# HOA Programmer's Guide

# Table of Contents

# 1.Project Overview

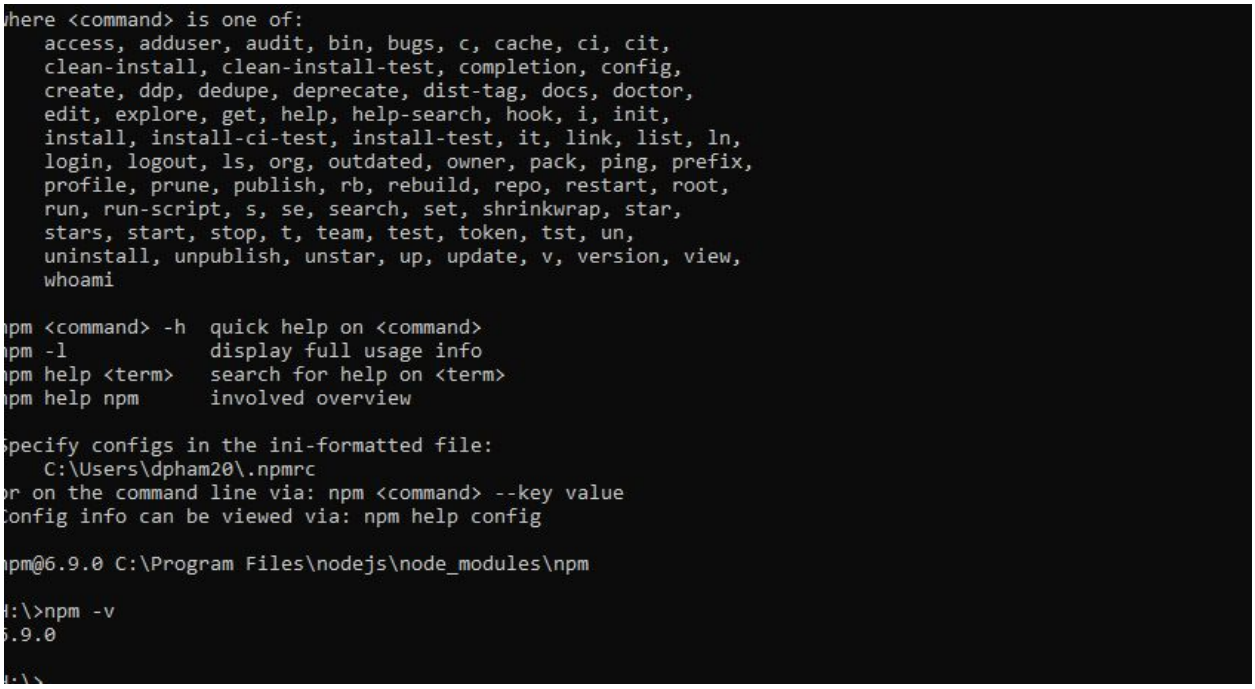
This project is a full-stack implementation of a website of a homeowners association in Columbus, Ohio. The application allows for homeowners to log in, view documents related to the association, send in tickets into a ticketing system, and for admins to view and handle open tickets through an admin portal. The webpages are data-driven and dynamic. The view is and should be responsive based on the device resolution.

The backend using NGrok, Spring Boot, and Java. It is built in a microservice design.

# 2.Software Installation / Running Angular Application(Frontend)

To be able to run and work on the front end portion of the web application you need to install and change some system settings. Angular documentation for reference: https://angular.io/.

