
System Requirements Specification

Index

For

E-Mail Pro Connect

Version 1.0

TABLE OF CONTENTS

BACKEND-EXPRESS NODE APPLICATION	3
1 Project Abstract	3
2 Assumptions, Dependencies, Risks / Constraints	4
2.1 Admin Constraints:	
2.2 Blog Constraints	
2.3 Order Constraints	
2.4 Product Constraints	
2.5 User Constraints	4
3 Rest Endpoints	5
3.1 AdminRoutes	
3.2 BlogRoutes	
3.3 OrderRoutes	
3.4 ProductRoutes	
3.5 UserRoutes	5
4 Template Code Structure (modules)	6
4.1 controller	6
4.2 dao	
4.3 routes	
4.4 service	
4.5 serviceImpl	9
5 Considerations	9
FRONTEND-ANGULAR SPA	10
1 Problem Statement	10
2 Proposed My Time Away Wireframe	10
2.1 Home Page	10
2.2 User Page	11
2.3 Admin Page	11
3 Business-Requirement:	12
7 Execution Steps to Follow for Backend	14
8 Execution Steps to Follow for Frontend	15

E-MALL PRO CONNECT

System Requirements Specification

You need to consume APIs exposed by Backend application and implement authentication (if required) in Angular to make application work as FULLSTACK

BACKEND-SPRING BOOT RESTFUL APPLICATION

1 PROJECT ABSTRACT

"E-mall Pro Connect" is a full-stack e-commerce application designed to provide a seamless online shopping experience. It leverages the MEAN stack, with MongoDB as the database, Express.js for the backend, Angular for the frontend, and Node.js as the runtime environment. This platform aims to connect buyers and sellers, allowing users to browse, search for, and purchase a wide range of products.

The system offers a variety of features and functionalities for both customers and sellers, enabling them to manage their transactions efficiently.

Following is the requirement specifications:

	E-Mall ProConnect	
Modules		
1	Admin	
2	Blog	
3	Order	
4	Product	
5	User	

Admin Module Functionalities	
1	Get list of all users
2	Get list of all products
3	Get list of all orders
4	Get list of all blogs
5	Get dashboard containing Users, Products, Orders and Blogs count
6	Get User, Product and Order reports for users count, products count with category and orders count grouped by total order price.
7	Get sales report by sharing total orders group by their total revenue
8	Get list of all products
9	Get insights of all order grouped by each user

Blog Module Functionalities	
1	Can create a new blog
2	Get blog by it's id
3	Update the blog by it's id
4	Delete blog by it's id
5	Get all blogs
6	Get list of all popular blogs
7	Add a comment on particular blog
8	Edit a comment on particular blog
9	Delete a comment on particular blog
10	Get list of all categories of blogs
11	Can like a blog
12	Get the comments count
13	Get the blog details by product

Order Module Functionalities	
1	Get all orders
2	Get order by id
3	Create an order
4	Update an order it's id
5	Delete an order by it's id
6	Get order for user
7	Cancel any order
8	Get the payment details of any order
9	Process the payment for any order
10	Get the analytics of orders with data for totalOrders, average order amount, highest and lowest order amount
11	Generate the invoice for any order with order id, order date and total amount details

12	Track the order shipment by returning status and tracking number
13	Get the revenue analytics which returns the total, highest and lowest revenue
14	Check whether the product is ordered or not

Product Module Functionalities	
1	Create a new product
2	Get product by id
3	Update product by id
4	Delete product by id
5	Get all products
6	Get list of all top rated products
7	Search product by name or description
8	Apply some discount in cart
9	Can checkout the cart
10	Add product to cart
11	Get all cart items
12	Update the product in cart
13	Delete the product in cart

User Module Functionalities	
1	Create a new user
2	Get user by id
3	Update user by id
4	Delete user by id
5	Get user by email
6	Get user activity
7	Get user favorites
8	Change the user password
9	Log the user activity
10	Login the user

2 ASSUMPTIONS, DEPENDENCIES, RISKS / CONSTRAINTS

2.1 ADMIN CONSTRAINTS

- 2.2 When accessing any admin route, it is necessary to check the authentication via token and also the user must be of type admin..

