Đỗ Hùng Cường

ITITIU13170

Thesis outline

0/ Abstraction: summary of all thesis – viết sau

List of table

List of figure

Definition

|  |  |
| --- | --- |
| Term | Descriptions |
| Guest |  |
| Customer |  |
| Administrator |  |
| HTML |  |
| CSS |  |
| Boostrap |  |
| Jquery |  |
| IDE |  |
| UML |  |
| Linux |  |

Table of content

I/ Introduction

1/ Situation without management system & difficulties with old management system

* Customer must come to hotel to book room => if huge number of customer come at the same time => have to wait => waste time and money => uncomfortable
* Booking based on pen and paper => not convenient for both the customers and the receptionists
* Management is very difficult *(specially in enormous hotels)*
* Reservation might cause a lot of risk *(invalid information of customers, wrong information of rooms or bookings)*
* Solution is management system
* But there are some difficulties with old management system
* Performance might be very bad
* Could overload or run extremely slow when a huge number of users access at the same time
* Look and feed *(User interface was not designed beautifully)*
* Not pleased to use

2/ Modern Hotel Booking System

* Many deluxe hotels or five-star hotels in the world *(Marriott International, Hilton Worldwide or InterContinental Hotels Group)* already have their own hotel booking systems.
* Friendly user interface
* High performance
* Ability to track the behavior of customers.
* The administrators, the managers or hotel owners could know what customer had done on their websites. *(which pages customers clicked on, how long customers stayed at each page, which rooms, which services that customers had searched, booked, ordered or send the feedbacks)*
* Based on the data collection, the systems will automatically suggest what customers may like, recommend which rooms customers should book.
* The hotel owners can improve their hotel business based on the information collected by their systems.

3/ New Technology

3.1/ Single Page Application

* Single Page Application is a web application that on only one single web page or only one index page contains dynamic actions which we do not need to refresh the page.
* Single Page Application interactions can be handle without reaching server.
* Single Page Application can improve performance in many cases such as loading time, using AJAX, easy to navigate pages etc.
* That makes the end users feel more comfortable when using Single Page Application.

3.2/ Mongodb:

* NoSQL, open-source database
* stores data in JSON-like documents that can vary in structure
* Related information is stored together for fast query
* Dynamic schemas => can create records without first defining the structure
* can change the structure of records simply by adding new fields or deleting existing ones.
* represent hierarchical relationships, to store arrays, and other more complex structures easily
* Documents in a collection need not have an identical set of fields and denormalization of data is common.
* MongoDB was also designed with high availability and scalability in mind, and includes out-of-the-box replication and auto-sharding.

3.3/ Nodejs & Express framework

* Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine.
* Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.
* Node.js package ecosystem, npm, is the largest ecosystem of open source libraries in the world
* Real time => can build Single page application easily
* Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
* Advantage of Nodejs combine with Express framework:
* APIs

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

* Performance

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

* Work well with json => easily work with mongodb

3.4/ Angular 2:

Angular allows us to build a single page application easily.

Recently, many frameworks, platforms or techniques were released to support building a Single Page Application.

Angular is one of the most popular Single Page Application framework

* Angular makes the HTML more expressive by support some features such as if-else condition, switch-case, loop and local variable.
* Angular has powerful data binding. Thank to data binding, we can easily display variables from the data model such as component, track changes, and process updates from the user.
* Angular promotes modularity by design. Every Angular application is a set of building blocks and that is easier to create and reuse content.
* Angular has built-in support for communication with a back-end service. In Angular application, it is easy for the front-end to integrate with a backend server to get and post data or execute server-side business logic.

3.5/ Sping MVC:

* Most powerful J2EE framework to build Java web application
* open source Java platform that provides comprehensive infrastructure support for developing robust Java based Web applications very easily and very rapidly.
* provides a model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications.
* The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

4/ Goals and Scope

* Experiment successfully MEAN + Spring MVC to implement Hotel Booking & Reservation System
* with more than 120 features which are similar to those five-star hotel booking system’s features.
* Online single page application with high performance
* Dynamically loading
* Cross-platform system runs well with all operating system.
* Friendly user interfaces
* Supports almost features for hotel bookings & reservations management.
* Ability to track user’s behavior
* Provide data collection for applying AI machine learning in the feature

II/ Background

1/ MVC:

general architecture of the whole system is MVC (Model – View -Controller)

Model

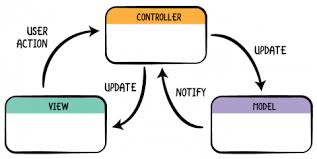
The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