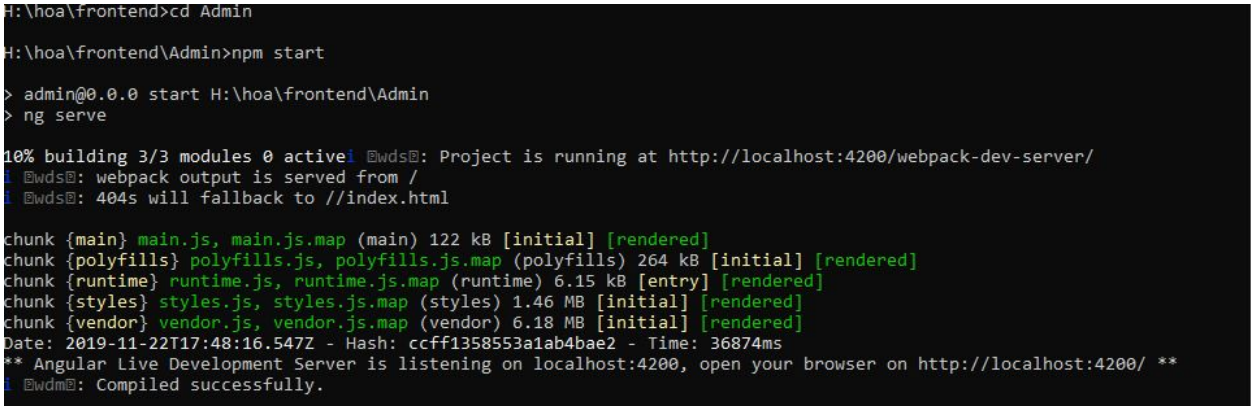
1. Install a text editor/IDE.
    a. I recommend installing Visual Studio Code as something to handle programming the frontend. There are a lot of neat extensions that make your life easier.
2. Install Node.Js
    a. Node is a package manager and will be used to install the Angular CLI onto your machine.
    b. You can go to https://nodejs.org/en/download/ and download the version that is compatible with your machine.
        i. This downloads Node Package Manager(npm) which allows for the installation of Angular CLI.
    c. Try running “npm” commands in your command prompt such as “npm -v”.
        i. Make sure npm is recognized as a valid command.




3. Installing Angular CLI
    a. To install Angular CLI onto your machine go to command prompt and type: "**npm install -g @angular/cli**"
    b. To test if it has successfully installed you can type in: "ng version"

**Here is a guide to all that was mentioned for installing Node.js and Angular.Cli:**
https://www.zeolearn.com/magazine/setup-angular-windows

4. Running the Angular Project
    a. To run the angular program please open the Admin folder located at hoa/frontend/Admin/ in a text editor.
    b. From the text editor or command prompt run:
        i. "**npm install**"
            1. Note that this only has to be run once per project.
            2. This creates the node_modules folder and populates it with the libraries and modules used in the application.
        ii. "**npm start**"
            1. Compiles and runs the Angular application on a given port.
            2. You can now open up a browser to view the project on the given port such as "http://localhost:4200/" is what the project is running on in this screenshot.




# 3. Source Control and Repository

The team uses Github as its source control.
Github Link: https://github.com/Hondohondo/hoa

# 4. Backend

## 1. Installation and Initialization

Please refer to Sections II and III in the User Manual

## 2. Back-End GitHub home:

https://github.com/Hondohondo/hoa/tree/master/backend

## 3. Remarks

This project follows a loose microservice architecture, as shown in the graph. There are a few key design patterns at play throughout the program. The main pattern for the backend is the Observer-Observable which passes a new request down to each feature. Then, each feature has a factory which is ultimately responsible for generating the appropriate response back to the client (which is the ApiController)

# 4. How to implement a new feature:

# 5. Frontend

Physical path to the angular project: hoa/frontend/Admin/

# 1. Styling

The application is styled through Cascading Style Sheets(CSS).
Here are the style sheets that are in use (they are listed in order of hierarchy):
**Style.css :** https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/styles.css
**App.component.css:**
https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/app.component.css
**Component style sheets:**
These can be found in the individual component folders.
Example path:
https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/home/home.component.css
Styling for style.css will be overridden by code from app.component.css and so on. If you need to replace a styling rule that is specific to a component change that component's CSS file. If you want to change an overall style that affects every page you can either make the change in the app.component.css or styles.css files.

# 2. Frontend Functionality

**API / HTTP Request Handlers**
In Angular, any file with ".service" in their extension is a service file. For this project, there are multiple service files that mainly deal with handling HTTP requests and messages from the API.

**hoa.serivce.ts**
https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/hoa.service.ts
This class handles the HTTP requests for the HoA ticketing system.
**Constants**

1. `const httpOptions`
   A custom header for the HTTP request being sent back.

**Constructor**

1. `constructor(private http: HttpClient) { }`

Http - Reference to Angular HttpClient. Used to make requests to API.
**Variables**

1. `public getAllTicketURL: string`
   Variable holding the URL connecting to the endpoint that returns all Ticket objects from API.
2. `public getTicketByIdURL: string`
   Holds URL connecting to the endpoint for getting a Ticket object by id.
3. `public postTicketURL: string`

Holds URL connecting to the endpoint for posting a Ticket to the API.

