===================

02 - Mini Project

==================

Sample Screens : http://ashokitquotes.s3-website.ap-south-1.amazonaws.com/index.html

1) User Registration

- Dependent Dropdowns (country, state & city)

- Dropdowns options should come from database

- Based on country selection, states should be populated

- Based on state selection, cities should be populated

- When Signup is clicked, Generate Random Pwd and send to user in email

2) User Login

- Validate credentials

- If user trying to login for first time, display Reset Pwd page

- After pwd reset, display Dashboard page with random quote

- We need to fetch random quote from third party rest api (webservice call)

(REST API URL : https://type.fit/api/quotes)

- If user trying to login for second time, display dashboard page directley

3) Reset Pwd Page

- This page page will display only when user login for first time

- This is used to update user password

4) Dashboard page

- After successful login, dashboard page will be displayed

- Dashboard page will display random quote given by third party rest api

- Get New Quote button will display new random quote

5) Logout

- When user logout, then we need to redirect to login page

================

Database tables

================

USER\_MASTER

- USER\_ID INT PK AUTO\_INCREMENT

- UNAME VARCHAR

- EMAIL VARCHAR

- PWD VARCHAR

- COUNTRY\_ID INT

- STATE\_ID INT

- CITY\_ID INT

- PWD\_UPDATED VARCHAR

- CREATED\_DATE TIMESTAMP

- UPDATED\_DATE TIMESTAMP

COUNTRY\_MASTER

- COUNTRY\_ID INT PK AUTO\_INCREMENT

- COUNTRY\_NAME VARCHAR

STATES\_MASTER

- STATE\_ID INT PK AUTO\_INCREMENT

- STATE\_NAME VARCHAR

- COUNTRY\_ID INT

CITIES\_MASTER

- CITY\_ID INT PK AUTO\_INCREMENT

- CITY\_NAME VARCHAR

- STATE\_ID INT

=======================

Java & UI Components

=======================

Entity & Repository

- User.java & UserRepo.java

- Country.java & CountryRepo.java

- State.java & StateRepo.java

- City.java & CityRepo.java

Bindings

- LoginForm.java

- RegisterForm.java

- ResetPwdForm.java

- QuoteApiResponse.java

Service

- UserService.java

- public Map<Integer, String> getCountries( );

- public Map<Integer, String> getStates(Integer countryId);

- public Map<Integer, String> getCities(Integer stateId);

- public User getUser(String email);

- public boolean saveUser(RegisterForm formObj);

- public User login(LoginForm formObj);

- public boolean resetPwd(ResetPwdForm formObj);

- private String generatePwd();

- DashboardService.java

- public String getQuote( );

Controllers

- UserController.java

- public String index(Model model);

- public String login(LoginForm form, Model model);

- public String register(Model model);

- public String resetPwd(Model model);

- public Map<Integer, String> loadStates(Integer cid);

- public Map<Integer, String> loadCities(Integer sid);

- public String userRegister(RegisterForm form, Model model);

- public String updatePwd(ResetPwdForm form, Model model);

- public String logout (Model model);

- DashboardController.java

- public String buildDashboard(Model model);

- public String getNewQuote( );

Utils

- EmailUtils.java

- public boolean sendEmail(String sub, String body, String to);

- DataLoader.java (Runner)

Constants

- AppConstants.java

Props

- AppProps.java

Exception

- AppExceptionHandler.java

Views

- login.html

- register.html

- resetPwd.html

- dashboard.html

- errInfo.html

- app.j

CID

=====================

Form Binding Classes

====================…

[1:45 AM, 3/31/2024] Adarsh ❣️: =======================

Logging in Spring Boot

========================

Local Env : Developers wil write the code and will test

DEV Env : Developers will perfrom integration testing

SIT Env : Testing team will test app functionality

UAT Env : Client side team will test app functionality

PROD Env : Live Deployment

===================

What is Logging ?

===================

=> The process of storing application execution details to console/ file is called as Logging

=> To understand runtime behaviour of our code

=> To identify exceptions in the code

=> To identify root cause of the exception

=====================

Logging Architecture

======================

1) Logger

2) Layout

3) Appender

=> Logger is a class which is used to generate log msgs

logger.trace()

logger.debug()

logger.info()

logger.warn()

logger.error()

=> Layout represents log msg format

=> Appender represents destination to store log msgs

Ex : ConsoleAppender (will print msgs on console)

Ex : FileAppender (will store msgs in file)

==================

Logging Levels

==================

=> Log Level will decide which msgs should be generated/printed.

TRACE > DEBUG > INFO > WARN > ERROR

=> When we set LOG level, from that level all higher levels will be printed.

Note: In springboot, the default log level is INFO.

=> We can change log level in boot application like below

logging.level.root = debug

logging.file.name=ashokit.log

-------------------------------------------------------

@RestController

public class MsgRestController {

private static Logger logger = LoggerFactory.getLogger(MsgRestController.class);

@GetMapping("/welcome")

public String getMsg() {

logger.info("getMsg() - execution stated..");

String msg = "Welcome to Ashok IT..!!";

try {

int i = 10 / 0;

} catch (Exception e) {

logger.error("Exception : " + e.getMessage());

}

logger.info("getMsg() - execution ended..");

return msg;

}

}

---------------------------------------------------------------

=> In realtime , multiple users will access our application on daily basis.

=> If we maintain single log file then lot of data will be stored in that file

=> After few days/months we can't open that file because of huge size.

=> To overcome this problem we will use Rolling concept in logging.

=======================

Rolling File Appender

=======================

2 ways

1) Time Based Rolling

Ex: Create New Log file for every 24 hours (day wise)

2) Size Based Rolling

Ex: Store only 1 GB data in one log file

=> We can configure rolling in 2 ways

1) Properties / yml file

2) xml configuration (recommended)

ex : logback.xml

===============

Logging Tools

===============

1) Log4J (vulnerability)

2) Log4j2

3) Logback

4) Logstash

5) JCL

Note: We will use SLF4J as a facede for logging.

========================

configuring logback.xml

========================

-> logback.xml file is used to configure logging information

-> We will keep logback.xml file in src/main/resources folder

-> Below is the sample logback.xml file

<configuration>

<appender name="Console" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d [%thread] %-5level %-50logger{40} - %msg%n</pattern>

</encoder>

</appender>

<appender name="RollingFile"

class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>MyApp.log</file>

<encoder>

<pattern>%d [%thread] %-5level %-50logger{40} - %msg%n</pattern>

</encoder>

<rollingPolicy

class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">

<fileNamePattern>MyApp-%d{yyyy-MM-dd}.%i.log</fileNamePattern>

<maxFileSize>1MB</maxFileSize>

<maxHistory>30</maxHistory>

<totalSizeCap>10MB</totalSizeCap>

</rollingPolicy>

</appender>

<root level="INFO">

<appender-ref ref="Console" />

<appender-ref ref="RollingFile" />

</root>

</configuration>

======================

Log Monitoring Tools

======================

=> These are used for monitoring logs of our application

1) ELK / EFK - Open source s/w

2) Splunk - Commercial s/