View

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

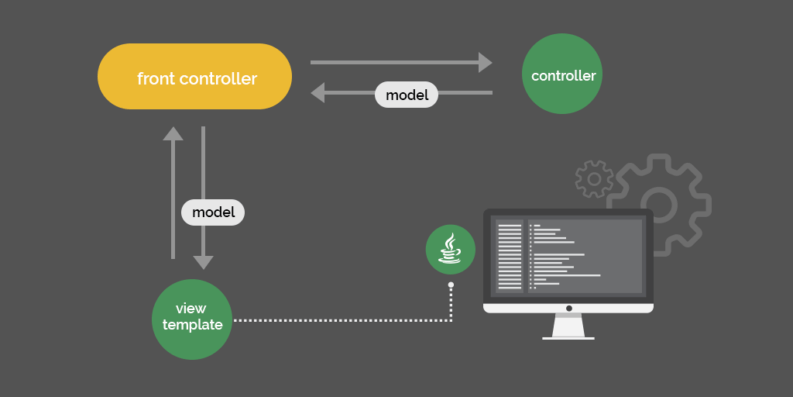
Controller

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

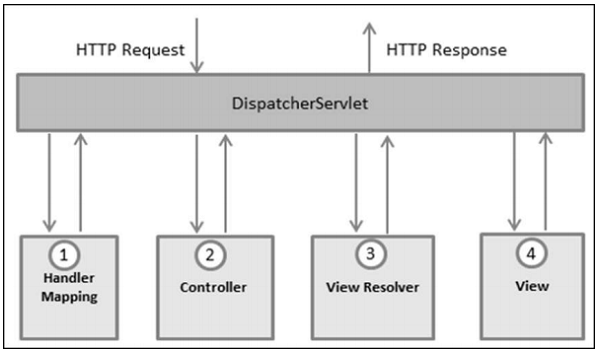


2/ Spring MVC

* Front-Controller: Is a initial level of contract point for handling a request.
* The front controller provides a centralized entry point for that controls and manages web request handling. By centralizing decision point and controls.
* The front controller also reduce java code and business logic by promoting code reuse ability across the requests.
* The front controller coordinates with dispatcher components. Dispatcher are responsible to view management only, the one who bring the view components to the user is called dispatcher.



The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC DispatcherServlet is shown in the following illustration.



Following is the sequence of events corresponding to an incoming HTTP request to DispatcherServlet −

* After receiving an HTTP request, DispatcherServlet consults the **HandlerMapping** to call the appropriate Controller.
* The Controller takes the request and calls the appropriate service methods based on used **GET** or **POST method**. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.
* The DispatcherServlet will take help from **ViewResolver** to pick up the defined view for the request.
* Once view is finalized, The DispatcherServlet passes the model data to the view, which is finally rendered, on the browsers.

All the above-mentioned components, i.e. HandlerMapping, Controller and ViewResolver are parts of **WebApplicationContext**, which is an extension of the plain **ApplicationContext** with some extra features necessary for web applications.

3/ Angular 2

There are 8 main building blocks of an Angular application in this architecture diagram: Modules, Components, Templates, Metadata, Data binding, Directives, Services and Dependency injection.

In each Angular application, there must be one or more Angular module class. The [root module](https://angular.io/docs/ts/latest/guide/appmodule.html) is always available in every Angular app.

In a small applications, they may have only one root module. However, almost larger applications may have the root module and many feature modules.

A component controls the view. For instance, some views of a page are controlled by components such as the header, the footer, the sidebar of a page.

We define the application logic in class of the component. That can be fields or functions which support the view. The class and the view interacts with each other through an API of properties and methods.

We define the view of a component by the **template**. We can write html code or put html code in the html file to build the template that tells Angular how to render the component

Metadata tells Angular how to process a class.

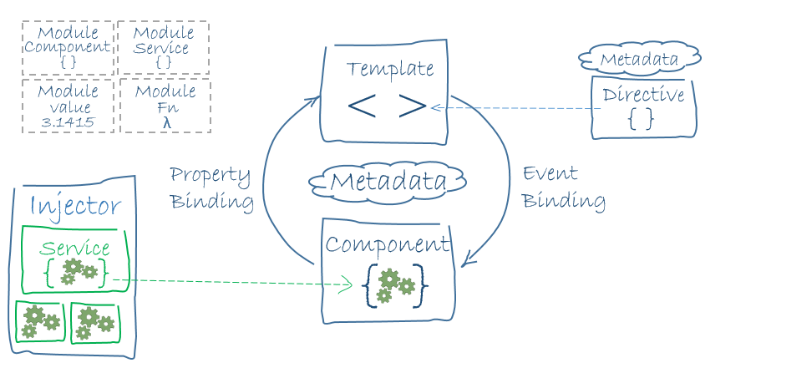
metadata that make the class  become a component.

Angular supports **data binding** for coordinating parts of a component with parts of a template. Data binding is added to the template HTML that tell Angular how to connect both sides

The directive is also a class, in directive, we will attach the metadata by the @Directive decorator.

Service is a class, we use service as a category contains any feature, function ,value or what we need to use in our application.

In Angular, we provide new components with the services they need by using dependency injection. Therefore, Angular can tell the components that the types of their constructor parameters may include the services they need.



4/ RESTfull webservice

Angular 2 is simply a front-end framework for building applications. It is not the right determinant for what backend you should use for your application. There are many ways to connect angular 2 to your backend server. RESTful Web Service which is essentially REST Architecture based Web Services is one of the architectural style that helps angular 2 and your backend server communicate with each other. In REST Architecture, everything is a resource. RESTful web services are light weight services so the developers usually use RESTful web services to make APIs for web-based applications.

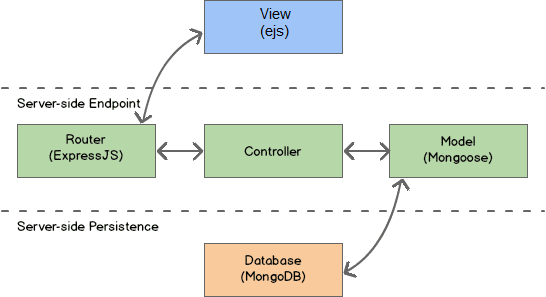
In some case, RESTful Web Service help us write a software application in various programming languages and we can run them on various platforms. For example, we can write a backend server in Java using RESTful web service and connect to fontend in Angular 2 using Typescript.