4. `public updateTicketURL: string`
   Holds URL connecting to the endpoint for updating a Ticket, same as Post.
5. `public deleteTicketURL: string`
   Endpoint URL for deleting a Ticket.

**Methods**

1. `getAllTickets(): Observable<Ticket>`
   Grabs all Ticket objects from API / Database.
2. `getTicketById(id: number): Observable<Ticket>`
   Grabs a specific ticket from API based on id parameter.
3. `addTicket(ticket: TicketPost): Observable<TicketPost>`
   This method sends back a TicketPost object, ticket, to the API to add a new ticket into the database.
4. `updateTicket(ticket: Ticket): Observable<Ticket>`
   This method is used to update an existing Ticket in the database by sending back another Ticket object to take its place.
5. `deleteTicket(id: number): Observable<{}>`
   Deletes the Ticket from the server based on id parameter. Sends back a DELETE HTTP request.

**alert.service.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/_auth_services/alert.service.ts

This service class handles showing pop-up alerts based on errors messages from the API for the login page.

**Constructor**

```
constructor(priva
te router:
Router) {
                        // clear alert messages on route
change unless 'keepAfterRouteChange' flag
is true
                        this.router.events.subscribe(event
=> {
                            if (event instanceof
NavigationStart) {
```

```
                    if
(this.keepAfterRouteChange) {
                        // only keep for a
single route change

this.keepAfterRouteChange = false;
                    } else {
                        // clear alert message
                        this.clear();
                    }
                }
            });
        }
```

Router - Reference to Router library. Used to route users to different components or pages. The constructor clears alert messages on route change unless 'keepAfterRouteChange' flag is true. The router is a reference to the routing library to navigate the user once an alert appears.

**Variables**

1. `private subject = new Subject<any>();`
   A data structure filled with alert messages.
2. `private keepAfterRouteChange = false;`
   Keeps track if the alert should reroute user based on the result of the alert message and type of the message.

**Methods**

1. `getAlert(): Observable<any>`
   Returns an observable that represents the alert.
2. `success(message: string, keepAfterRouteChange = false)`
   States if a return is successful, and clears the error message.
3. `error(message: string, keepAfterRouteChange = false)`
   States if an alert has an error.
4. `clear()`
   Clears subject by calling subject.next() without parameters.

**authentication.service.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/_auth_services/authentication.service.ts

This service class handles sending back user login information and handling the response from the API for logging in.

**Constructor**

```
constructor(pri
vate http:
HttpClient) {
                    this.currentUserSubject = new
BehaviorSubject<Member>(JSON.parse(localStora
ge.getItem('currentUser')));
                    this.currentUser =
this.currentUserSubject.asObservable();
    }
```

Http - Reference to Angular's HttpClient library. Used to make requests to API.
Sets the current user to the person who is logged on for all other components to keep track of. Also has http which is a reference to the HttpClient library.

**Variables**

1. `private currentUserSubject: BehaviorSubject<Member>;`
   Observable to the current user reference.
2. `public currentUser: Observable<Member>;`
   The actual reference to the current user for this class to keep note of.

**Method**

1. `public get currentUserValue(): Member`
   Grabs the current user value from currentUserSubject.
2. `login(username: string, password: string)`
   Makes an HTTP request to the API sending back a username and password to check credentials before allowing the user to be logged in. Saves user to local storage if the credentials match.
3. `logout()`
   Removes user from local storage and sets currentUserSubject to null.

**user.service.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/_auth_services/user.service.ts

This service class handle grabbing information for a specific user and all users. For this application we use this class to keep track of who is currently logged in.

**Constructor**

```
constructor(private http: HttpClient) { }
```

Http - Reference to Angulars HttpClient library. Used to make requests to API.

**Method**

1. `getById(id: number)`
   Returns a Member object by their given Id.
2. `getAll()`
   Returns all Member Objects from API.

**Components**

Components are the individual "pages" of the Angular application. Each has their own functionality, typescript file.

**Login.component.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/login/login.component.ts

This file handles the logic behind the login screen / component.

**Constructor**

```
constructor(
            private formBuilder: FormBuilder,
            private route: ActivatedRoute,
            private router: Router,
            private authenticationService:
            AuthenticationService
                )
```

formBuilder - Reference to Formbuilder module. Used to group the input fields for login component into one formgroup.
Route - Reference to ActivatedRoute module. Used to keep a reference to the current page and to navigate to different components.
Router - Reference to Angular Router library. Used to send the user to a different page based on login attempt response from API.
authenticationSerivice - Reference to service class for authentication. Allows login component to spend HTTP requests through the service class.

