

# Virginia City Players Seat Planner

*ESOF 423 Spring 2021: Group 2*

Caleb Hayes

Sam Luther

Nicholas Zetterberg

## **Table of Contents**

<b>1</b>	<b>Program - VCP Seat Planner</b>	<b>3</b>
<b>2</b>	<b>Teamwork</b>	<b>4</b>
	Team Member 1:	4
	Team Member 2:	5
	Team Member 3:	6
<b>3</b>	<b>Design Pattern</b>	<b>7</b>
<b>4</b>	<b>Technical Documents</b>	<b>9</b>
	User Documentation	9
	Site Visual Guide - Step by step	11
	Developer Documentation	15
<b>5</b>	<b>UML</b>	<b>17</b>
<b>6</b>	<b>Design Trade Offs</b>	<b>17</b>
<b>7</b>	<b>Software Development Life Cycle Model</b>	<b>18</b>
<b>8</b>	<b>Appendix</b>	<b>20</b>

# 1 Program - [VCP Seat Planner](#)

The guidelines for this web-app were outlined by the initial project description as follows:

An application is needed to (1) automate the process of tracking available seating for a performance, and (2) format nightly seating data for print on demand tickets. This application requires:

- Database of shows, seating, and guest information
- UI interaction with database and formatted ticket data to print

Using this as our basis we developed a program that completes all of these specifications and more. Our web-app allows a user to create show entries and then assign tickets to a seat plan that is unique to that show. It also allows users to print individual tickets, or whole tickets for an entire show. A user may add, delete, or edit tickets using our app as well. Each ticket entry has fields for name, email, phone number, number of tickets, and for accommodations. Each of these features is linked together to form a conducive and efficient product that completes the goals that were assigned to us.

Our site was built using Google Firebase as our initial framework. Firebase handled all of our database functions, as well as hosting. Our database was set up such that each show entry created its own unique entry within a “shows collection” while all of our tickets were stored in a separate “ticket collection”. When a new ticket is created it is assigned a variable that is the ID of the show it is connected to. This is how we query our database for tickets from a specific show, by finding all tickets with that ID as its showID variable. The one other standout feature of our app is our interactable seat planner. This was done using an SVG file that interacts with a seatAssign function that then creates a string entry for a ticket. Using the SVG provided to us allowed us to make it interactable and actually show updates when seats are assigned. Further information can be found at our GitHub or in the developer documentation on our site.

Overall we achieved the assigned specifications of our project and even created some additional functionality. Project can be explored on our [GitHub](#).

## 2 Teamwork

### Team Member 1:

My responsibilities for our group's VCP software was the seat planner. This is the functionality that allows a user to select specific seats for a customer. Once the user creates a new ticket they will be brought to a screen where they can select which seat the customer is buying. Once the seat is selected it will no longer be available because the seat is full. This function lets the user map out the theater in a simple, no hassle system. This system uses mostly html with a few lines of JavaScript. The html is how I attached buttons to the seat planner based on the SVG for the picture. The JavaScript connects the button presses to the database.

Below are the burn down chart tasks that have been assigned to me. In total I have been assigned 26 hours of burn down tasks. This has been a semi-accurate representation of actual time spent working on the project. I definitely spent more time struggling through issues than the burndown gives me credit for. However, this additional time spent is not out of the ordinary when developing complex applications. I felt like everything I was tasked with was manageable and achievable in the short amount of time we were allotted.

There was a large learning curve when attempting to teach myself javascript and the firebase implementation. Once I understood how to code and format the tasks I had, things got a lot easier. I spent a lot of time understanding how to push and pull data from the database without breaking or overwriting anything. It was manageable, but difficult. The most difficult thing was learning how to interact with an svg to properly attach buttons to different parts of the picture. This was a hard conceptual problem rather than an execution problem.

2	Write to Database	
4	Seat Planner (Design)	
4	Start Prototype	
4	Functional Prototype for seat planning	
3	Start linking seat planner to database functionality	
6	Link seat planner to database	
2	Update Documentation	

1	Fix odd case buttons on the seat planner (greyed out seats)	
---	---	--

### Team Member 2:

For this project my responsibilities have largely been in the testing department as well as small tasks scattered around things like authentication, some site security and print formatting. Primarily though I have put work into managing our continuous integration system via github and finding and fixing bugs that are merged into master. For the testing Framework I opted for a JS testing library called Jasmine which was somewhat easy to implement into the CI environment. I also worked with security on the site handling the authentication process as well as site-wide auth checkers. Finally I worked on the ability to print off tickets and used a js library called PrintJS which I implemented using programmatically generated html which was then converted to pdf.

All totalled I committed to approximately 29 hours of work assuming each story point is an hour. However, many of these tasks were shared so it may be more accurately a total of 22-26 hours. I think overall this is probably quite close to the actual time spent on the project if not a little under. As there were many other tasks that I worked on that were unlisted in our burndown chart I would say that any tasks that I finished early were probably made up for elsewhere. There was also the addition of scrum meetings, coordinating and site maintenance. So if I would approximate all time spent with the project in any way it would be nearer 40 hours all in all. But under 30 working with the code.

