

Assinment39

pebj, smot

October 2, 2014

1 Part I - OOSE

1.1 Architectual style

Why have we chosen the client/server architecture?

We have chosen the client/server architecture style because we need to have constantly running services to notify a user when an event comes up etc. This architecture fits this scenario because it is possible to have that running on a server constantly, but not on a client which can't be expected to have the program open constantly.

Another reason is that we can have a database and have code that a client won't be able to see or manipulate, thus allowing us to prevent a user to harm the system and its other users.

Additional explanations is given by the benefits below that we have deemed more interesting than the drawbacks.

Benefits: A server will be able to handle security manners in a way, so that the client won't be able to harm the system and its other clients. For example, accesing a database won't need the client to hold a database connectionstring that could be sniffed/hacked out of the program. Also, because of this architure, code execution like database manipulation can be more centralized rather than being executed from each client or different services etc. and in that sense, a server can also function as a respiratory architure.

Another benefit is that by having a server handling the database etc. it can be minimized what the client needs to send and receive, thereby possibly increasing performance for a client with bad network connection.

Drawbacks: Each service in the system runs on its own and is therefore susceptible to either being attacked individually or breaking down. If the services is dependent on some state of another service and that one break down, it could possibly corrupt the system.

Another drawback is with performance which will strongly rely on the client-server connection speed. If a massive amount of data was to be transmitted regurally with a bad connection, it could harm the performance potentially.

1.2 Design goals

- *Usability* To integrate CALENDAR in the users daily activities, the interface and functionality have to be kept simple and intuitive. This design goal is a refinement of the nonfunctional requirement “Usability”.
- *High availability* For better user interaction it should be possible for the clients to access our system from as many places and as often as possible.
- *High Reliability* No potential issues may occur during daily usage that would make it impossible to do a use case. This design goal is a refinement of the nonfunctional requirement “Reliability”.
- *Stable response time* The system’s “response time” may not increase with increasing numbers of users or events, Therefore the CALENDAR should support a growing user base. Performance should be as close to constant as possible. This design goal is a refinement of the nonfunctional requirement “Performance”.
- *High extensibility* It should be possible to extend key sectors of the system. Thus a feature in addition to notifications should be addable without having to alter the code necessary for the notification task etc. to work. This design goal is a refinement of the nonfunctional requirement “Supportability”.
- *High portability* To maximize the availability of CALENDAR it has to be available on as many of the users platforms as possible. This design goal is a refinement of the nonfunctional requirement “Implementation”.

1.3 Decomposition

ralla dalla katamaran diagram med prut Figure 1 asfsafafs

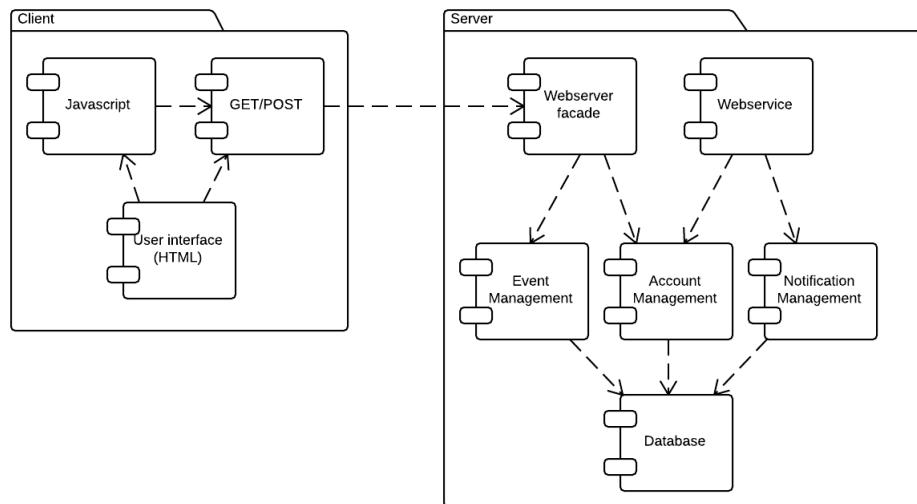


Figure 1: CALENDAR subsystem decomposition (UML component diagram)