2.3 BLOG CONSTRAINTS

- 2.4 When creating a blog, if a blog post is not found with a given id, it must return a Blog post not found.
- 2.5 When updating a blog, if a blog post is not found with a given id, it must return a Blog post not found.
- 2.6 When deleting a blog, if a blog post is not found with a given id, it must return a Blog post not found.
- 2.7 When add a comment in a blog, if a blog post is not found with a given id, it must return a Blog post not found.
- 2.8 When editing a comment in a blog, if a blog post is not found with a given id, it must return a Blog post not found.
- 2.9 When deleting a comment in a blog, if a blog post is not found with a given id, it must return a Blog post not found.
- 2.10 When liking a blog, if a blog post is not found with a given id, it must return a Blog post not found.
- 2.11 When getting comments count in a blog, if a blog post is not found with a given id, it must return a Blog post not found.

2.12ORDER CONSTRAINTS

- 2.13 When getting an order, if an order is not found with a given id, it must return an Order not found.
- 2.14 When editing an order, if an order is not found with a given id, it must return an Order not found.
- 2.15 When deleting an order, if an order is not found with a given id, it must return an Order not found.
- 2.16 When canceling an order, if an order is not found with a given id, it must return an Order not found.
- 2.17 When retrieving payment details for an order, if an order is not found with a given id, it must return an Order not found.
- 2.18 When processing the payment for an order, if an order is not found with a given id, it must return an Order not found.
- 2.19 When generating an invoice for an order, if an order is not found with a given id, it must return an Order not found.

2.20 When tracking a shipment for an order, if an order is not found with a given id, it must return an Order not found.

2.21 PRODUCT CONSTRAINTS

- 2.22 When getting a product, if a product is not found with a given id, it must return a Product not found.
- 2.23 When updating a product, if a product is not found with a given id, it must return a Product not found.
- 2.24 When deleting a product, if a product is not found with a given id, it must return a Product not found.
- 2.25 When checking out a cart, if a cart is not found with a given id, it must return a Cart not found.
- 2.26 When applying a discount on a cart, if a cart is not found with a given id, it must return a Cart not found.

2.27 USER CONSTRAINTS

- When getting a user, if a user is not found with a given id, it must return a User not found.
- When updating a user, if a user is not found with a given id, it must return a User not found.
- When deleting a user, if a user is not found with a given id, it must return a User not found.
- When getting a user by email, if a user is not found with a given id, it must return a User not found.
- When getting a user activity, if a user is not found with a given id, it must return a User not found.
- When changing user password, if a user is not found with a given id, it must return a User not found.
- When logging an user's activity, if a user is not found with a given id, it must return a User not found.
- When logging, if a user is not found with a given email, it must return a User not found.

Common Constraints

- All the database operations must be implemented in serviceImpl file only.
- Do not change, add, remove any existing methods in the service file.
- In the service layer, custom methods can be added as per requirements.
- All RestEndpoint methods and Exception Handlers must return data in json format.
- Any type of authentication and authorisation must be added in routes file only.

3 REST ENDPOINTS

Rest End-points to be exposed in the routes file and attached with controller method along with method details for the same to be created. Please note, that these all are required to be implemented.

3.1 ADMIN CONTROLLER

Note: All routes must be authenticated and only accessible to admin type user

URL Exposed		Purpose
1. /api/admin/users		Fetches all the users
Http Method	GET	
Parameter	-	
Return	list of users	
2. /api/admin/products		Fetches all the products
Http Method	GET	
Parameter	-	
Return	list of products	
3. /api/admin/orders		Fetches all the orders
Http Method	GET	
Parameter	-	
Return	list of orders	
4. /api/admin/blogs		Fetches all the blogs
Http Method	GET	
Parameter	-	
Return	list of blogs	
5. /api/admin/dashboard		Fetches the dashboard details
Http Method	GET	
Parameter	-	
Return	fetch dashboard	
6. /api/admin/reports		Fetches all reports
Http Method	GET	
Parameter	-	
Return	all reports	
7. /api/admin/reports/sales		Fetches the sales reports
Http Method	GET	
Parameter	-	
Return	fetch sales reports	
8. /api/admin/products/inventory		Fetches all the names of products
Http Method	GET	
Parameter	-	
Return	list of names of products	

9. /api/admin/orders/analytics		Fetches the orders analytics
Http Method	GET	
Parameter	-	
Return	fetch analytics of all orders	

3.2 BLOG CONTROLLER

Note: Routes which must be authenticated are mentioned with “isSecured” below.

