Internet applications / web application programming

Version: 2.0

Job text (description)

We have a set of files that a group of people are working on and we want to ensure that no two people are writing in parallel to the same file. What is required:

1. A system on the web that enables users to add their files to it and place files in either a free or a used state and is reserved for a user.
2. Files must be organized into groups so that the user cannot access all files, but only files belonging to groups he is entitled to access. 3. The basic use cases are **in-check** as it allows a user free to browse files, reserve a file, replace the file in his favor, upload and modify it, while **out-check** allows the second method for the user to return the old modified file and return it to its state free.
4. The user can select more than one file and perform the in-check process, and the system must ensure that all of them are free before booking them. Either all or the reservation fails for all files.
5. The system must ensure that two users cannot reserve the same file at the same time. 6. We must be able to export reports of booking and editing operations according to the file or user. 7. The system must ensure that 100 users can work in parallel.

Required: Developing a system of prerequisites that fulfills the following conditions:

structural requirements

The mechanism for verifying the implementation of the requirement	requirement
Interface layer: Android or Web System layer: Contains system (must be APIs) Storage layer: Contains engine Database	Separating the application into 3 layers so that this helps to distribute the work better (3 tier)
Separation of components so that any component can be dispensed without problems or modification.	The system must separate functional and non-functional requirements so that there is no overlap or programming link between them
The code can be transferred to a non-framework without the need to change	A separation between the system code and the framework code
Correct use of design patterns and no design errors.	The system should use patterns design that support the distribution of work among group members
The way to access the system is through APIs	Use APIs to connect the interface layer to the system layer

The models in the system are properly mirrored on Database engine	Using the ORM tool and not directly entering the database
Realization of transaction within functional requirements	Achieve a transaction at the system level
Achieving authentication within Middleware (Aspect)	Authentication
Secure authorization verification within a separate Middleware	Authorization
functional requirements	
Adding a new user to the system by writing his basic information such as: email, full name, username and password (you can add whatever you want). There must be no users with the same email or username	User register
Provide login feature by username and password	User Login
The user can upload a new file to the system (in case it is free) or delete a file (if the file is free). Ensure that there is no competition for the file in the event of creation or must be deleted	Create or delete a file
The user can create a group (with a special name) where: • A file can be added to the group (only if the file is the owner) • A file can be deleted from the group (only if the file is the owner) • Add other users to the group • Delete users from the group (if the user is not registered for any file in the group) • Deleting a group in the event that there are no files held by users in it The system contains an exceptional private group called public as well so that any user can add his files to it so that any user in the system can access the file (in-Check and out Check)	Create or delete groups to share files
• Files uploaded by the user • Display the status of the file (free or reserved) • Display the name of the user who reserved the file if it is Reserved View groups owned by View files in each group (in the same view). previous(	View created files and groups

Read a file: the user can read the file without changing the state Reserve the file in-Check: the user checks the file so that no other user can read or write to the reserved file (only if the file is free) Modify: the user (who has checked) can file) from the modification of the reserved file, the completion of the reservation or out-Check: canceling the reservation on the file to return to the free state in-check- bulk: so that the user can select more than one file and reserve all of them at the same time or the process fails. Caching is used to shorten the display time of files in a collection. What is meant here is to provide access time for the Database in order to find out which files are in the collection. Note: The system must use the system's Systems File for file storage and management	Operations on files
The user can export the following reports: • A report that contains a history of complete events for a file so that the report contains (the file upload date, the file reservation date for the file that was reserved by the system, the modification date, and the reservation cancellation date) arranged from the most recent event to the oldest, and it contains the user name at each event	reports
• The system contains user admin so that the files in the system have additional authority to view all files and groups	Admin user
• The group does not contain a group	Notes
Examples: • Search on a file • Search within files	Other features (optional)
non-functional requirements	
Outgoing Response Record all incoming and outgoing Requests associated with the previous Request. Ensure that Logging will use the same port He should Services for accessing the Database and storing information.	(compulsory) Login

The system must provide the ability to change some values such as: • Multiple engine database support so that the engine can be changed from the system settings. • Options for changing connection settings with the Database (database, port, or credential address) • Support for Loglevel control options • System-specific options (ex: more files per user)	(compulsory) Configuration
The system must be able to serve at least 100 users simultaneously. This must be proven by using Jmeter Apache or application test load (exporting a report) or any proof that the system succeeded in bypassing the previous condition.	(compulsory) Performance
The system must support escalation: • Run Load balancing between 2 or more instances of the system • Have a health check api to check the state of each instance in the system If cache is used at cluster level: You must ensure that cache inconsistency does not occur between instances	Scalability
Use a reverse proxy to receive and forward requests Register users' IP Support for 2http protocol Support for https applications without check health for the api Support Support for compression response if larger than 1MB Provide a limiting rate for the number of requests allowed from a source (through aspect or without reverse (proxy) Specify the maximum size of the request to a certain limit (such as 10MB) In the case of using aspect to develop the limiting rate, you should pay attention to the change in the request source	Security and optimization features

Required:

• Fulfillment of all structural, functional, and non-functional (compulsory) requirements: • Number of students 3 or less: only the compulsory requirements • Number of students: 4: fulfillment of one additional non-functional requirement • Number of students: 5: fulfillment of two additional non-functional requirements