Bonanza

A Museum Inventory Management System

Group 3

Author Note

First Part: Final Report

Appendix A: Architectural Design

Appendix B: Database Design and Sample Data

Appendix C: Web API Design

Appendix D: Screenshots of Front-End interface

Developed by Huicong Jiang and Guanchen Zhao in 2021

Project Documentation and Final Report

Abstract

Collections are the most important tangible and intangible assets for a museum. On the one hand, the collections themselves are of high values. On the other hand, it's the collections that attract visitors instead of the museum building. Therefore, a reliable and user-friendly collections management system is critical to every museum. To address this issue, we proposed a museum inventory management system, Bonanza, using the RFID technology and service-oriented architecture.

In this system, each collection is tagged with an RFID device to identify itself, and every possible location, such as display shelves and storage rooms, is equipped with an RF reader to find the tagged collections within its range. As long as an RF reader finds a tagged collection, its location will be updated in the database through network and web service, by which we realize real-time monitoring of the collections' location. From visitors' perspective, they can subscribe to some specific collections. For example, whenever a subscribed collection changes its location from a stock room to a display shelve, the user will receive a notification that this collection is in display now.

Keywords: Inventory, Museum, service, RFID

Introduction

Bonanza realized real time monitoring of collections location in a museum. As a manager you don't need to update the item's location manually. Instead, the RF reader updates hundreds of thousands items automatically and reliably. Meanwhile, Bonanza also gives public access to the

museum inventory and provides notification services to subscribers, which allow people to make informed decisions before they visit the museum.

For example, a museum staff is moving a famous painting out of stock room 1 for display. As soon as the painting is out of the stock room 1, the RF reader will notice that and update the item's status in the inventory as "relocating" through a web service. When the painting is safely placed on the display shelve, the RF reader in the shelve will find the painting is inside its range and update the painting status into "display, floor 1, shelve 10A". Meanwhile, if a user subscribed to this painting, he will receive an email or message notification that his beloved collection is now on display.

Feature

Front-end and back-end separation architecture

Frond-end: React

Back-end: Spring-Boot

In the decoupled web architecture, frontend and backend code are separate, with no shared resources. The two codebases communicate, but each has its own server instance. The backend application serves data via an API (application programming interface) using a framework such as JSON (JavaScript Object Notation). A decoupled approach is advantageous for facilitating changes, enabling individual services and components to be independently scaled up or down or removed entirely without disrupting the entire application. Additionally, decoupling allows frontend and backend developers each to optimize their portions without fear of how their work

impacts the rest of the system. Developers in general prefer the decoupled approach as it tends to remove production bottlenecks, simplify testing and make for a more easily reusable backend codebase.

Authentication

After the user logs in, the authentication token will be returned to the user, which is required to be carried in the header of subsequent requests. The token store in Redis, the key is token, value is user information.

Application of Cache Service

We have access to the redis to provide us with the caching service.

Https

Our requests are all made through https, in which we use the domain name and the certificate service provided by aws.

Soap and Restful

We combine restful service with soap service. The framework is spring boot and spring ws.

Data is stored after encryption

We use md5 + salt to encrypt sensitive information, such as passwords.

Single page application

There is only one html file in the entire application, and the transformation of web content depends entirely on the change of the front-end DOM tree. Therefore, the web page is free from refreshing, and the experience is smoother.

Reliable router

For example, if there is no token, no other pages can be accessed (except for registration), and will only be redirected to the login page.

Front-end information linkage

Any change, such as updating the user name, adding a new item, the front-end can respond immediately, no need to refresh, no need to reload.

Full application of component-based development

The last is the full application of component-based development, which improves the efficiency of project card issuance and code reuse rate.

Deploy

We tried multiple platforms for deployment.

Back-end

AWS Elastic Beanstalk and AWS EC2.

Front-end

AWS Amplify and Google App Engine.

Database

RDS on Google Cloud Platform and RDS on AWS.

Redis

AWS EC2.

Future work

Deploy on App Engine

After joining mybatis, we encountered compilation problems when deploying to app engine. It seems to be related to the jdk version, and we will study this issue in the future.

AWS Pinpoint and SNS

AWS Pinpoint and SNS are service notification services provided by aws, we can introduce them for message notification.

Microservice structure

We can start multiple services to make a microservice structure.

Related Work

Cross-Origin Resource Sharing (CORS)

As a project that separates the front and back ends, we encountered cors problems when communicating, which took some effort.

Cross-Origin Resource Sharing (CORS) is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading resources. CORS also relies on a mechanism by which browsers make a "preflight" request to the server hosting the cross-origin resource, in order to check that the server will permit the actual request. In that preflight, the browser sends headers that indicate the HTTP method and headers that will be used in the actual request.

AWS

You may find that we use a lot of AWS services. We have done a lot of work in this area of research.

References

Architecture

https://www.vshift.com/ideas/conscious-decoupling-the-future-of-web-development

Spring Boot

https://spring.io/guides/gs/spring-boot/

Spring WS

https://www.springboottutorial.com/creating-soap-web-service-with-spring-boot-web-services-starter

AWS EC2

https://docs.aws.amazon.com/ec2/index.html?nc2=h_ql_doc_ec2

Appendix A: Architectural Design

Major modules

We have 4 modules:

- Login: contains login, register and logout.
- User: contains add, update, delete and page list with search.
- Inventory: contains add, update, delete and page list with search.
- Subscribe: contains add, update, cancel and page list with search.
- Appointment: contains add, update, cancel and page list with search.

Front-end and back-end separation architecture

In the decoupled web architecture, frontend and backend code are separate, with no shared resources. The two codebases communicate, but each has its own server instance. The backend application serves data via an API (application programming interface) using a framework such as JSON (JavaScript Object Notation). A decoupled approach is advantageous for facilitating changes, enabling individual services and components to be independently scaled up or down or removed entirely without disrupting the entire application. Additionally, decoupling allows frontend and backend developers each to optimize their portions without fear of how their work impacts the rest of the system. Developers in general prefer the decoupled approach as it tends to remove production bottlenecks, simplify testing and make for a more easily reusable backend codebase.

Design Issues

1. About the role

We have 2 roles: **admin** and **normal**.

The front end will control whether the corresponding menu is displayed according to the role.

2. why redis?

Redis works with an in-memory dataset.it is very fast to read data. Obtaining current user information is a very frequent operation.

3. The design of the subscribe button

The icon is a heart shape, it means like, it is in line with the action of paying attention.

4. About the index

The index is the inventory list page, which is out main service, and also the user's target.

Service Architecture

Load balance: The backend service is deployed on AWS Elastic Beanstalk, so we config load balance with AWS Elastic Beanstalk. We have two pods to process the request.

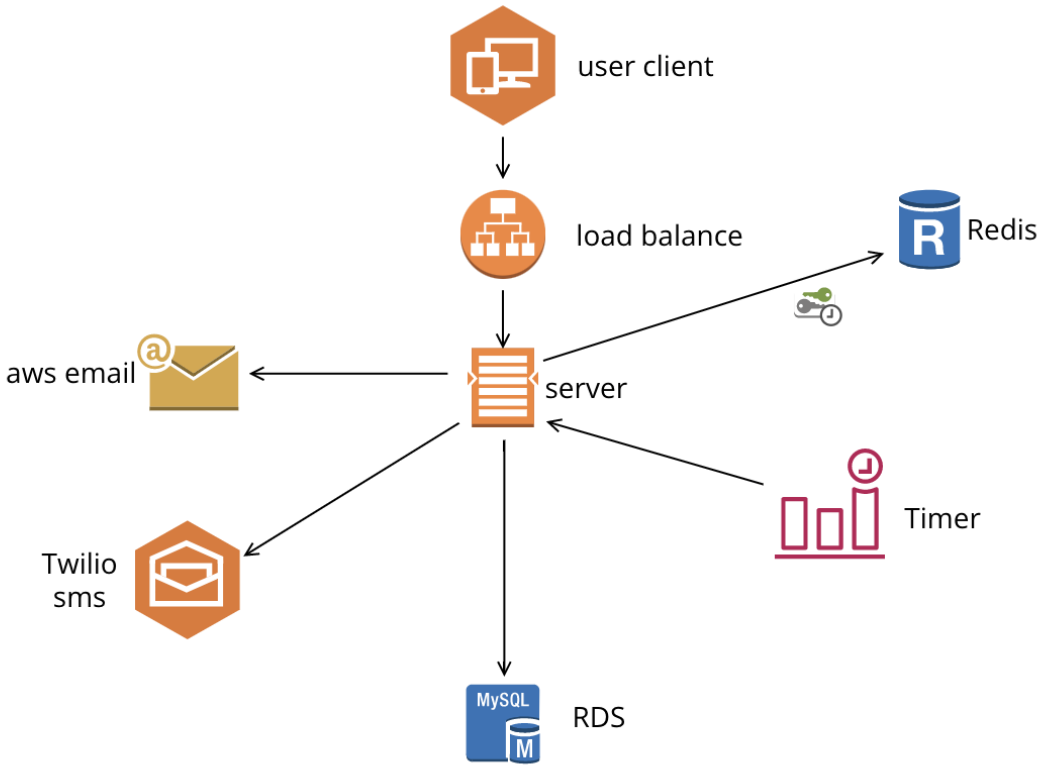
Redis: Redis server is installed on AWS EC2, we use Redis to save token and user information.