URL Exposed		Purpose
1. /api/blogs/create		Creates a new blog [isSecured]
Http Method	POST	
Parameter	-	
Return	Blog	
2. /api/blogs/:id		Fetches a blog by id
Http Method	GET	
Parameter	id	
Return	Blog	

3. /api/blogs/:id		Updates a blog by id [isSecured]
Http Method	PUT	
Parameter	id	
Return	Blog	
4. /api/blogs/:id		Deletes a blog by id [isSecured]
Http Method	DELETE	
Parameter	id	
Return	Blog	

5. /api/blogs/product/:productId		Fetches a blog by product id
Http Method	GET	
Parameter	productId	
Return	Blog	
6. /api/blogs/all		Fetches all blogs
Http Method	GET	
Parameter	-	
Return	List of blogs	

7. /api/blogs/popular		Fetches the popular blogs
Http Method	GET	

Parameter	-	
Return	List of blogs	
8. /api/blogs/:id/comments		Adds a comment on a blog [isSecured]
Http Method	POST	
Parameter	id	
Return	Blog	
9. /api/blogs/:id/comments/:commentId		Updates a particular comment on a blog [isSecured]
Http Method	PUT	
Parameter	id commentId	
Return	Blog	
10. /api/blogs/:id/comments/:commentId		Deletes a particular comment on a blog [isSecured]
Http Method	DELETE	
Parameter	id commentId	
Return	Blog	
11. /api/blogs/categories		Fetches the list of all distinct categories
Http Method	GET	
Parameter	-	
Return	List of categories	
12. /api/blogs/:id/like		Adds a like on a comment on a blog [isSecured]
Http Method	PUT	
Parameter	id commentId	
Return	Blog	
13. /api/blogs/:id/comments/count		Counts the number of comments on a blog
Http Method	GET	
Parameter	id	
Return	Count	

Order CONTROLLER

Note: All routes must be authenticated.

URL Exposed		Purpose
1. /api/orders/all		Fetches all the orders
Http Method	GET	
Parameter	-	

Return	list of orders	
2. /api/orders/create		Creates a new order
Http Method	POST	
Parameter	-	
Return	Order	

3. /api/orders/:id		Fetches an order by id
Http Method	GET	
Parameter	id	
Return	Order	
4. /api/orders/:id		Updates an order by id
Http Method	PUT	
Parameter	id	
Return	Order	

5. /api/orders/:id		Deletes an order by id
Http Method	DELETE	
Parameter	id	
Return	Order	
6. /api/orders/user/:userId		Fetches all order by userId
Http Method	GET	
Parameter	userId	
Return	List of order	

7. /api/orders/cancel/:id		Cancels an order by id
Http Method	DELETE	
Parameter	id	
Return	Order	
8. /api/orders/:id/payment		Fetches the payment details for an order
Http Method	GET	
Parameter	id	
Return	payment details	

9. /api/orders/:id/pay		Process the payment for an order
Http Method	POST	
Parameter	id	
Return	Acknowledgement message	

10. /api/orders/analytics		
Http Method	GET	

Parameter	-	Fetches the analysis of order and return an object with totalOrders, average order amount, highest order amount and lowest order amount
Return	Analytics object	

11. /api/orders/:id/invoice		Creates an invoice object with details like orderid, order date and total amount.
Http Method	GET	
Parameter	id	
Return	Invoice object	

12. /api/orders/:id/shipment		Checks the shipping status of any order
Http Method	GET	
Parameter	id	
Return	Shipment object	

13. /api/orders/revenue		Analysis the revenue on all orders by returning an object with data for total revenue, highest revenue, lowest revenue
Http Method	GET	
Parameter	-	
Return	Revenue object	

14. /api/orders/ordered/:userId/:productId		Checks whether user has ordered some product or not
Http Method	GET	
Parameter	userId productId	
Return	boolean	

PRODUCT CONTROLLER

Note: Routes which must be authenticated are mentioned with “isSecured” below and wherever admin user is required will be mentioned as “isAdmin”.