**Variables**

1. `loginForm: FormGroup;`
   The form group reference that handles user inputs into the login input boxes.
2. `loading = false; //checks to see if page is loading.`
   Checks to see if the page is currently loading.
3. `submitted = false; //checks to see if loginForm has been submitted.`
   Checks to see if the loginForm has been submitted.
4. `returnUrl: string; //used to route user after login action is completed.`
   Reference to the route after the login action has been completed.
5. `error = ''; //string to represent error message.`
   String that represents an error message.

**Method**

1. `ngOnInit()`
   Builds a loginForm using two input elements of username and password.

Get return url from route parameters or default to '/'.

2. `get f()`
   This method allows for convenience getter for easy access to form fields.
3. `onSubmit()`
   This method calls the login method from the authentication service using the user input for password and username.

**home.component.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/home/home.component.ts

This component handles all functionalities of the home page.

**Variables**

1. `loading = false;`
   Checks to see if the page is currently loading.
2. `currentUser: Member;`
   Reference to the current user of the website. Keeps track of the user.
3. `userFromApi: Member;`
   Reference to who they API thinks the current user is.

**Constructor**

```
constructor(private userSerive:
UserService, private
authenticationService:
AuthenticationService) {
```

```
        this.currentUser =
    this.authenticationServ
    ice.currentUserValue;
      }
```

The constructor sets current user equal to the currentUserValue from the authenticationService class reference. The currentUserValue is the API's return of what the current user logged in is.

**Methods**

1. `ngOnInit()`
   When the page is initialized, it sets loading to true meaning that the page is loaded.

**Ticket.component.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/ticket/ticket.component.ts

This file holds all functionality to the Ticket component which handles the ticket form for users to send back tickets to the API.

**Variables**

1. `intakeForm: FormGroup; //Represents the FormGroup used to send a ticket back to the API.`
   Variable reference to the intake form group that is used to send a Ticket Object back to the API.
2. `submitted = false; //checks to see if intakeform has been submitted.`
   Boolean variable that checks to see if the intakeForm form group has been submitted.

**Constructor**

```
constructor(public HOAService : HOAService, private fb:
FormBuilder)
```

HOAService - Reference to HOAService class to be able to send requests to API regarding the Ticketing system.

Fb - Reference to the FormBuilder library. Used to group and create the intake form for users to be able to send a Ticket to the API.

The constructor also builds the intakeForm form group using fb (reference to FormBuilder) in the constructor.

**Methods**

1. `ngOnInit()`
   Empty method.
2. `get f()`
   Returns an object reference to the form group, intakeForm.
3. `addNewTicket()`

This method calls the addTicket() from HOAService with an input of the intakeForm values. This sends the values of the intakeForm back to the API for the API to add those values as a new ticket to the database. The method then resets all data fields in the intakeForm and an alert will appear letting the user know that they have successfully sent in a ticket.

**Admin.component.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/admin/admin.component.ts

This typescript file holds the code for the logic behind the admin component/page.

**Variables**

1. `tickets: Ticket[]; //Holds all ticket objects for API.`
   Holds all Ticket objects that are sent from the API.
2. `modalForm: FormGroup; //modal intake formgroup`
   Reference to the FormGroup that appears when a user clicks on a specific ticket.
3. `ticketData: Ticket; //Ticket object that will represent a more informative version of a selected ticket.`
   The variable that grabs a more detailed look of a selected Ticket. Used to show the details of a specific ticket in a modal.
4. `ticketUpdate: Ticket; //Ticket object that will be sent to API to update a ticket.`
   Ticket object that is sent to the API when a ticket is updated.
5. `loading = false;`
   Used to check if the page is loaded on initialization.
6. `users: Member[] = []; //Holds information for the current user.`
   The variable that is used to hold information for the current user.
7. `modalRef: NgbModalRef;`
   The reference to the modal that appears when a user clicks on a specific ticket.
8. `headers = ["ticketId", "createdDate", "name", "subject", "isActive"];`
   Variable used to create the headings of the table created on the Admin page.

**Constructor**

```
constructor(public HOAService:
HOAService, private modalService:
NgbModal, private fb: FormBuilder,
```

```
private userService:
UserService, private cd:
ChangeDetectorRef)
```

HOAService - Service to access Ticket system requests to API
modalService - Object that handles events and actions of the popup modals.
Fb - FormBuilder reference that creates the modal intake form group to send updates and delete HTTP requests.
userService - Service object that gives access to user service HTTP requests.
The constructor creates the modalForm which is a form group for the modal.