RDS: we use Google RDS - MySQL as our data storage.

AWS Email: we use AWS Email Service to send email to users.

Twilio: we use Twilio to send text messages to users.

Timer: The timer is used to realize the function of sending reminders regularly.



Code Architecture

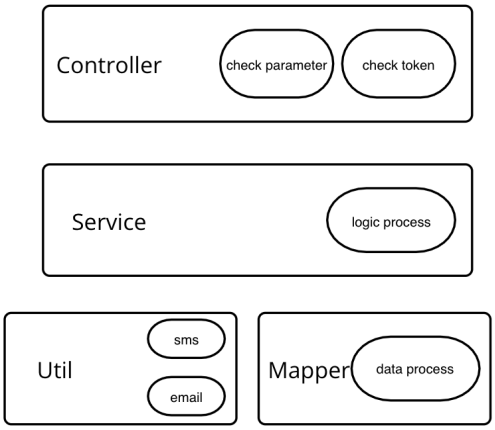
For the design of code, we use Spring Boot Framework and Spring WS. The layout is below.

Controller: controller is the portal of the service. The main work is check and parse the parameter, check the user token for security.

Service: The service is the main part of handling logical transactions.

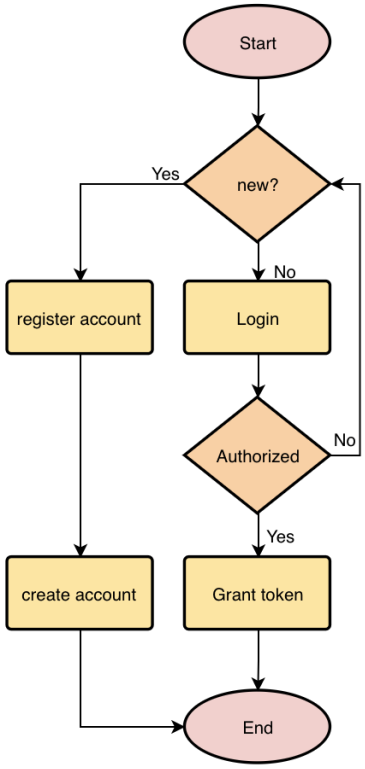
Mapper: The mapper is responsible for the interaction with the database. His dependency is mybatis framework.

Util: Util contains a lot of tool classes, which is a lower level.



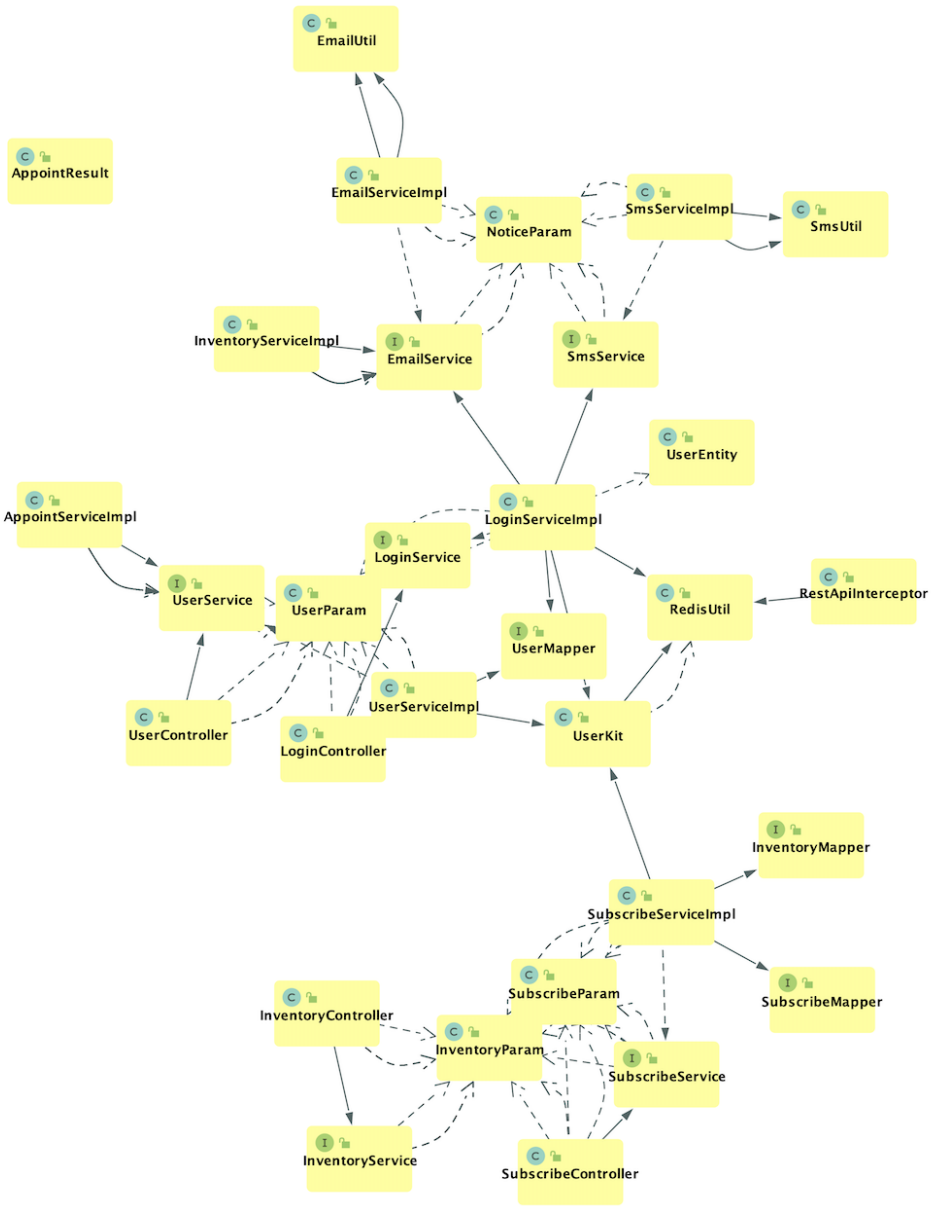
Sample Diagram

It is the login and register module.



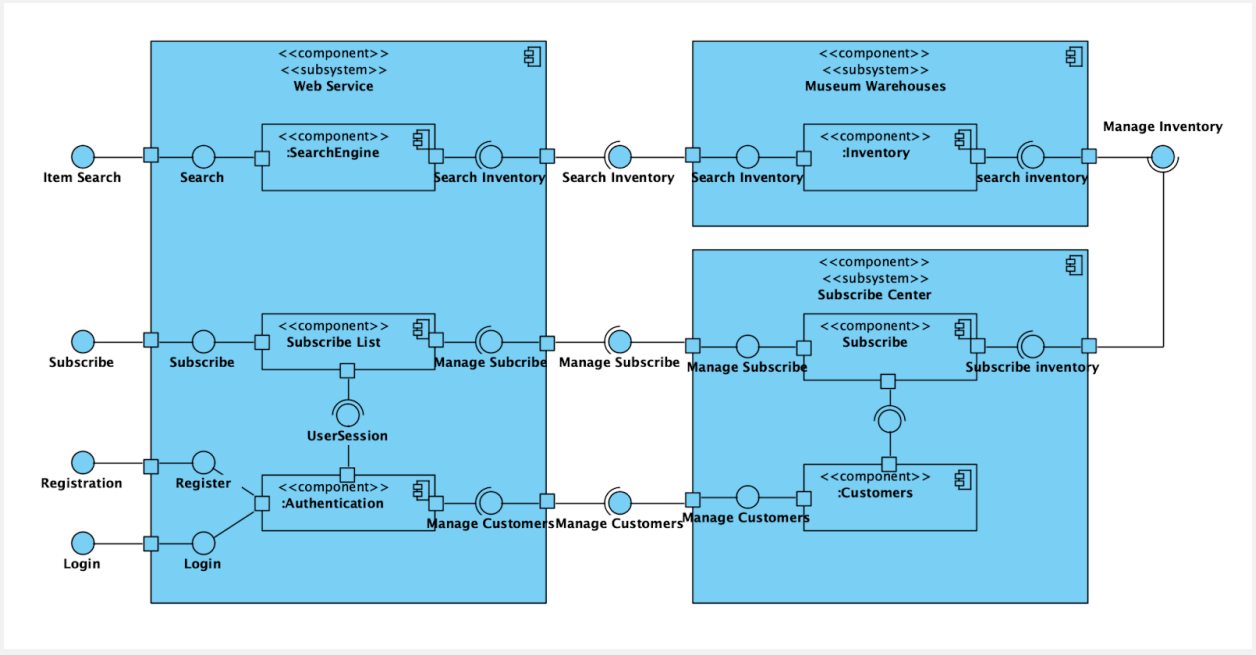
Class Architecture

There is a snapshot diagram of the classes, which contains most of classes the their connection.



Component Diagram

The component diagram depicts how components are wired together to form larger components or software systems. There are three components: Web service, Warehouse and Subscribe center.