URL Exposed		Purpose
1. /api/products/all		Fetches all products
Http Method	GET	
Parameter	-	
Return	List of products	
2. /api/products/create		Creates a new product [isSecured] [isAdmin]
Http Method	POST	
Parameter	-	
Return	Product	

3. /api/products/search		Searches a product
Http Method	GET	
Parameter	-	
Return	List of products	
4. /api/products/top-rated/:limit		Fetches a list of all top-rated products [isSecured]
Http Method	GET	
Parameter	limit	
Return	List of products	
5. /api/products/discount/:userId		Applies the discount on all products for a user [isSecured]
Http Method	POST	
Parameter	userId	
Return	Acknowledged message	
6. /api/products/cart/:userId		Fetches the cart for a user
Http Method	GET	
Parameter	userId	
Return	Cart	
7. /api/products/cart/add/:userId		Adds a product to cart [isSecured]
Http Method	POST	
Parameter	userId	
Return	Cart	
8. /api/products/cart/checkout/:userId		Checks out a cart for user [isSecured]
Http Method	POST	
Parameter	userId	
Return	Acknowledged message	
9. /api/products/cart/update/:userId/:itemId		Updates an item for a user in cart [isSecured]
Http Method	PUT	
Parameter	userId itemId	
Return	Cart	
10. /api/products/cart/remove/:userId/itemId		Deletes a particular item in cart for a user [isSecured]
Http Method	DELETE	
Parameter	userId itemId	
Return	Cart	

11. /api/products/:id		Fetches the product by id
Http Method	GET	
Parameter	id	
Return	Product	

12. /api/products/:id		Update a product by id
Http Method	PUT	
Parameter	id	
Return	Product	

13. /api/products/:id		Deletes a product by id [isSecured][isAdmin]
Http Method	DELETE	
Parameter	id	
Return	Product	

PRODUCT CONTROLLER

Note: Routes which must be authenticated are mentioned with “isSecured” below and wherever admin user is required will be mentioned as “isAdmin”.

URL Exposed		Purpose
1. /api/users/login		Logins user
Http Method	POST	
Parameter	-	
Return	User	
2. /api/users/create		Creates a new user [isSecured] [isAdmin]
Http Method	POST	
Parameter	-	
Return	User	
3. /api/users/:id		Fetches a user by id [isSecured]
Http Method	GET	
Parameter	id	
Return	User	
4. /api/users/:id		Updates a user [isSecured]
Http Method	PUT	
Parameter	id	
Return	User	
5. /api/users/:id		Deletes a user
Http Method	DELETE	
Parameter	id	
Return	User	

6. /api/users/:id/activity		Checks the activity of user [isSecured]
Http Method	GET	
Parameter	id	
Return	Activity of user	

7. /api/users/:id/favorites		Checks the favorites of user [isSecured]
Http Method	GET	
Parameter	id	
Return	Favorites	

8. /api/users/change-password/:id		Changes the password of user [isSecured]
Http Method	PUT	
Parameter	id	
Return	User	

9. /api/users/email/:email		Fetches the user by email [isSecured]
Http Method	GET	
Parameter	email	
Return	User	

4 TEMPLATE CODE STRUCTURE

4.1 Admin code structure

1) MODULES/ADMIN: controller

Resources

AdminController (Class)	This is the controller class for the admin module.	To be implemented
-----------------------------------	--	-------------------

2) MODULES/ADMIN: dao

Resources

File	Description	Status
models/admin model	Models for admin	Already implemented

schemas/admin schema	Schemas for admin	Already implemented
-----------------------------	-------------------	---------------------

3) MODULES/ADMIN: routes

Resources

File	Description	Status
Admin routes	Routes for admin	Partially implemented.

4) MODULES/ADMIN: service

Resources

Class	Description	Status
AdminService	<ul style="list-style-type: none"> Defines AdminService 	Already implemented.

5) MODULES/ADMIN: service/impl

Resources

Class	Description	Status
AdminServiceImpl	<ul style="list-style-type: none"> Implements AdminService. 	To be implemented.

4.1 Blog code structure

1) MODULES/BLOG: controller

Resources

BlogController (Class)	This is the controller class for the blog module.	To be implemented
----------------------------------	---	-------------------

2) MODULES/BLOG: dao

Resources

File	Description	Status
models/blog model	Models for blog	Already implemented
schemas/blog schema	Schemas for blog	Already implemented

3) MODULES/BLOG: routes

Resources

File	Description	Status
Blog routes	Routes for blog	Partially implemented.