REST is a web standards based architecture which was first presented by Roy Fielding in 2000. The word ‘REST’ means REpresentational State Transfer. REST uses HTTP Protocol for data communication. It spins around resources where each component is a resource and a resource accessed by a typical interface utilizing HTTP standard methods.

RESTful Web Services are Web services based on REST Architecture. They use HTTP methods to implement the concept of REST architecture. URI is usually a service which a RESTful Web Service provides resource such as Text, JSON and XML.

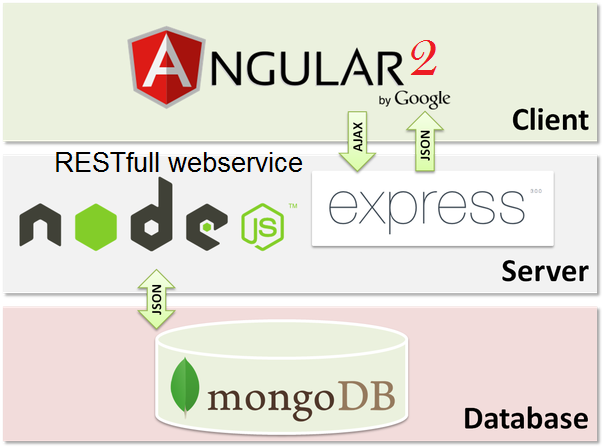
5/ Express framework + Nodejs

* Model use Mongoose to retrieve data or update to database
* Controller uses model as the datatype, retrieve data from model after mongoose get from database
* Controller sends request to model to update database
* Controller provides functions for router
* Router uses controller’s functions
* Router provides Http method to provides API for view
* Router can render the view (ejs)
* Router receive resquest from view and call controller’s functions to update database



6/ MEAN stack

* Nodejs and Express framework on server side
* Server updates and retrieve data from database (in Json format)
* Server communicate with client side via HTTP method (RESTfull web service)
* Server provides API for client side
* Angular 2 on client side get API (in Json) from RESTfull web service
* Angular 2 send http request to server via Ajax
* When user views a page: server retrieves data from mongodb => provides API (json) for client => display
* When user requests an update: client send http requests to client via ajax => server call RESTful Http method => update mongodb



II/ Software Requirement

1/ System Overview

* Hotel Bookings & Reservations System is a web application running on 2 servers
* 2 servers are running at the same time => each server doesn’t have to do a lot of job.
* The main architecture is using MEAN stack technology and J2EE with Spring MVC framework.
* MEAN stack technology => becomes an online single page application with high performance
* Nodejs and express framework => RESTFULL web service + Angular 2 => Dynamically loading + user tracking
* Java => becomes a cross-platform system runs well with all operating system.
* Spring MVC => most powerful java framework => flexible and loosely coupled web applications
* HTML5 + CSS3 + Bootstrap + AngularJS + Angular 2 => Friendly user interfaces

=> comfortable, easy to use.

* 42 primary feature and hundreds of small features.

2/ Feature

* There are many features that my system support for each role *(List functions of each role: guest, customer and admin)*

a/ Guests;

+ view introduction and gallery of the hotel

+ send reservation form, contact with administrators.

+ view, search the rooms or the items in the restaurant which they would like to see more details.

+ register an account to become a customer.

b/ Customer

+ do anything which the guests can do.

+ login to the system to book room or cancel it

+ rate the room, send feedback,

+ check profile

+ view transaction history.

+ With data collection feature, customers were tracked => the system can suggest the recommendation rooms for the customers.

c/ Administrator

+ login to the website and go to their dashboard to manage the hotel

+ check his profile, add, update and delete rooms or other services in the restaurant

+ receive the request of customers and reply them with several available email templates. + manage the users, view information and activity of users or ban them if they did something unacceptably.

+ thank to follow-users feature, administrator is able to see which page customers clicked, how long they stayed in each page, which keyword they used to search, which image they used to click on

+ view the chart with the statistics of visitor from country.

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Feature** | **User** | **Description** |
| 1 | Register | Guest, Admin |  |
| 2 | Login | Guest, Admin |  |
| 3 | Logout | Customer, Admin |  |
| 4 | View Rooms | Guest, Customer, Admin |  |
| 5 | View Restaurant | Guest, Customer, Admin |  |
| 6 | Search for Room | Guest, Customer, Admin |  |
| 7 | Search for Food, Drink | Guest, Customer, Admin |  |
| 8 | View gallery of hotel | Guest, Customer |  |
| 9 | View introduction of hotel | Guest, Customer |  |
| 10 | Filer rooms | Guest, Customer, Admin |  |
| 11 | Filer food or drink | Guest, Customer, Admin |  |
| 12 | Send contact | Guest, Customer |  |
| 13 | Send reservation form | Guest, Customer |  |
| 14 | Book room | Customer |  |
| 15 | Cancel room | Customer |  |
| 16 | View profile | Customer, Admin |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Feature** | **User** | **Description** |
| 17 | Edit profile | Customer, Admin |  |
| 18 | Change password | Customer, Admin |  |
| 19 | View activity | Customer |  |
| 20 | Send feedback | Customer |  |
| 21 | Dashboard management | Admin |  |
| 22 | Receive notification | Admin |  |
| 23 | Send message | Admin |  |
| 24 | View users | Admin |  |
| 25 | Manage users | Admin |  |
| 26 | Ban users | Admin |  |
| 27 | Add new room | Admin |  |
| 28 | Delete room | Admin |  |
| 29 | Update room | Admin |  |
| 30 | Add food or drink | Admin |  |
| 31 | Remove food or drink | Admin |  |
| 32 | Update food or drink | Admin |  |
| 33 | Update profile image | Admin |  |
| 34 | Follow users | Admin |  |
| 35 | Send feedback & rate hotel | Customer |  |
| 36 | Send feedback & rate room | Customer |  |
| 37 | View customer activity | Admin |  |
| 38 | View statistic of visit times | Admin |  |
| 39 | View recommendation room | Guest, Customer |  |
| 40 | View related room | Admin |  |
| 41 | View top of rooms | Guest, Customer, Admin |  |
| 42 | Email template | Admin |  |