Component Diagram

Sequence Diagram

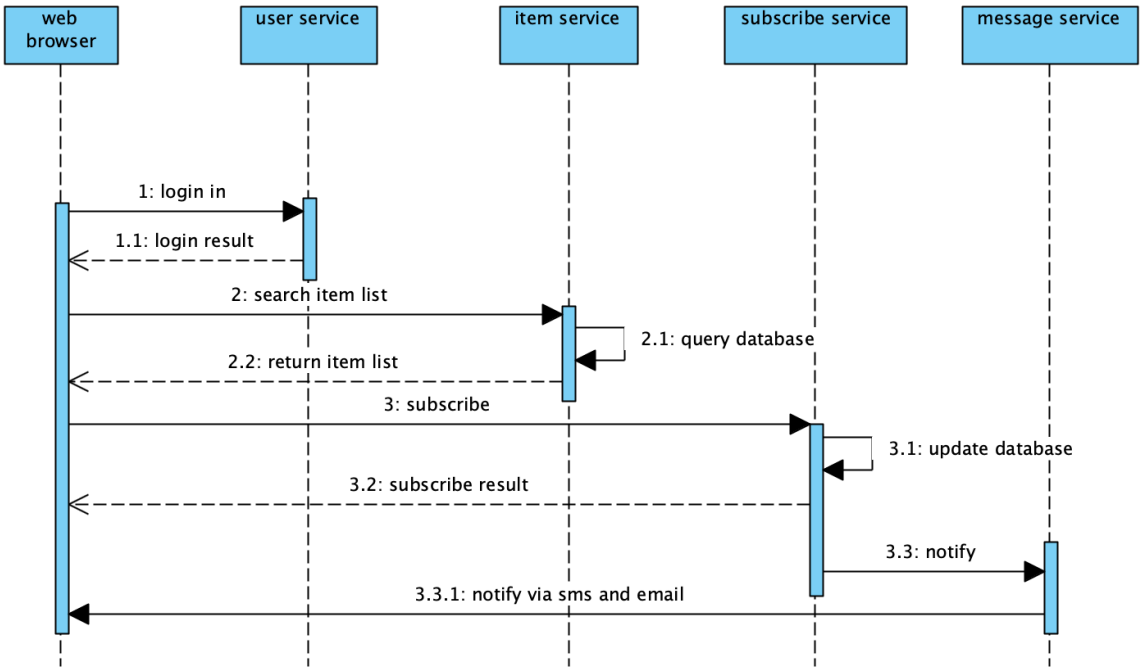
The sequence diagram shows such a process:

1. The user logs in or registers the system.

2. Users browse the cultural relics in the museum collection and find them through search.

3. The user initiates the attention subscription for the cultural relic.

4. The system sends a reminder message (exhibition time and place) to the user through SMS or e-mail.



Sequence Diagram

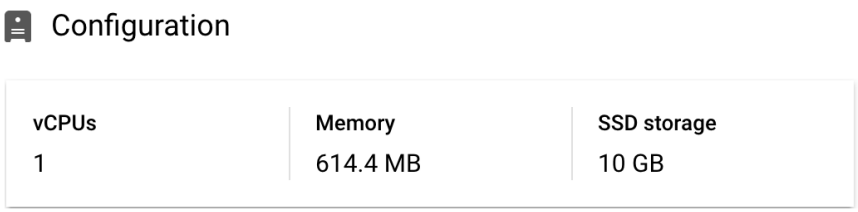
Appendix B: Database Design and Sample Data

Database

We use two kinds of databases, relational databases(**MySQL**) and NoSQL databases(**Redis**).

MySQL

We build an SQL instance at Google Cloud Platform, The version of Mysql is 5.7. Below is the configuration information.



Redis

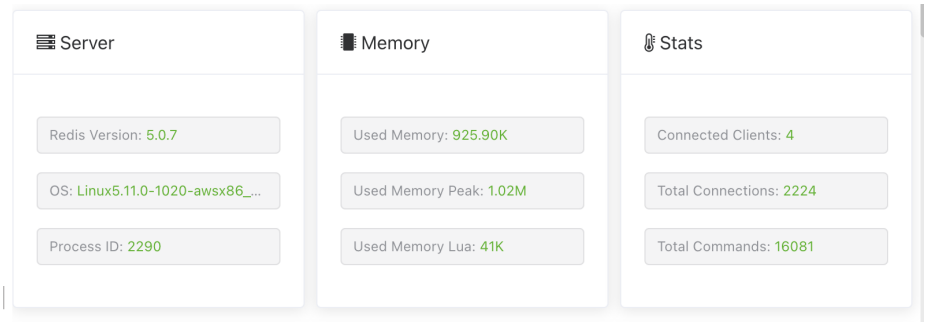
We installed a Redis Server at EC2 instance on AWS(Amazon Web Service), the version of Redis is 5.0.7.

Redis is used to save token and user information.

Why Redis?

Redis works with an in-memory dataset.it is very fast to read data. Obtaining current user information is a very frequent operation.

Below is the configuration information.

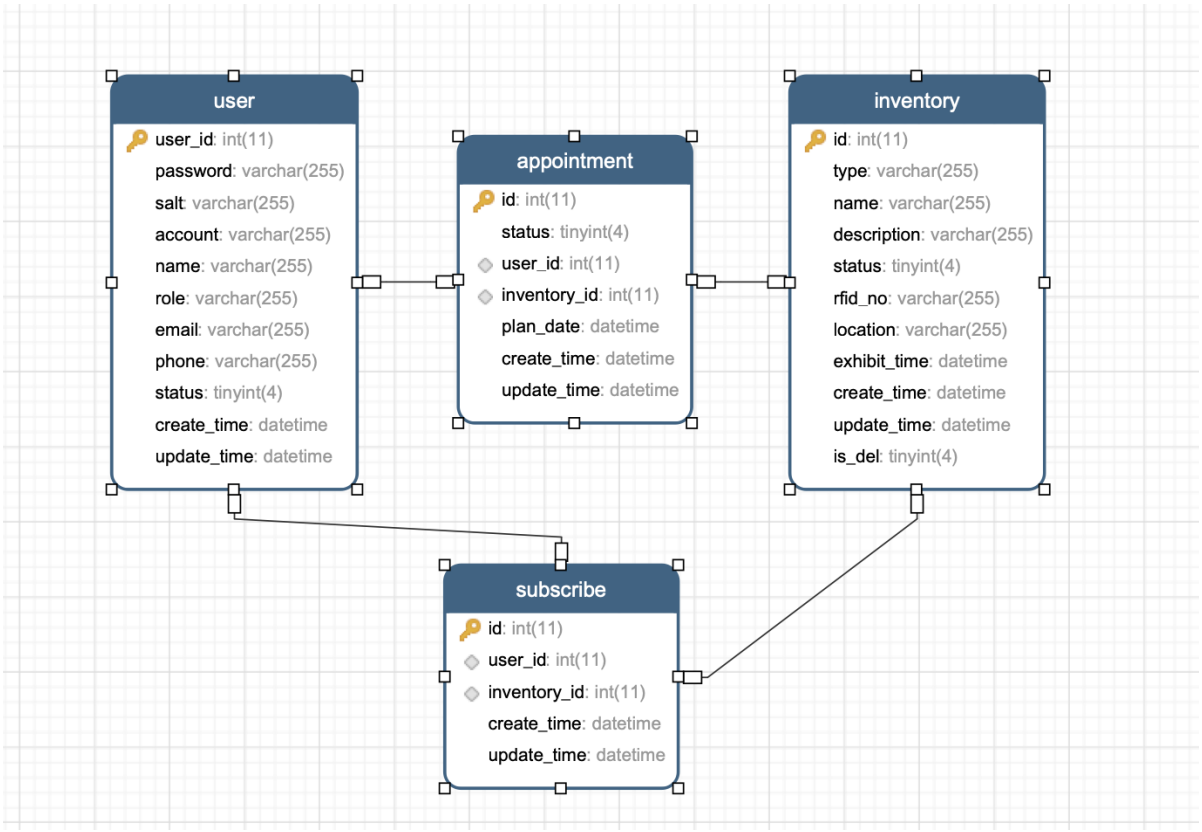


Schema

Here is the E-R model diagram.

We have 4 tables: `user`, `inventory`, `subscribe` and `appointment`.

- User: contain all user information.
- Inventory: contain all museum inventory and status.
- Subscribe: save subscription record.
- Appointment: save appointment record.



There are 2 Foreign keys for both `subscribe` and `appointment`.

```
FOREIGN KEY (`user_id`) REFERENCES `user` (`user_id`)
FOREIGN KEY (`inventory_id`) REFERENCES `inventory` (`id`)
```

Sample Data

MySQL

We have 4 tables: `user`, `inventory`, `subscribe` and `appointment`.

- User: contain all user information.

- Inventory: contain all museum inventory and status.
- Subscribe: save subscription record.
- Appointment: save appointment record.