4) MODULES/BLOG: service

Resources

Class	Description	Status
BlogService	<ul style="list-style-type: none"> Defines BlogService 	Already implemented.

5) MODULES/BLOG: service/impl

Resources

Class	Description	Status
BlogServiceImpl	<ul style="list-style-type: none"> Implements BlogService. 	To be implemented.

4.1 Order code structure

1) MODULES/ORDER: controller

Resources

OrderController (Class)	This is the controller class for the order module.	To be implemented
-----------------------------------	--	-------------------

2) MODULES/ORDER: dao

Resources

File	Description	Status
models/order model	Models for order	Already implemented
schemas/order schema	Schemas for order	Already implemented

3) MODULES/ORDER: routes

Resources

File	Description	Status
Order routes	Routes for order	Partially implemented.

4) MODULES/ORDER: service

Resources

Class	Description	Status
OrderService	<ul style="list-style-type: none">Defines OrderService	Already implemented.

5) MODULES/ORDER: service/impl

Resources

Class	Description	Status
OrderServiceImpl	<ul style="list-style-type: none"> Implements OrderService. 	To be implemented.

4.1 Product code structure

1) MODULES/PRODUCT: controller

Resources

ProductController (Class)	This is the controller class for the product module.	To be implemented
------------------------------	--	-------------------

2) MODULES/PRODUCT: dao

Resources

File	Description	Status
models/cart model	Models for cart Models for product	Already implemented
models/product model		
schemas/product schema	Schemas for product	Already implemented

3) MODULES/PRODUCT: routes

Resources

File	Description	Status
Product routes	Routes for product	Partially implemented.

4) MODULES/PRODUCT: service

Resources

Class	Description	Status
-------	-------------	--------

ProductService	<ul style="list-style-type: none"> Defines ProductService 	Already implemented.
-----------------------	--	----------------------

5) MODULES/PRODUCT: service/impl

Resources

Class	Description	Status
ProductServiceImpl	<ul style="list-style-type: none"> Implements ProductService. 	To be implemented.

User code structure

1) MODULES/USER: controller

Resources

UserController (Class)	This is the controller class for the user module.	To be implemented
----------------------------------	---	-------------------

2) MODULES/USER: dao

Resources

File	Description	Status
models/user model	Models for user	Already implemented
schemas/user schema	Schemas for user	Already implemented

3) MODULES/USER: routes

Resources

File	Description	Status
------	-------------	--------

User routes	Routes for user	Partially implemented.
--------------------	-----------------	------------------------

4) MODULES/USER: service

Resources

Class	Description	Status
UserService	<ul style="list-style-type: none"> Defines UserService 	Already implemented.

5) MODULES/USER: service/impl

Resources

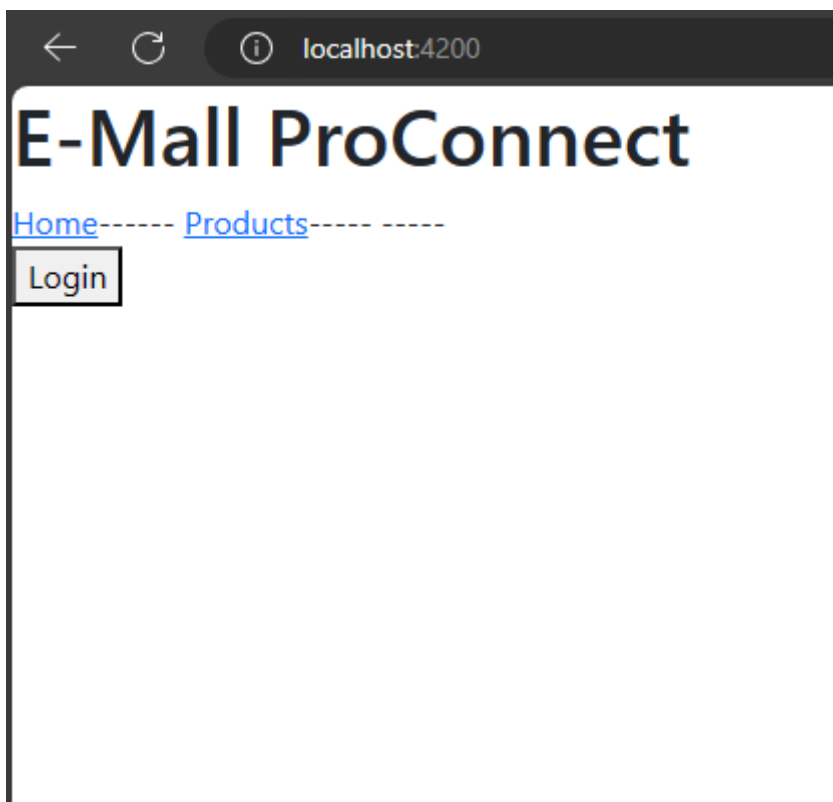
Class	Description	Status
UserServiceImpl	<ul style="list-style-type: none"> Implements UserService. 	To be implemented.

