Fundamentals of Software Design Unit 1

David Finger

Colorado Technical University

Abstract

Remove differences between User Interfaces of various software vendors. Enhance the security of the data and improve system availability and performance for a global company.

Funcational Requirements:

1. View Databases
   1. Provide a window that allows for selection of databases available to view.
2. Save Data within Databases
   1. Allow writing of data to databases, creating new fields as needed.
3. Create new forms within databases
   1. Allows creation of new sub-forms within database structures.
4. Delete old databases
   1. Allows for the removal of databases that are no longer relevant.
5. Create New Databases
   1. Allows for the creation of new databases as needed to be stored on the server.

NonFunctional Requirements:

1. syncs all data between databases
   1. ensures that data stays up to date within all databases saved on the server.
2. encrypts databases containing sensative information
   1. Encypts data within certain forms to ensure security of data is preserved.
3. allows databases to be viewed by allowed users remotely
   1. authorized user list allows databases to be viewed remotely by authorized users only.

User Interface Specifications:

Diagram

Description automatically generated

User case descriptions:

1. client login
   1. clients are allowed to login remotely to view documents within authorized databases stored on the cloud server.
2. Employee login
   1. Employees are allowed to login and view as well as write to open contracts on the cloud server. These changes will be seen real time when viewed.
3. Employee forms
   1. Employees will be able to add additional forms to contracts within database system.
4. Admin User lists
   1. Administrators will be able to maintain user lists as well as remove contracts from the database when necessary.
5. Admin database creation
   1. Admins will be able to create new databases as needed for new contracts.
6. Admin Login
   1. Administrators will be able to login to view, create, and delete databases and their forms.

Components Overview:

Login services: Login services are provided per the company to admins, employees and clients. Secure login is required, through the use of organization VPN services. Admins, clients, and employee login use cases apply.

Authorized user lists: Authorized User Lists are to be maintained at all times for each contract stored on the database. Admins may update these lists remotely using secured connections. Admin user list use case apply.

Database forms: Database forms are able to be Viewed by Clients, Employee’s, and admin users while signed in securely, either remote or on site. Clients may access contract forms with read only access. Employees and admins may view contract forms to read and write with admins being able to add and delete contracts as well.

Database storage: Database storage is taken care of utilizing cloud services. This enables clients, employees, and admins to access contract forms at all times. Updates to contract forms, via employees or admins, are viewable almost immediately using remote connections. Admin database use case applies.

Design Rationale:

Service based paradigm: The Service Design Paradigm creates logical designs that creates quality interactions between the service provider and the end users. This paradigm allows for collaboration between designers and stakeholders to ensure that all the needs of the users are meet before the final product is presented. However, this can also lead to longer development times and in the end a much more simplistic featured design.

Functional Design Paradigm: The Functional Design Paradigm creates very simple designs that allow for each part to be clearly shown where it leads to and how it will behave. This paradigm allows for things to go very quickly as everything is placed out in simple ways to ensure usability amongst a vast array of users. However, this leads to the designs not being very complex and therefore requires more navigation to reach intended areas if they fall within certain bounds.

Overall, I believe that the Functional Design Paradigm works better for this design because we want everything to be simple yet protected. Allowing for a simple design keeps the needed information available while hiding everything that can be considered sensitive behind more complex searches.

Resources

Kaisler, S. H. (2005). Software paradigms. Wiley-Interscience.

*Example: Software Architecture Document*. (n.d.). Retrieved April 4, 2021, from <https://www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch.htm>

Functional vs Non-Functional Requirements—Understand the Difference. (2012, April 4). *ReQtest*. <https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/>

Functional design. (2020). In *Wikipedia*. <https://en.wikipedia.org/w/index.php?title=Functional_design&oldid=939128138>

Service design. (2021). In *Wikipedia*. <https://en.wikipedia.org/w/index.php?title=Service_design&oldid=1015168762>