Below is the sample data of `user`. It contains id, password, account, which is essential, and other personal information, such as phone and email, which can be used to connect via sms and email service.

user_id	password	salt	account	name	role	email	phone	status	create_time	update_time
1	313f3219d90eec595eecbbf542797063	SQlGl	guanchen	Guanchen Zhao	admin	gczhao@uw.edu	2065748662	1	2021-12-10 03:	2021-12-10 03:03:06
2	313f3219d90eec595eecbbf542797063	SQlGl	gczhao	G Z	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-09 23	2021-12-09 23:35:13
3	91c9fc2b474918df9c07b356e7ca19ea	TLqWt	a2	Guanchen Zhao	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-10 03:	2021-12-10 03:04:57
4	85638cc235b5a7ea078ded7e51fa40dc	ZyPHe	a3	Guanchen Zhao	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-10 03:	2021-12-10 03:05:09
10	313f3219d90eec595eecbbf542797063	Djwfo	a4	Guanchen Zhao	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-10 03:	2021-12-10 03:05:33
11	818ed0d882677b77f4ecc20fb1cae1c1	SQlGl	a5	Huicong J	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-10 03:	2021-12-10 03:05:30
12	008fcc87a6a48cfcdb61161e4fea9a67	Tkjti	a6	Guanchen Zhao	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-10 03:	2021-12-10 03:00:41
13	313f3219d90eec595eecbbf542797063	SQlGl	a7	Guanchen Zhao	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-10 03:	2021-12-10 03:00:41
14	313f3219d90eec595eecbbf542797063	SQlGl	a8	Guanchen Zhao	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-02 05	2021-12-02 05:02:27
15	313f3219d90eec595eecbbf542797063	SQlGl	a9	Guanchen Zhao	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-02 05	2021-12-02 05:02:27
16	e78d244f53856eaa7c6ff583c0722cfe	sdjcx	a10	Guanchen Zhao	normal	guanchenzhao@gmail.com	9999991234	1	2021-12-09 21:	2021-12-09 13:08:56
17	313f3219d90eec595eecbbf542797063	SQlGl	admin	Guanchen Zhao	admin	guanchenzhao@gmail.com	2065748662	1	2021-12-10 03:	2021-12-10 03:00:55
18	313f3219d90eec595eecbbf542797063	SQlGl	a12	Guanchen Zhao	normal	guanchenzhao@gmail.com	2065748662	1	2021-12-02 05	2021-12-02 05:02:27

Below is the sample data of `appointment`. It contains associated userid and inventory, also the plan date.

id	status	user_id	inventory_id	plan_date	create_time	update_time
1	1	1	1	2021-12-02 23:17:39	2021-12-09 15:33:57	2021-12-09 15:34:00
2	1	1	1	2021-12-03 20:19:35	2021-12-03 18:16:40	2021-12-09 19:06:09
3	0	2	1	2021-09-20 23:09:00	2021-12-03 18:25:31	2021-12-03 18:25:31
4	0	1	2	2021-12-07 22:26:53	2021-12-09 15:34:11	2021-12-09 15:34:08
5	0	10	10	2021-09-20 23:09:00	2021-12-09 16:38:32	2021-12-09 16:38:32

Below is the sample data of `inventory`. It contains id, type, name, which is essential, and other status information, such as status(repair, show, store) and location. The rfid_no is the number of RFID,which is bound to the inventory item one by one.

The RFID can be used to get location.

id	type	name	description	status	rfid_no	location	exhibit_time	create_time	update_time	is_del
1	Painting	Mona Lisa	amazing	2	a01	F1, A1	2021-12-22 13:00:0(	2021-12-09 23:29:4	2021-12-09 23:29	0
2	Painting	Sunflower	Beautiful	2	a02	F1, A2	2021-12-09 02:37:00	2021-12-09 23:30:26	2021-12-09 23:30:26	0
3	Painting	Roses	Beautiful	3	a03	F2, A2	2021-12-09 13:00:00	2021-12-09 23:30:34	2021-12-09 23:30:34	0
4	Instrumer	piano	Beautiful	2	a04	F1, A5	2021-12-01 21:00:00	2021-12-09 23:31:58	2021-12-09 23:31:58	0
5	Painting	Roses	Beautiful	0	a05	F3, A2	2021-12-01 13:00:00	2021-12-09 23:33:27	2021-12-09 23:33:27	0
6	Painting	Mona Lisa	Beautiful	2	a06	F1, A2	2021-12-01 21:00:00	2021-12-09 23:29:50	2021-12-09 23:29:50	0
7	porcelain	las	Beautiful	2	a07	F4, A2	2021-12-01 21:00:00	2021-12-09 23:33:04	2021-12-09 23:33:04	0
8	Painting	Mona Lisa	Beautiful	2	a08	F1, A2	2021-12-01 21:00:00	2021-12-09 23:29:51	2021-12-09 23:29:51	0
9	Painting	Mona Lisa	Beautiful	2	a09	F1, A2	2021-12-01 21:00:00	2021-12-09 23:29:51	2021-12-09 23:29:51	0
10	Painting	Mona Lisa	Beautiful	2	a10	F1, A1	2021-12-01 21:00:00	2021-12-09 23:29:51	2021-12-09 23:29:51	0
11	Painting	Mona Lisa	Beautiful	2	a11	F2, A1	2021-12-01 21:00:00	2021-12-09 23:29:51	2021-12-09 23:29:51	0
12	Painting	Mona Lisa	Beautiful	2	a12	F1, A1	2021-12-01 21:00:00	2021-12-09 23:29:52	2021-12-09 23:29:52	0
13	Painting	Mona Lisa	Beautiful	2	a13	F1, A1	2021-12-01 21:00:00	2021-12-09 23:29:52	2021-12-09 23:29:52	0
14	Painting	Mona Lisa	Beautiful	2	a14	F1, A1	2021-12-01 21:00:00	2021-12-09 23:29:52	2021-12-09 23:29:52	0
15	Painting	Mona Lisa	Beautiful	2	a15	F1, A2	2021-12-01 21:00:00	2021-12-09 23:29:52	2021-12-09 23:29:52	0

Below is the sample data of `subscribe`. It contains associated userid and inventory, indicating that a user subscribes to an inventory item.

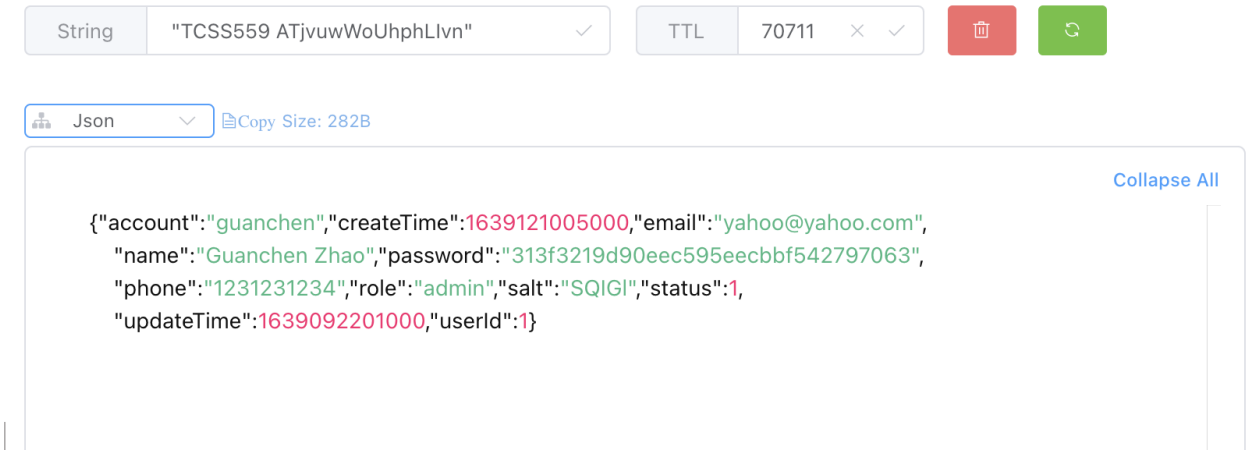
id	user_id	inventory_id	create_time	update_time
1	1	2	2021-12-02 16:55:54	2021-12-02 16:55:54
3	0	3	2021-12-09 15:34:39	2021-12-09 13:15:21
4	1	1	2021-12-04 05:23:26	2021-12-04 05:23:26
5	2	1	2021-12-09 13:15:21	2021-12-09 13:15:21
6	3	2	2021-12-09 13:15:21	2021-12-09 13:15:21
7	1	10	2021-12-09 13:15:21	2021-12-09 13:15:21
9	2	2	2021-12-08 06:25:57	2021-12-08 06:25:57
10	2	3	2021-12-09 13:15:21	2021-12-09 23:34:33
11	2	4	2021-12-09 13:15:21	2021-12-09 23:34:33
12	1	12	2021-12-09 13:15:21	2021-12-09 23:34:33

Redis

Redis is used to save token and user information.

Token is the token for login and authorization verification, which can uniquely identify a user. The request header of every request is required to carry it. With the token, we can find the user information.

Below is the sample data of the token.



Appendix C: Web API Design

Please click this link to review the web API design.

Link

https://documenter.getpostman.com/view/3290385/UVR4M9nP

Appendix D: Screenshots of Front-End interface

Description

Bonanza is a single page web application developed using React https://reactjs.org/ , which is a JavaScript library for building user interfaces.

In this part, we will demonstrate our web by pages and views.

Pages

Pages contains several modules: login, inventory, subscribe, setting and about.

Module Login

This is the login part, which contains login, register and logout.