1 PROBLEM STATEMENT

E-Mall ProConnect is SPA (Single Page Application), it enables users to manage various aspects and streamline the process of managing products.

2 PROPOSED EMALL PROCONEECT

UI needs improvisation and modification as per given use case and to make test cases passed.



E-Mail ProConnect

[Home](#)-----[Products](#)-----

Login

Login

Email:

Email is required

Password:

Password is required

Login

localhost:4200/products/list

E-Mail ProConnect

[Home](#)

[Products](#)

Login

[Products](#)

[View Cart](#)

Product List

Search Product by Name:

Search

Product2

Price: \$10.99

Description: \$Product Description

Add to Cart

View Details

Product1

Price: \$10.99

Description: \$Product Description

Add to Cart

View Details

Updated Product1

Price: \$10.99

Description: \$Product Description

Add to Cart

View Details

←

↺

i

localhost:4200/products/list

E-Mail ProConnect

[Home](#)

[Products](#)

Login

[Products](#)

[View Cart](#)

Product List

Search Product by Name:

product1

Search

Product1

Price: \$10.99

Description: \$Product Description

Add to Cart

View Details

Updated Product1

Price: \$10.99

Description: \$Product Description

Add to Cart

View Details

3 BUSINESS-REQUIREMENT:

As an application developer, develop the E-Mall ProConnect (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Home Page	As a user I should be able to visit the Home page as the default page.

US_01	Login Page	As a user I should be able to visit the Login page once I click on the Login button.
US_01	Products Page	<p>As a user I should be able to see the homepage and perform all operations:</p> <p>Acceptance criteria:</p> <ol style="list-style-type: none"> 1. As a user I should be able to furnish the following details at the time of creating a leave application. <ol style="list-style-type: none"> 1.1 Product name 1.2 Price 1.3 Description 1.4 Add to cart button 1.5 View details button.
US_02	Product Details Page	<p>As a user I should be able to see the product details as:</p> <p>Acceptance criteria:</p> <ol style="list-style-type: none"> 2. Product name 3. Description 4. Price 5. Category 6. Image 7. Add to cart button 8. Blog details like title, content, author name, category and comments 9. It should also show likes

EXECUTION STEPS TO FOLLOW FOR BACKEND

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.
3. This editor Auto Saves the code.
4. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use **CTRL+Shift+B**-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
5. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
6. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.
7. You can follow series of command to setup express environment once you are in your project-name folder:
 - a. `npm install` -> Will install all dependencies -> takes 10 to 15 min
 - b. `npm run start` -> To compile and run the project.
 - c. `npm run jest` -> to run all test cases and see the summary of all passed and failed test cases.
 - d. `npm run test` -> to run all test cases and register the result of all test cases. **It is mandatory to run this command before submission of workspace** -> takes 5 to 6 min
8. You need to use **CTRL+Shift+B** - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

EXECUTION STEPS TO FOLLOW FOR FRONTEND

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal ->New Terminal.
3. This is a web-based application, to run the application on a browser, use the internal browser in the environment.
4. You can follow series of command to setup Angular environment once you are in your project-name folder:
 - a. npm install -> Will install all dependencies -> takes 10 to 15 min
 - b. npm run start -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:4200 to open project in browser -> takes 2 to 3 min
 - c. npm run test -> to run all test cases. **It is mandatory to run this command before submission of workspace -> takes 5 to 6 min**
5. You need to use **CTRL+Shift+B** - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.