was increasing the computation overhead, reducing performance. At the same time the increased dependencies would have made it impossible to make modifications by not affecting most of the system. Therefore, this solution also improves modifiability. Layered architectures are common so are easier to understand and implement. We don't have too many layers to increase upfront cost. The info-hiding design principle is also improved as eg the purchase info is only shared with service provider or the ticket database. also ensuring security The availability will be ensured through process monitoring by the system admin. The structure of this solution with multiple databases for load balancing, separation of concerns and higher security ensures a secure, highly available, modifiable and good performing system

2

QP4 – Normal ~~A~~

- Yes, I participated in the workshop.
- Yes, I remember the trade-off points and decisions not fully but quite well.

A.



What is the architectural aspect of this tradeoff?

For ① it was important to make a tradeoff according to my ~~team~~ because for enhancing ~~usab~~ user usability for the app ~~we~~ the ~~org~~ organisation would require to spend more resources like financial, time and others on the application. It ~~was~~ became a debate to determine if so many resources were worth being spent on enhancing the look and usability of the app. We decided we were slightly in favour of enhancing usability. We chose this as we assumed the goverment and organisation has enough resources through the taxes and purchases that it is still affordable to choose usability. This is because usability enhancement is an investment itself as it will lead to more user satisfaction ~~so~~ thus more users and profit to the company.

Or?

2

Q P4-Normal continued

A ....

For (2) it was important to make this tradeoff as ~~increased security~~ prioritising increased security could lead to more computational overhead and so ~~a~~ leading to poor performance. Performance on the other hand was also important as all users would require quick responses during their travels.

We slightly favoured performance as it is very significant in travel apps where users might have very few minutes to eg catch the bus and need information or processing immediately. Security could be compromised on a little as the serious security procedures like payments are dealt by external companies like 'visa' themselves so we don't need too much implementation for that

4 P

2

QP4; B°. 8P

They chose microservice style for it to ensure quick deployability, testability and maintainability along with layered style.
After reflection and listening to other groups' presentations I would not use it like that. Instead, I would choose layered and service oriented style only. This is because microservices ~~can~~ can lead to poor performance and high delay due to network calls which take longer than method calls. Moreover, the system is more complex to build. We do not even have too many different services to provide for using this style. Theuse of several databases is also not suitable as we will constantly need to update relevant data and database will need to maybe interact. This effect performance very badly so I would not choose this style. Layer & service-oriented style will be easier to implement & less complex. We will also make sure good modifiability by decoupling of components. Infact it would also reduce computational overhead, improving performance. As it's simpler we can focus on handling failures and improve availability. Single database will also ensure data is consistent improving accuracy

2

Q P4 Normal C, [Assuming 5 sentences ≠ 5 lines]

1. I realised how different people approach the problems and their solutions very differently.
2. How discussing and coming to a conclusion unanimously in a team while convincing everyone is not as easy as it sounds.
3. No architectural style is good or bad; it is only a good or bad choice relevant to a system.
4. Every architectural style can be improved by tactics. We have a huge range of tactics
5. ; we just need to find the ones suitable for our system.

6