3/ Use case:



* There are 3 actors using the system: guest, customer and administrator

4/ User story:

Write user story:

* As a guest, I can register a new account so that I can login to the system
* As a guest, I can view the rooms so that I can see the details of the rooms, watch the image of the rooms.
* As a guest, I can view the food or drink in the restaurant of the hotel so that I can see the details, watch the images of each item in the restaurant.
* As a guest, I can view introduction and gallery page so that I can see the information of the hotels and watch the image gallery of the hotel.
* As a guest, I can send contact to the administrator so that I can write what I want to communicate with him and wait for his response.
* As a guest, I can view the recommendation rooms so that I can see which room that the system automatically suggests me book.
* As a customer, I can login to the system or logout so that I can use more features.
* As a customer, I can edit my profile so that I can change my personal information.
* As a customer, I can book room so that when I come to the hotel, this room belongs to me,
* As a customer, I can send a feedback about a room or about the whole hotel services so that I can rate the star of service and comment or complaint my opinion.
* As a customer, I can view my activity so that I can see the transaction history, what I have done, what I interacted with the hotel.
* As an administrator, I can login to the system or logout so that I can use admin features.
* As an administrator, I can edit my profile so that I can change my personal information.
* As an administrator, I can manage the rooms so that I can view the rooms, add a new room, edit a room or delete it.
* As an administrator, I can manage the items in restaurant so that I can view the items, add a new item, edit an item or delete it.
* As an administrator, I can manage users so that I can view user information, view what they interacted with hotel or delete a user from database.
* As an administrator, I can view my messages and notifications which the guests or customers send to me so that I can interact with them and reply their message.
* As an administrator, I can follow user’s behavior so that I can see what they clicked, what they searched, what they did on the website.
* As an administrator I can view the visitor chart from country so that I can easily compare which is the most visited country, which is the less visited country and another.
* As an administrator I can view the page access chart based on all IP address or single IP address so that I can easily compare which is the most visited page, which is the less visited page and another.
* As an administrator, I can receive the message, the booking request, cancel room request and feedback of the customers so that I can view the information that they send to me and reply them by myself or using some available email templates

Kẻ thêm 1 số table use case quan trọng, ko quan trong đem xuống appendix

III/ Methodology

1/ All Technology used:

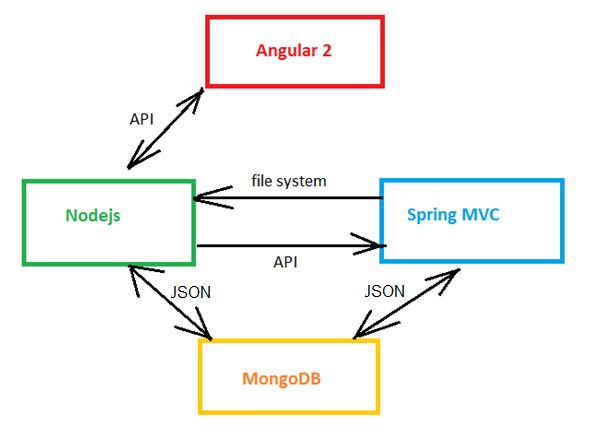
(list technology and introduce)

List technology used:

* Back end: Java web J2EE + Spring MVC framework, Node.js + Express framework
* Front end: HTML5, CSS3, Javascript, Jquery, Boostrap, AngularJS & Angular 2 framework
* Database: MongoDB, RoboMongo
* UML tool: Edraw
* IDE: VSCode, Eclipse, Netbeans
* Front-end design tool: Adobe Dreamweaver CS6
* Server: npm, tomcat, glassfish
* Code review and analysis: Sonar Lint
* Version Control: Git hub
* Project management: Trello