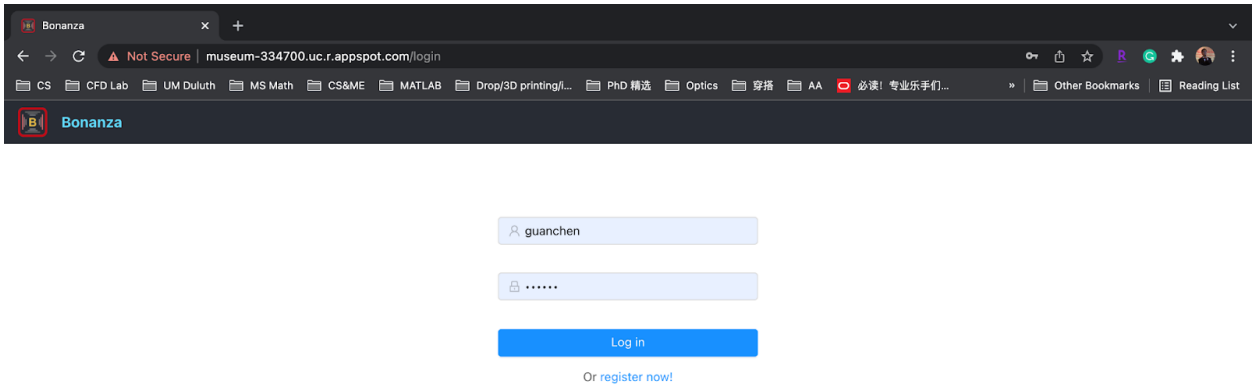
Login page

You need to provide an account and password.

Here is a screenshot.

interface

POST ims/login/login



Register page

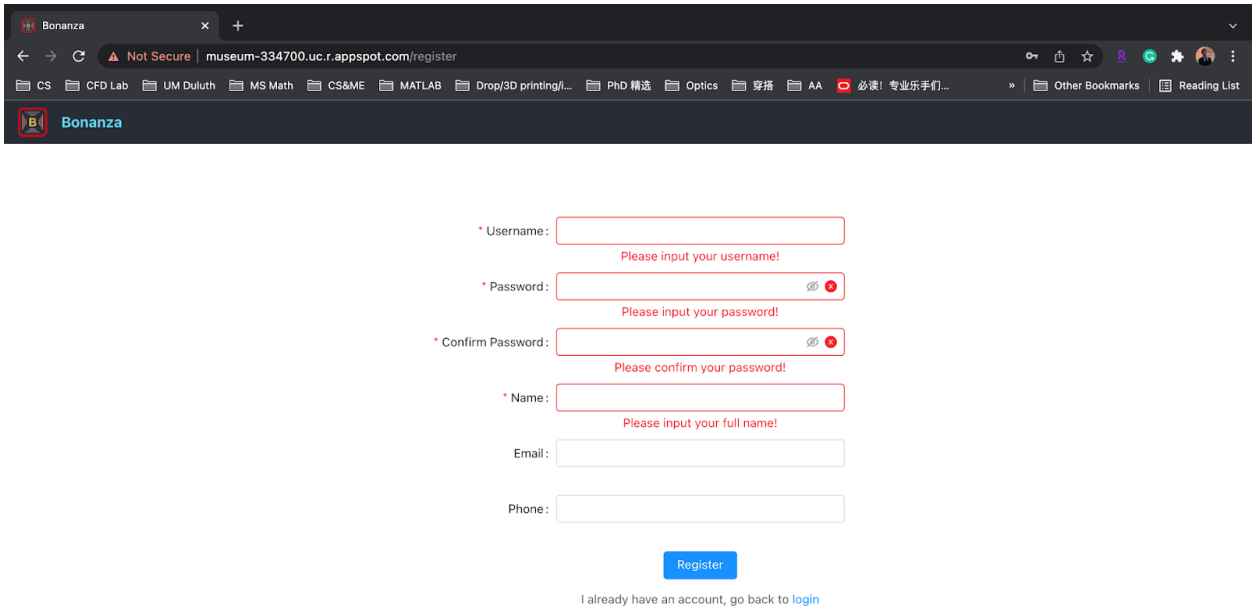
There are many checks on required items.

You need to provide an account name, and password. Email and phone are optional.

Here is a screenshot.

interface

POST /ims/login/register



Module Inventory

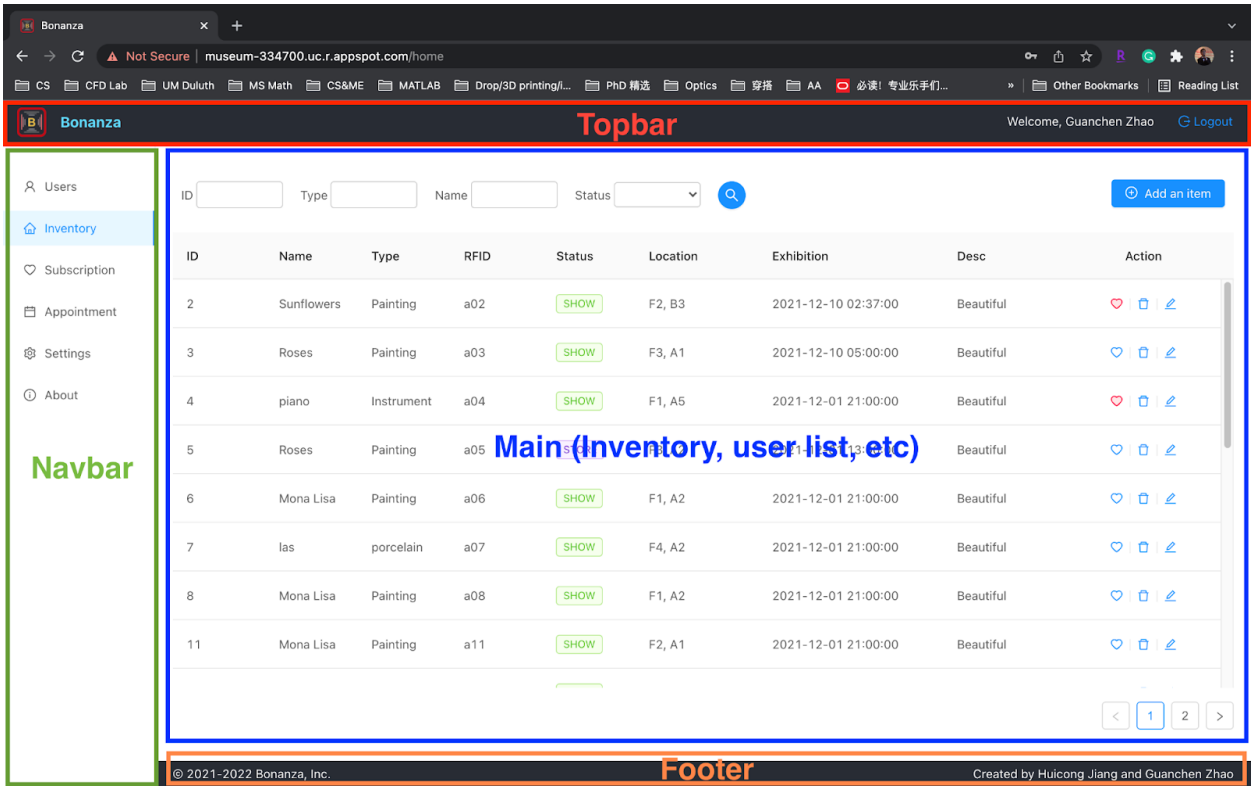
This is the inventory part, also the home page, Which contains the view and operation of inventory, such as update, subscribe and cancel.

Note that the list is paged.

The edit button is only open to admin roles.

General layout

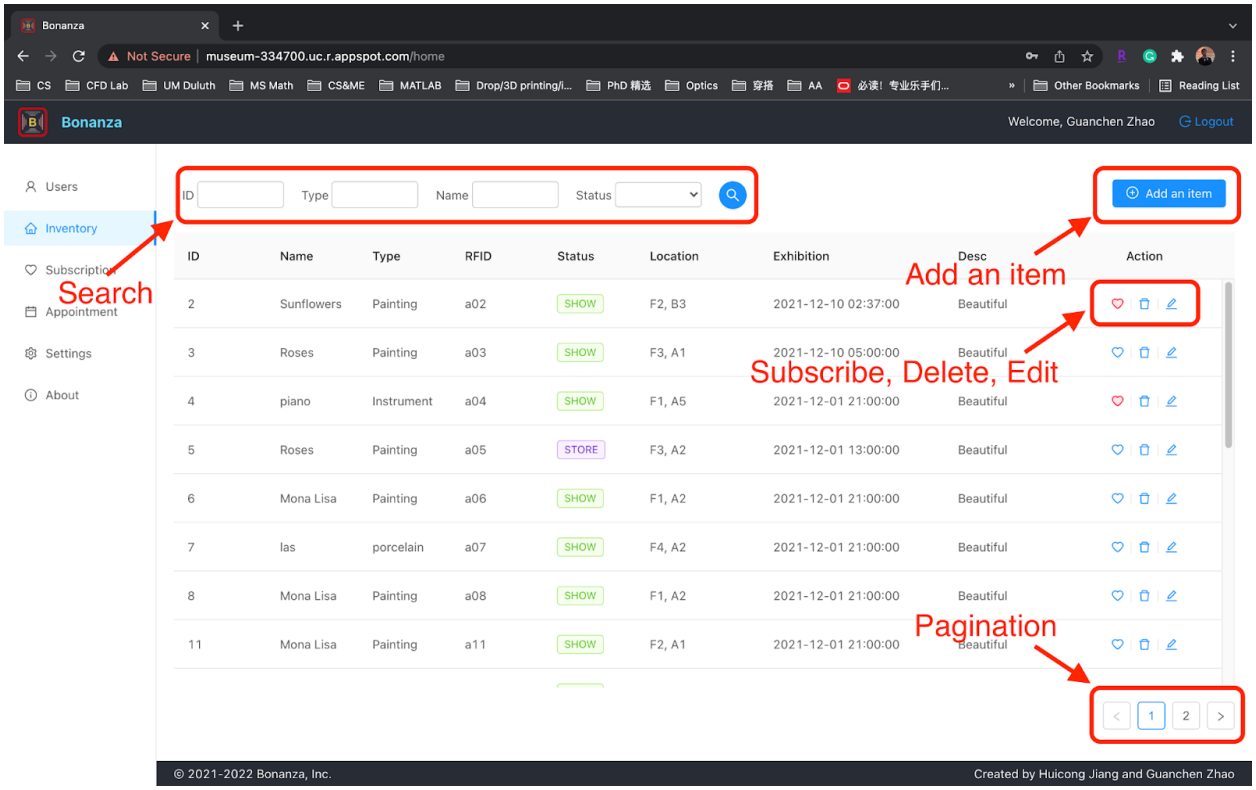
Here is a screenshot, which contains the layout design.