**Methods**

1. **`ngOnInit()`**
   Sets loading to true and makes a call to the API to grab all active tickets to populate the table.
2. **`getAllTickets()`**
   This method makes a call to the API to get all active tickets from the database. Populates the variable, tickets with all of the Ticket objects returned.
3. **`async getSelectedTicket(id: number)`**
   This method returns the specific information about a ticket by the ticket's id. ticketData is set to the Ticket object returned by the API.
4. **`async getTicketInfo(content, ticket: Ticket)`**
   Content - represents the element (modal)
   Ticket - represents the ticket that is clicked on. Uses this ticket's id to grab more information.
   Method used to set all fields of the modalForm based on the attributes of ticketData (the selected ticket by id). Also opens the modal.
5. **`open(content)`**
   Content - Represents, a reference to the element modal.
   Used to open the modal.
6. **`close(content)`**
   Content - Represents, a reference to the element modal.
   Used to close the modal.

7. **`sendUpdate()`**
   This method sends back a ticket object to update the specifically selected ticket. It resets input fields in the modal, closes the modal after the data is sent, and sends an alert to the user when a ticket is sent.
8. **`sendDelete()`**

This method sends back a ticket id for the selected ticket. This ticket will be deleted. Closes the modal window when delete request is sent.

**Forum.component.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/forum/forum.component.ts

This component was supposed to be used for the forum page that the client wanted. This page was never touched so there is no functionality in the typescript file.

**App.component.ts**

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/app.component.ts

The parent component of the angular project. In this application, it houses the navigation bar and header for all of the other pages.

**Variables**

```
currentUser: Member; //Holds the Member object that is currently logged in.
```

Holds information for the member object that is currently logged in.

**Constructor**

```
 construc
 tor(
                private router: Router,
                private authenticationService:
            AuthenticationService
             )
```

Router - Router object that is used to navigate to different components/pages.
authenticationService - AuthenticationService reference that is used to call a quest to the API that deals with authentication.
The constructor grabs the current user information from the API.

**Methods**

```
get isAdmin()
```

Checks to see if the current user is an Admin under their role attribute.

```
logout()
```

A function used to log a user out and then navigates the user back to the log in screen.

**Model Classes**

Classes that handle representations of objects used in the application.

https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/Ticket.ts

Ticket.ts
https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/TicketPost.ts
TicketPost.ts
https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/_auth_models/Member.ts
Member.ts
https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/_auth_models/Role.ts
Role.ts

**Routing File**

Handles routing between pages and sets keys used to connect to other components.
https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/app/app-routing.module.ts
App-routing.module.ts

**Variables**

```
const routes: Routes
```

It contains a reference for all component pathnames for navigation purposes.

**Base HTML File**

The index.html file is the base HTML file and is not an Angular file. It calls a tag that pulls in the Angular components into the body.
https://github.com/Hondohondo/hoa/blob/master/frontend/Admin/src/index.html
Index.html

The <app-root></app-root> tag in the body of the index is where the angular application lives.

# 3. Adding New Content

**Adding a new component:**

To add a new component to the angular project used the terminal on your machine or text editor/IDE and type in “ng generate component <name of your component>”.
This should create a new folder that contains a HTML file, component.ts, and a .css file for your component.

```
ng generate component hero-detail
```

**Adding a new component path:**

To add a new path for navigating to different components you have to alter the routes variable.

```
{path: 'login', component:LoginComponent},
  {path: '**', redirectTo:''}
```

Add a comma at the end of the last path in the data structure of routes and then type in:
{path: “name of path”, component: “component class name”}
Make sure that you have a reference to the component in this typescript file before adding a route.

```
import { ForumComponent } from './forum/forum.component';
import { AdminComponent } from './admin/admin.component';
import { LoginComponent } from './login/login.component';
```

# 4. Troubleshooting

| Issue | Issue Description | Fix |
|---|---|---|
| 404 Error | When opening the URL for the angular application leads to a 404 page not found error. | 1. Run **npm start** in folder directory or IDE/Text Editor that you are using. You have to compile the project first before running it.<br>2. You might have typed in the URL wrong. Check what port your angular application is listening on by looking at the end of the compile statement of npm start. Example: ** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ ** |
| | | |
| | | |
| | | |