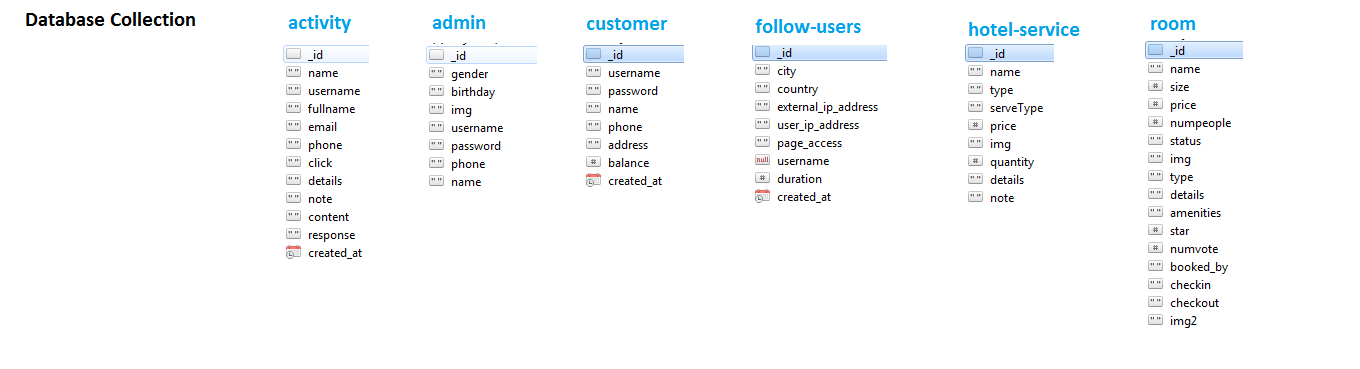
2/ System Architecture

* System includes 2 main architectures:
* MEAN for Customer page
* Spring MVC for Admin Page
* Communicate through API, mongodb and file system
* Draw background diagram first



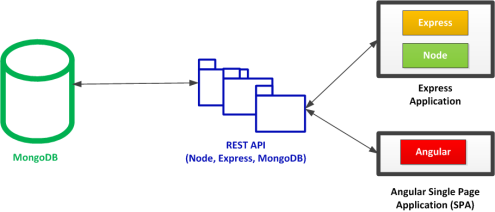
2.1/ MongoDB

6 collections

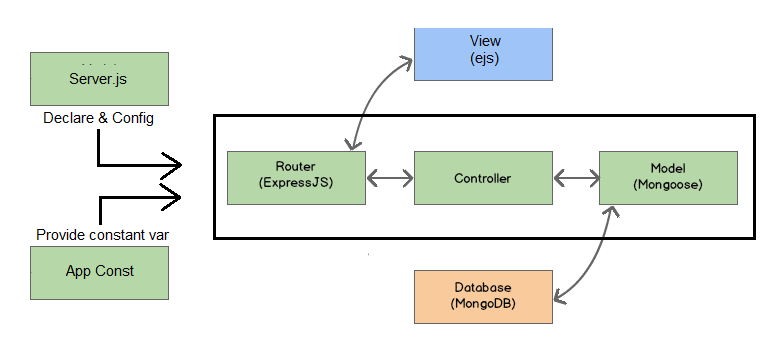


2.2/ MEAN stack technology:

* Mongodb, Express Framework, Angular 2 framework, Nodejs – for Customer & guest page
* Nodejs with Express framework build RESTful webservice provides API for Angular
* RESTful webservice GET data from mongodb and provides GET HTTP method for Angular to display the data
* Resquest from Angular via AJAX => PUT POST DELETE HTTP method => RESTAPI => update mongodb