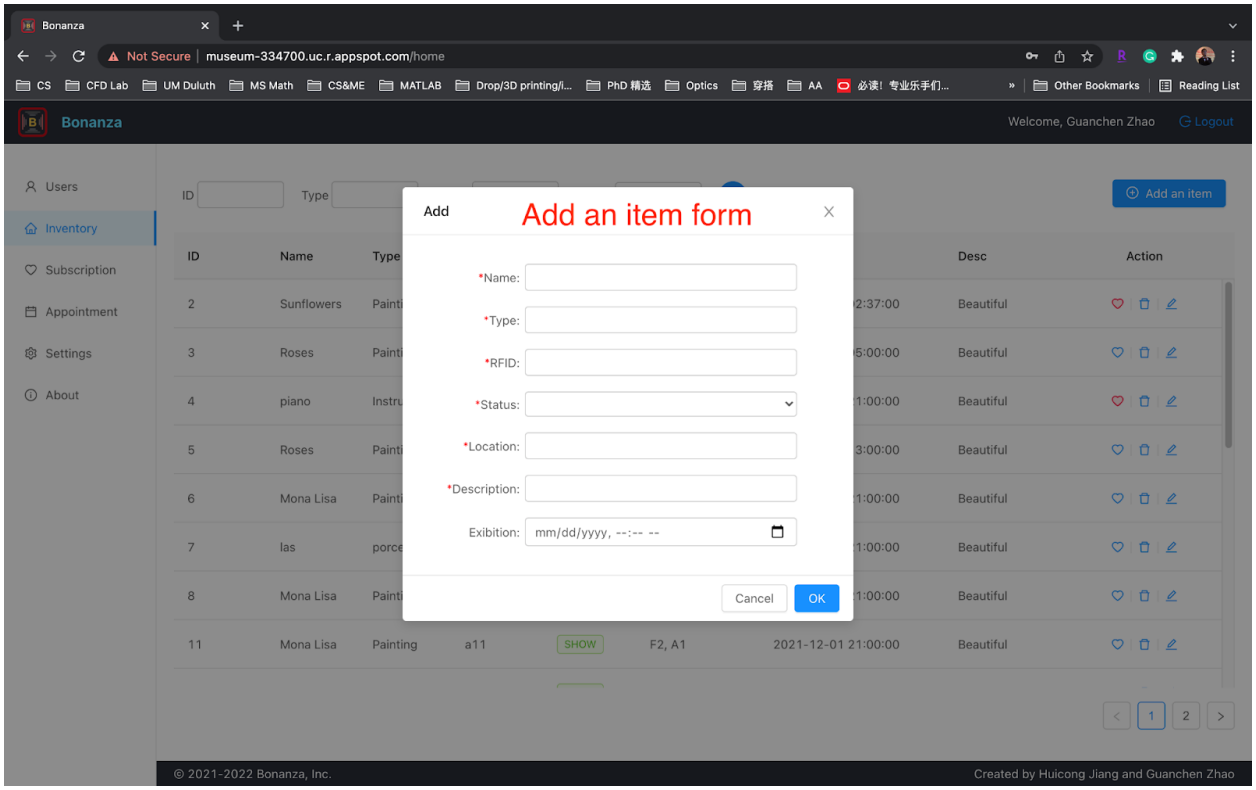
Interface

- Search: GET /ims/inventory/list
- Add an item: POST /ims/inventory/add
- Subscribe an item: POST /ims/subscribe/add
- Cancel Subscription: POST /ims/subscribe/cancel
- Delete an item: DELETE /ims/inventory/remove
- Update an item: PUT /ims/inventory/update
- Pagination: GET /ims/inventory/pageList

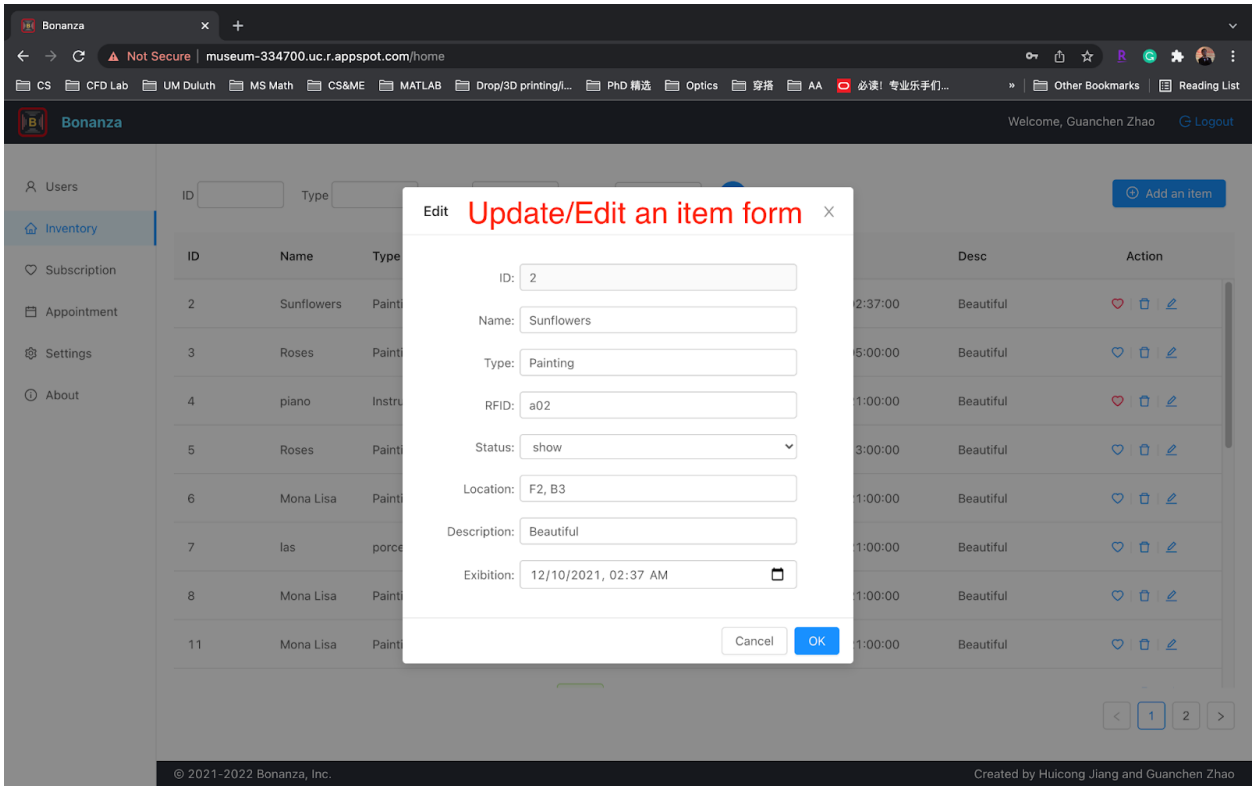
inventory


Bonanza
Not Secure
museum-334700.uc.r.appspot.com/home
Bonanza
Welcome, Guanchen Zhao
Logout
Users
Inventory
Subscription
Appointment
Settings
About
ID
Type
Name
Status
Add an item
Search
Add an item
Subscribe, Delete, Edit
Pagination
ID
Name
Type
RFID
Status
Location
Exhibition
Desc
Action
2 Sunflowers Painting a02 SHOW F2, B3 2021-12-10 02:37:00 Beautiful
3 Roses Painting a03 SHOW F3, A1 2021-12-10 05:00:00 Beautiful
4 piano Instrument a04 SHOW F1, A5 2021-12-01 21:00:00 Beautiful
5 Roses Painting a05 STORE F3, A2 2021-12-01 13:00:00 Beautiful
6 Mona Lisa Painting a06 SHOW F1, A2 2021-12-01 21:00:00 Beautiful
7 las porcelain a07 SHOW F4, A2 2021-12-01 21:00:00 Beautiful
8 Mona Lisa Painting a08 SHOW F1, A2 2021-12-01 21:00:00 Beautiful
11 Mona Lisa Painting a11 SHOW F2, A1 2021-12-01 21:00:00 Beautiful
1
2
© 2021-2022 Bonanza, Inc.
Created by Huicong Jiang and Guanchen Zhao

Add inventory



Update inventory



Module User

This is the user part, Which contains the view and operation of the user, such as add, search and delete.