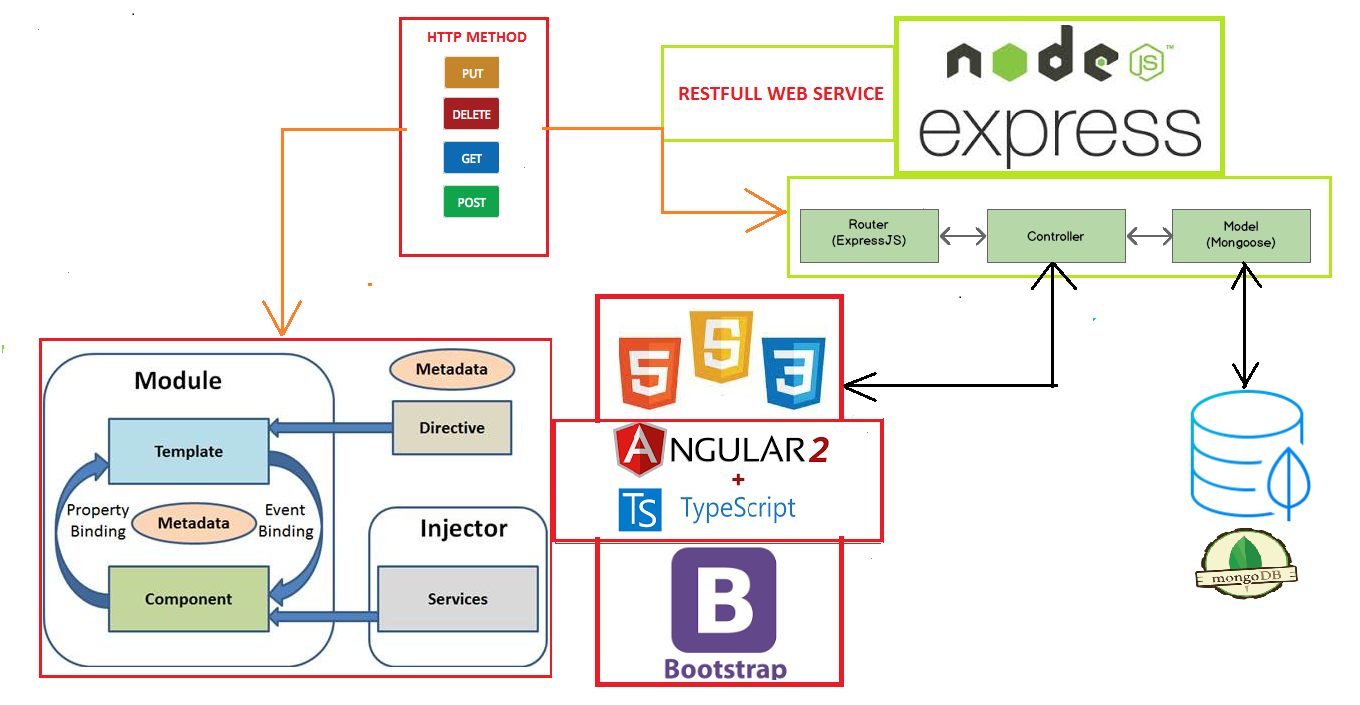


* In Nodejs & Express framework:
* The server.js starts a server and listens on port 3000, contains configuration & declaration of the server
* AppConst provides contants variable for the whole express system
* Model uses Mongoose connect database update or retrieve data
* Controller use mongoose to provide functions for router
* Router communicate with the view



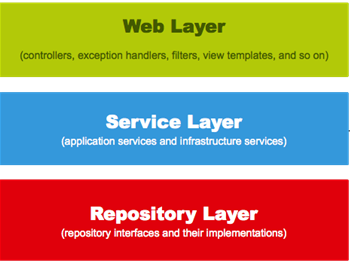
* Embed Boostrap, HTML, CSS, javascript, Jquery, Angular 2 into HTML file (template)
* The components control template by metadata => provide data binding
* Services provide function for components
* module control all the components, templates, metadata, directives, services
* Angular 2 api services interact with RESTfull Webservice
* Request from user => angular 2 routing => determine which page (template+component)

=> display

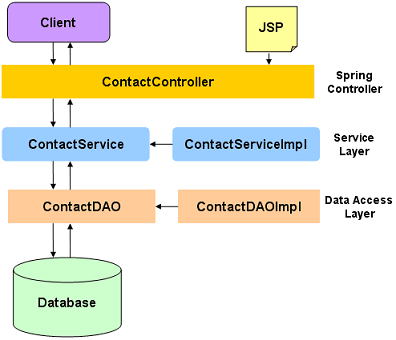


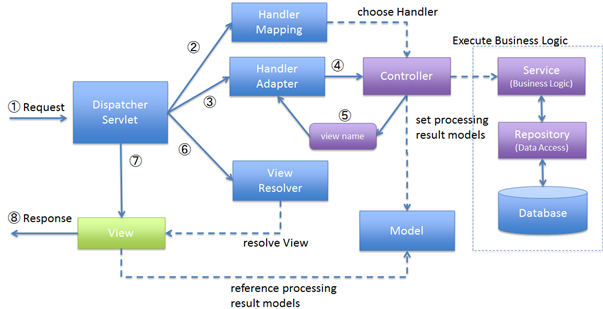
2.3/ Spring MVC:

* Follow MVC architecture + 3 layers architecture

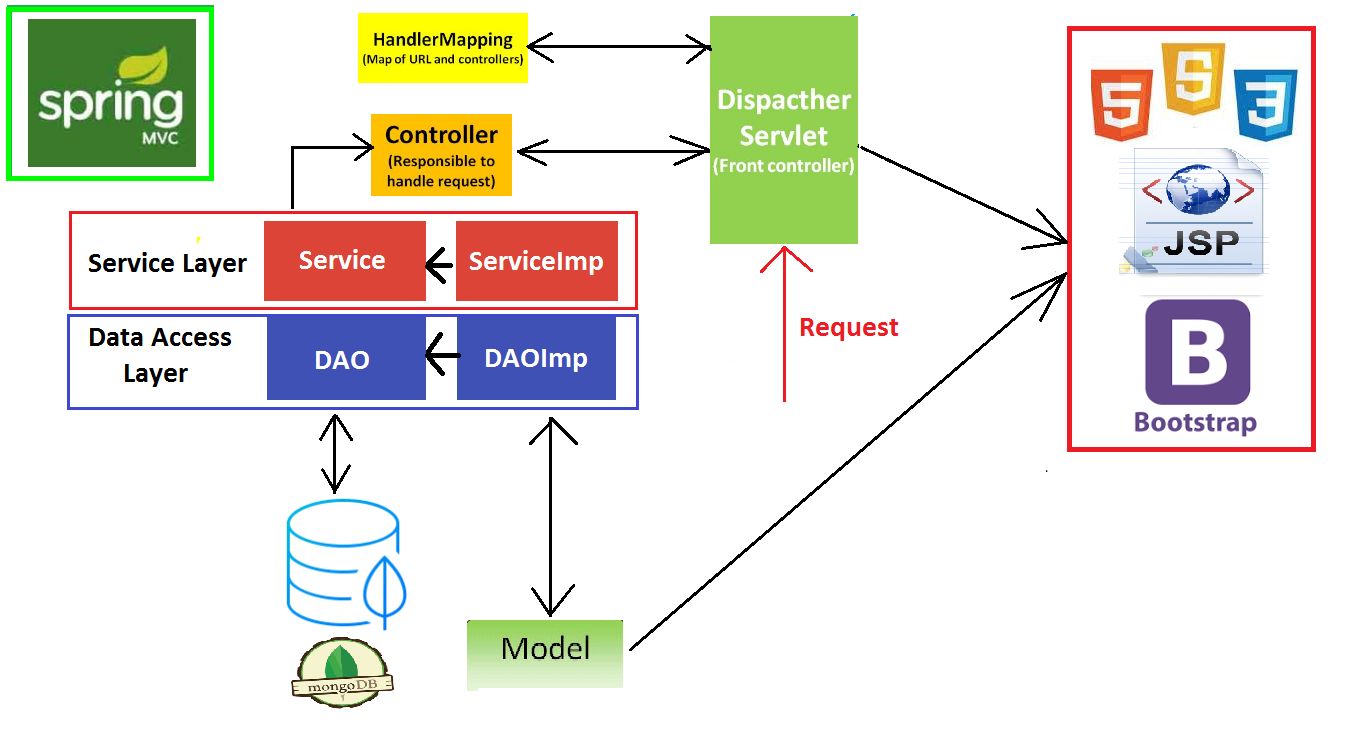


* Spring MVC + 3 layers architecture





* With DispatcherServlet is Front Controller
* DispatcherServlet receives the request.
* DispatcherServlet dispatches the task of selecting an appropriate controller to HandlerMapping. HandlerMapping selects the controller which is mapped to the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet.
* DispatcherServlet dispatches the task of executing of business logic of Controller to HandlerAdapter.
* HandlerAdapter calls the business logic process of Controller.
* Controller executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter.
* DispatcherServlet dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolverreturns the View mapped to View name.
* DispatcherServlet dispatches the rendering process to returned View.
* View renders Model data and returns the response.
* Embed Boostrap, HTML, CSS, javascript, Jquery, Angular into jsp
* Request -> FrontController -> handle mapping -> read mapping configuration
* FrontController -> Controller -> Service -> ServiceImp -> DAO -> DAOImp -> connect dababase -> update database or retrieve data => Controller use retrieved data -> FrontController -> display
* Controller -> Service -> SericeImpl -> DAO -> DAOImpl -> update to database
* Controller -> Service -> ServiceImp -> API -> APIImp -> interact with API from nodejs server -> api
* MainControler -> RequestMapping -> model map -> display
* RESTController -> api -> angular get api by HTTP methods
* Model: represents the data, provide data type



2.4/ Diagram

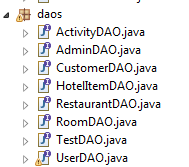
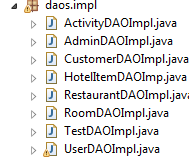
Class diagram – chia rõ class diagram cho frontend và backend