This part is only open to admin roles.

Note that the list is paged.

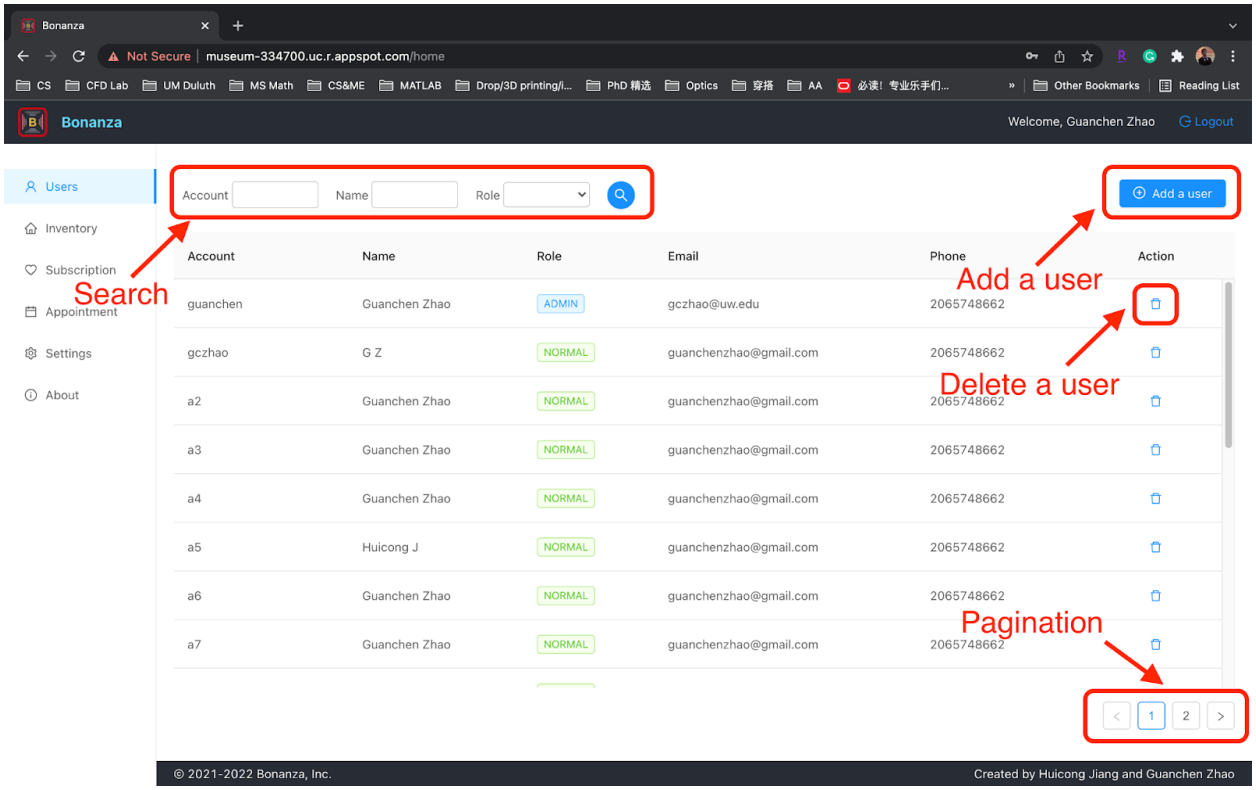
General layout

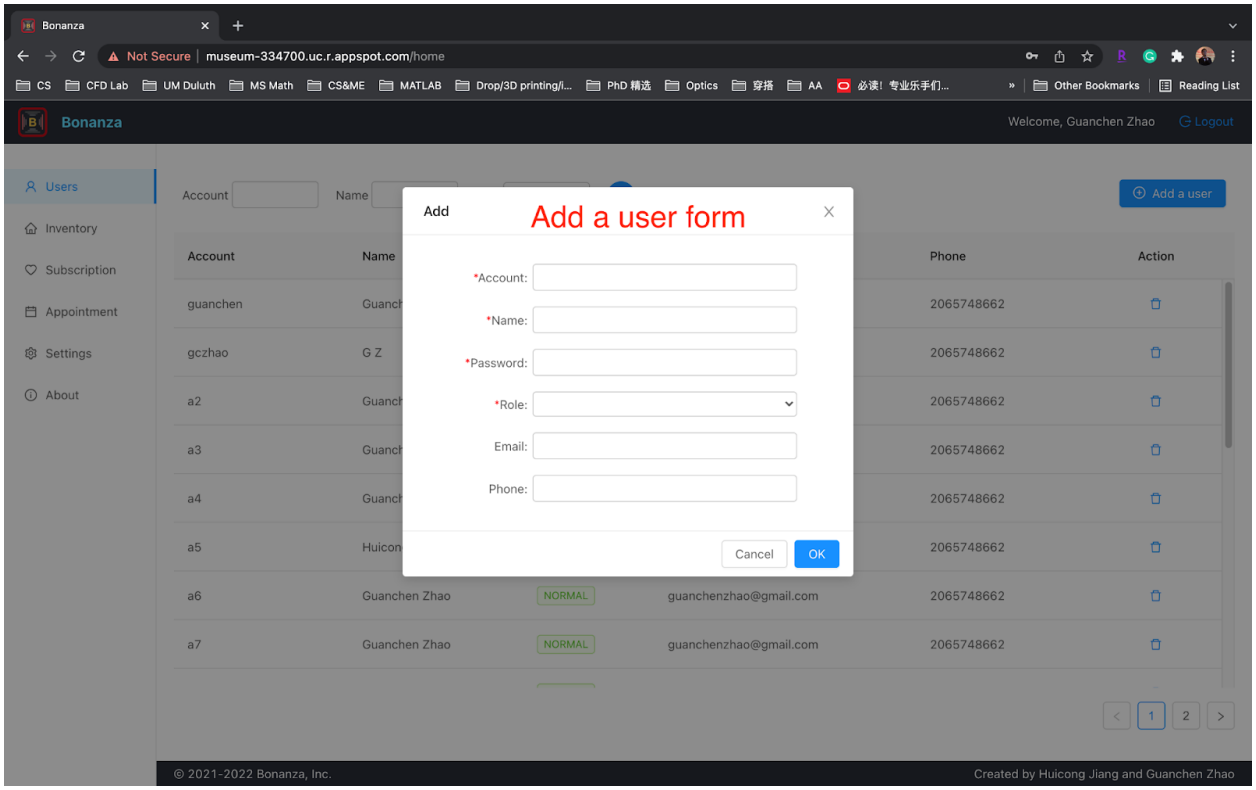
interface

- Search: GET /ims/user/pageList
- Add a user: POST /ims/user/add

- Delete a user: PUT /ims/user/remove
- Pagination: GET /ims/user/pageList

Here is a screenshot, which contains the layout design.





Module Subscribe

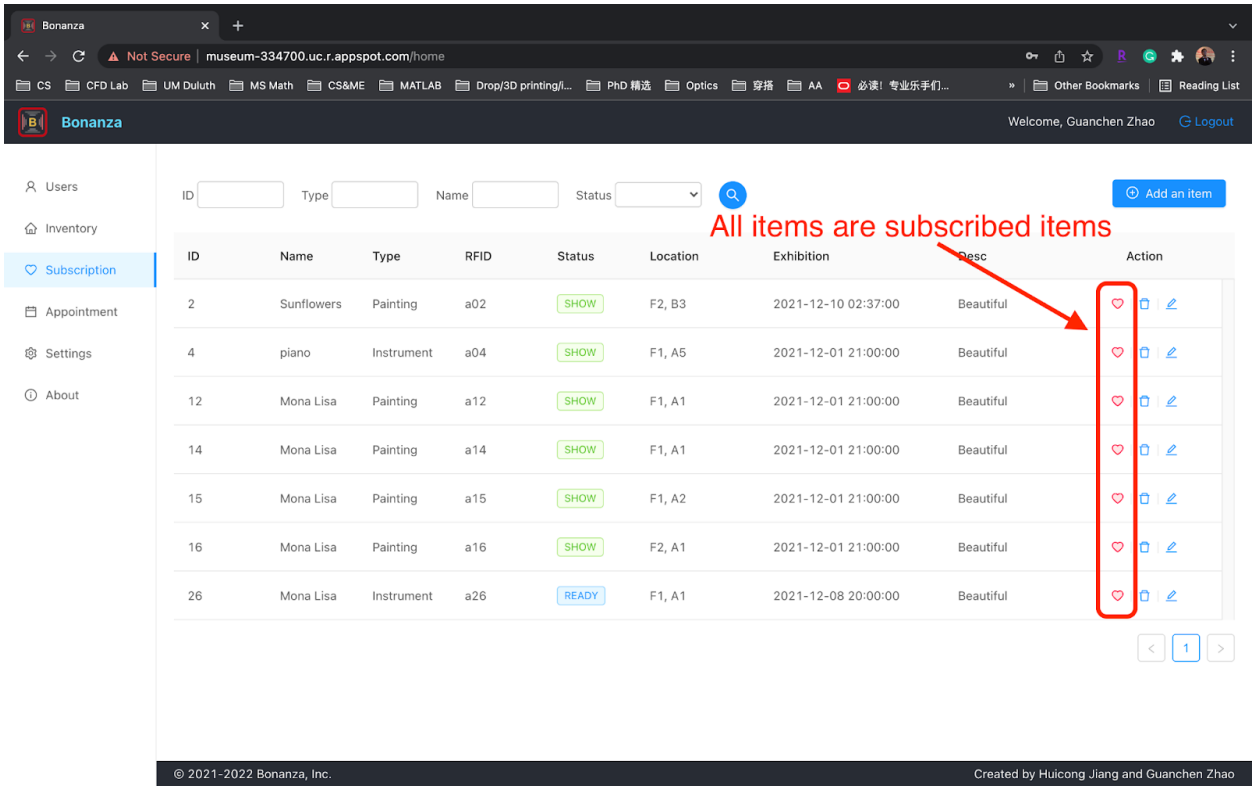
This is the subscribe part, Which contains the view and operation of the subscribe, such as list, add subscription and cancel.