Activity diagram

Database

2.5/ Details

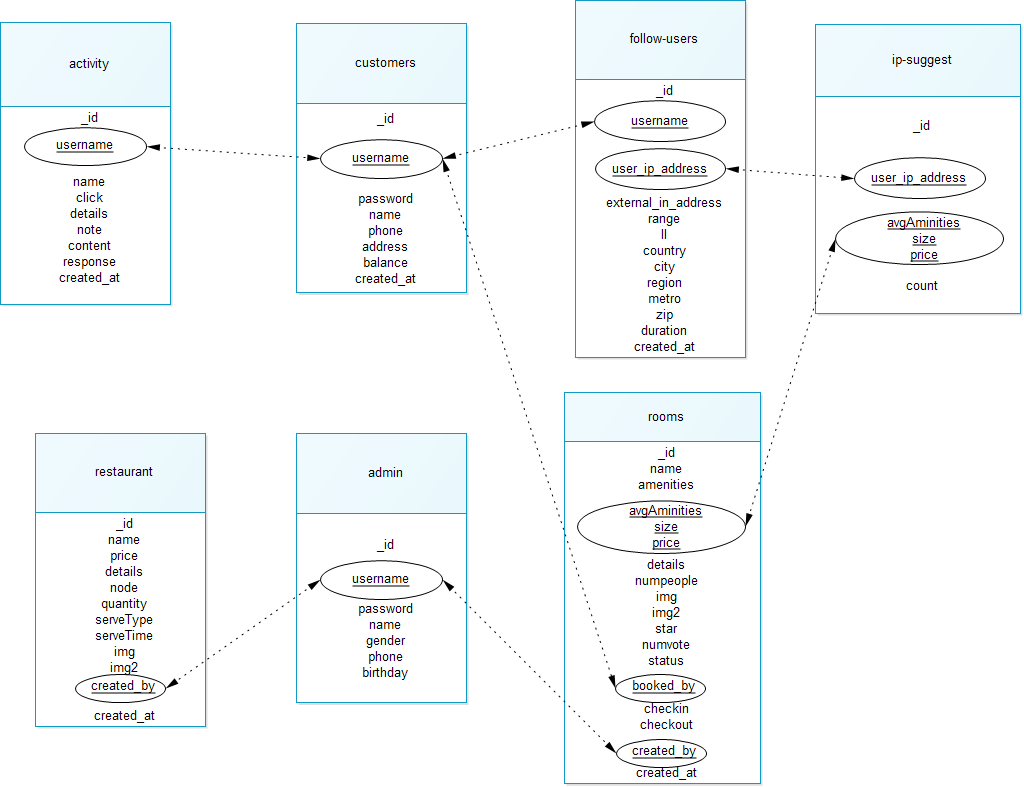
*2.5.1/* Spring MVC

* Maven project – file pom.xml contains dependencies => auto download library needed
* Webapp folder contains resources folder, WEB-INF folder and redirect.jsp file
* resources folder contains all resources for Admin page client side (image, css, js, boostrap, jquery, angular)
* describe how angular work? How embed image, css, js, boostrap, jquery, angular in client side?
* Almost written in java & jsp
* WEB-INF folder is the place storing the view (jspf & jsp files) and configuration files
* describe jspf & jsp. How they work?
* (applicationContext.xml, dispatcher-servlet.xml, web.xml)
* Java Resources include src/main/java, src/main/resources, src/test/java
* Read properties from src/main/resources
* Three Layers Architecture - DAO – Service - Controller
* Using 3 layers architecture => what? Why 3 layers? Separate database layers from controllers + easy to maintain
* MongoDBConnector connect to database mongodb
* Model: represents the data, provide data type
* APIImpl implements API provides methods for ServicesImpl
* API contains method connect with RESTfull Nodejs Server and get the api
* DAOsImpl use MongoDBConnector.java to connect mongodb and implements DAOs provides some methods for ServicesImpl
* ServicesImpl use DAO implements Services provide methods for Controllers
* Controllerss use Services
* Statics package includes AppConst.java and static providers
* AppConst.java contains all Constant variable, array => no hard code
* Statics providers provide some classes contains many static methods (calculate & format date time, send Email, File Upload, edit image, round, StringUtils) for the whole application
* MainController & RESTfullController. How they work?
* How RESTfullController interact with Angular Client side?
* Describe mapping, how Webapp interact with java resources, in MVC architecture
* List all important files and describe => give image
* Ex:
* 

*2.5.2/* MEAN

* package.json file contains all dependencies => auto download lib
* server.js declares some configuration (router, cookie, app Constanst) and host the server
* app-const.js file contains all constant variables provide for the whole server
* routes determine which page (template+component) is considering and will redirect.
* Nodejs model mongoose connect database => update or retrieve data
* Controller use model, send update request to model or get data from mongoose
* Controller provides function for routers to provide api
* Nodejs + Express framework => RESTfull Web service => provides HTTP methods (GET, PUT POST, DELETE) => api => json
* Embed Boostrap, HTML, CSS, javascript, Jquery, Angular 2 into HTML file (template controlled by component (typescript files)
* The components control template by metadata => provide data binding
* Services are injected to component and provide function for components
* All the components, templates, metadata, directives, services are declared in a module and controlled by it.
* Angular 2 api services interact with RESTfull Webservice
* services receive request from component or return data to component
* Nodejs + Express => RESTfull interact with Angular 2 by HTTP Methods
* List all important files and describe => give image

*2.5.3/ Mongodb*



- No entity relationship => don’t need to define the structure

- stores data in JSON-like documents => easily work with javascript, Angular, nodejs

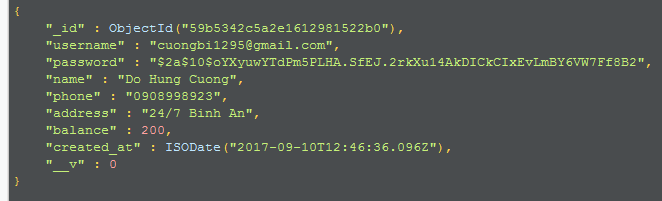
- redundant data but high performance (Related information is stored together for fast query) => Optimize Query Performance

- difficulty in update, when the customer changes his information => have to update all collection that related to this customer or when delete a room …

- However, delete a room or change user information is not usually => reduce update speed to improve query performance is ok

* Show all collections and describe => give image

Ex:



2.6/ Implementation:

chụp hình 1 vài đoạn code quan trọng

chụp hình chương trình chạy

V/ Experiment and Result

1/ Experiments

* System run well on window
* Test on Linux
* Responsive website => test mobile UI
* Encrypt password
* Recommendation room
* Chart

2/ Evaluation

* Online single page application with high performance => speed? Compare with what?
* Dynamically loading? Why? prove
* Cross-platform system runs well with all operating system. => prove? Run well on window + Linux, + mobile?
* Friendly user interfaces, easy to use => show image
* Supports almost features for hotel bookings & reservations management. => show list features
* Ability to track user’s behavior => show chart, image
* Give examples, image to prove

VI/ Conclusion

* Hotel business is a highly profitable industry but requires huge investment as well as having to meet the customer's demand.
* Management system is really important
* Good management system brings higher profit
* The key is to pleasure the customers
* After applied MEAN stack and Spring MVC to implements Hotel Booking system, I recognize that
* Friendly user interface
* High performance
* Dynamically loading
* Tracking customer’s behavior
* Improve system day by day
* In the future, I will add more features, apply machine learning with enough data collection
* After thesis => learn a lot of new things
* Learn a lot of technologies: AngularJS, Angular 2, MongoDB, Spring MVC, Nodejs, Express Framework…
* Experience in building single page application
* Working with a lot of frameworks
* Ability to learn new technology
* Improve myself in the future for working in professional environment

VII/ Appendix

1/ Use case bổ sung

Kẻ thêm nhưng bảng use case bổ sung cho phần software requirement, trên phần software requirement nói “để hiểu rõ về … xem phần appendix 1.1, 1.2…

2/ Sequence diagram bổ sung

* Draw some sequence diagram that represent for the whole system

3/ Test Case bổ sung

* Test case table

4 User Manual

* How to use + image

VIII/ Reference

<https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html>

http://terasolunaorg.github.io/guideline/1.0.1.RELEASE/en/Overview/SpringMVCOverview.html

http://viralpatel.net/blogs/spring3-mvc-hibernate-maven-tutorial-eclipse-example/