Note that the list is paged. The list will only show the items subscribed by the current login user.

General layout

Here is a screenshot, which contains the layout design.

interface



Module Setting

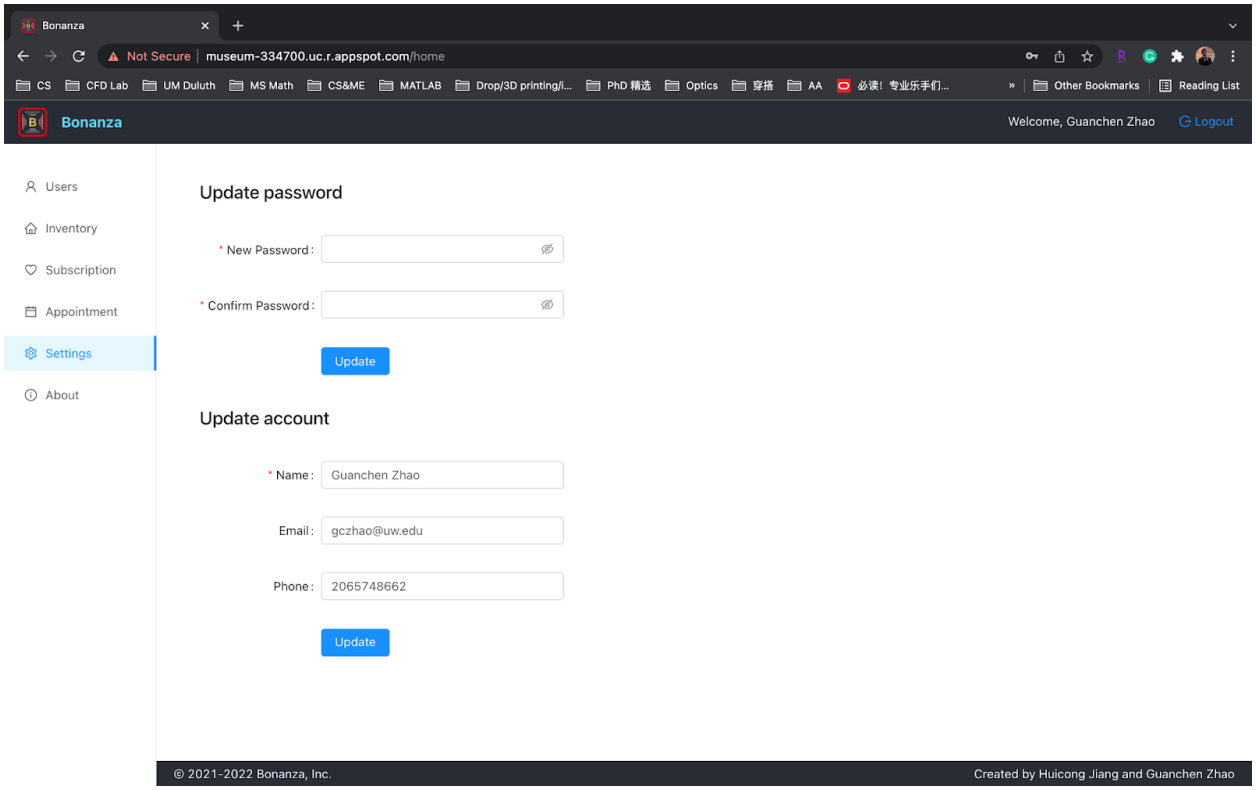
This is the setting part, Which contains the view and operation of the user, such as updating password, email, name and phone.

Note that the page will only show the information of the current login user.

Interface

- Update password: PUT /ims/user/updatePassword
- Update account info: POST /ims/user/update

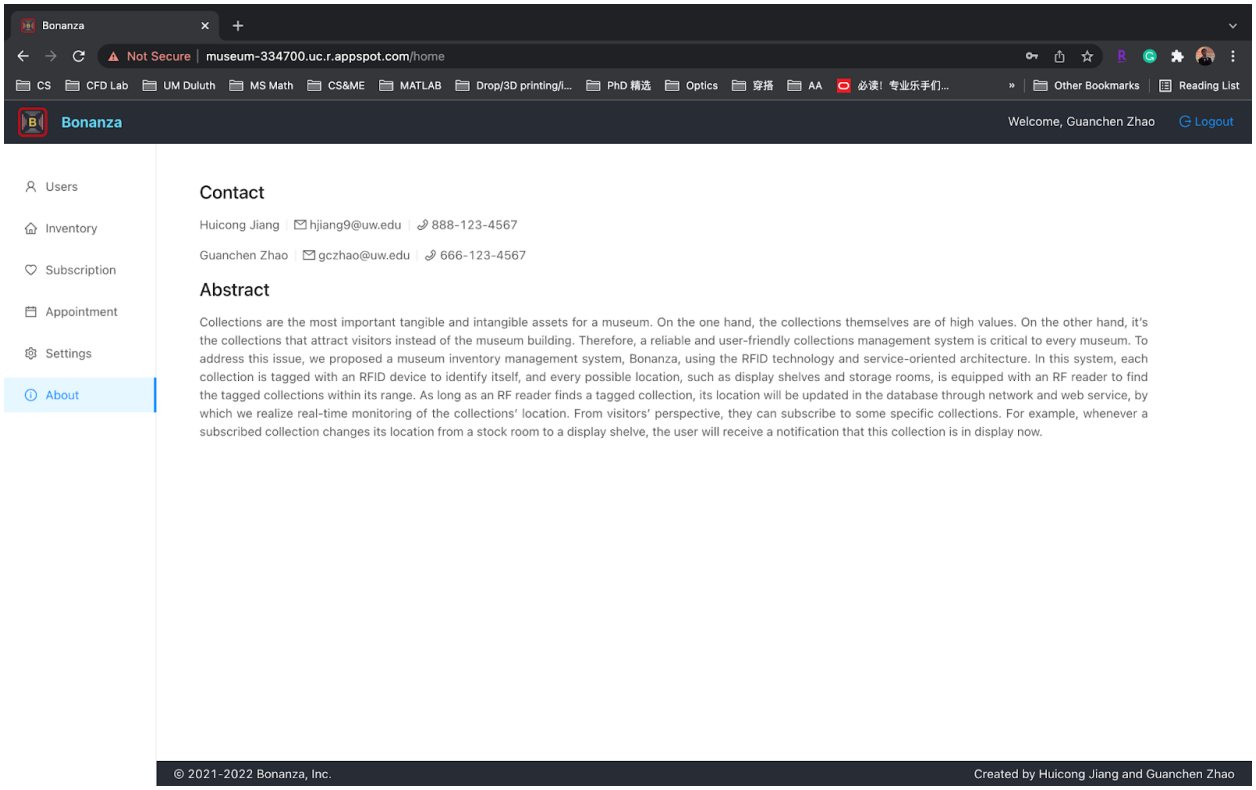
Here is a screenshot, which contains the layout design.



Module About

This is the about part, Which contains the information of the contact and museum.

Here is a screenshot, which contains the layout design.


Bonanza
Not Secure | museum-334700.uc.r.appspot.com/home
CS
CFD Lab
UM Duluth
MS Math
CS&ME
MATLAB
Drop/3D printing/i...
PhD 精选
Optics
穿搭
AA
必读！专业乐手们...
Other Bookmarks
Reading List
Bonanza
Welcome, Guanchen Zhao
Logout
Users
Inventory
Subscription
Appointment
Settings
About
Contact
Huicong Jiang | hjiang9@uw.edu | 888-123-4567
Guanchen Zhao | gczhao@uw.edu | 666-123-4567
Abstract
Collections are the most important tangible and intangible assets for a museum. On the one hand, the collections themselves are of high values. On the other hand, it's the collections that attract visitors instead of the museum building. Therefore, a reliable and user-friendly collections management system is critical to every museum. To address this issue, we proposed a museum inventory management system, Bonanza, using the RFID technology and service-oriented architecture. In this system, each collection is tagged with an RFID device to identify itself, and every possible location, such as display shelves and storage rooms, is equipped with an RF reader to find the tagged collections within its range. As long as an RF reader finds a tagged collection, its location will be updated in the database through network and web service, by which we realize real-time monitoring of the collections' location. From visitors' perspective, they can subscribe to some specific collections. For example, whenever a subscribed collection changes its location from a stock room to a display shelve, the user will receive a notification that this collection is in display now.
© 2021-2022 Bonanza, Inc.
Created by Huicong Jiang and Guanchen Zhao