2	Testing Framework	
2	ER Diagram for database layout	
4	Write Test Functions for existing functionality	
2	Testing	
3	Start linking seat planner to database functionality	
2	Create Print Buttons on page	

1	Continued Testing	
1	Finish auth with user access list	
5	Count Tickets	
2	Finish Ticketing	
3	Github Issues	
2	Complete Dev Documentation	

### Team Member 3:

My main focus on the project so far has been all of the database functions and all the ticket handling on the site. I began by creating the read and write functions for the database to utilize this in our ticket creation for when a user creates a new ticket. Then the ticket is reflected on the front end so that the user can then edit it, and also delete it. The tickets are also linked to a show by the use of firestore's document ID's. I also linked the seat planner to the tickets for each show. This included first understanding how the other team members had implemented the seat planner and then adopting it into my already existing code. Most of this work has been using firebases built in functions in JS as well as modifying the site CSS to reflect changes.

Using the burndown chart for a reflection of how much time I've spent on this project, if every point is one hour then I've done 25-30 hours of work. That's just on the primary functions listed, but there are a lot of other random fixes that go into it. Especially with the current database functions. I spend a lot of time troubleshooting the fixes after the functions are integrated into the site so it's probably an additional 2-3 hours on top of that amount of work. Realistically I have probably worked a total of around 35 hours on the project which is slightly higher than the burndown charts amount. This is probably due to some specific functions requiring a lot more time to complete than previously assumed, also site maintenance was never listed on the burndown chart but was continually happening.

2	Read from Database	
2	update/Delete from database	

1	Listing all database entries	
1	Beautification of database tools	
1	Update Database function	
6	Link seat planner to database	
1	Continue work on edit/create/delete tickets	
1	Fix Github Issues	
2	Integrate showID into current ticket system	
1	Site Beautification	
1	Print Functionality-introduced	
5	Count Tickets	
2	Delete show/ Delete ticket prompt	
2	Remove Clickability of chosen seats	
2	Add title to Seat Planner Page	

### 3 Design Pattern

When planning the design of our web application we considered various design patterns but ended up settling with the observer pattern. This pattern has three primary functions at its core. They are as follows: The tools that the user uses to interact with the application, the backend functionality that responds to the usage of the tools, and finally displaying the results of the usage of the tools.

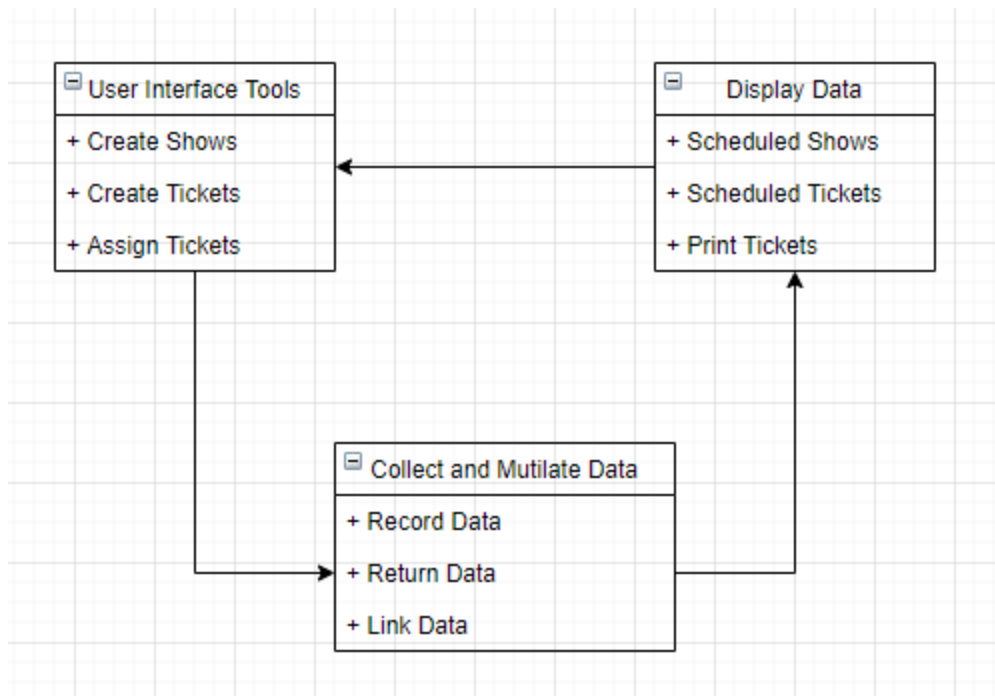
First, is the interface that the user interacts with in order to execute functionality. In our system this was done through the create shows, create tickets, and assign tickets pages. These pages gave the user the ability to input information to the system. This is where each show and customer would be properly connected together. These pages have various buttons that activate different functionalities that all work together to create a cohesive effective unit of a ticketing system.

Second, is the backend work that the web application does to insure functionality. This was done with the javascript code that connects the inputs from the user to the database. This code received the inputs from the user, formats it, then puts it in the database. This component of the design pattern acts as the transportation of resources

throughout the system. It also will return data to the interface when it is requested.

Finally, the last component of this design pattern is the display of information. This was completed through the create shows, create tickets, view seat plan, and print tickets pages. All of these pages interact with the data in the database and display it to the user in a comprehensive manner. The create shows page has all shows that are in the database displayed below the input for a new show. This is identical when it comes to the tickets page. The print tickets page grabs all the tickets for a specific show and queues them in a printing window.

Below is the simplistic diagram of our project's observer pattern. In each box are the pages and functions associated with each component of the observer pattern.



## 4 Technical Documents

### User Documentation

Our product is a simple yet effective online ticketing system. Our product allows for a user to assign seats at a movie theater online via an interface that closely resembles



the seating availability of the theater. The app lets users select available seats for a given show based on a customer's request. The app creates a data entry that is updated live as the user makes changes. This prevents selling two tickets for one seat. One of our goals was to give the user as much freedom as possible. We wanted to make it easy to open that app and start scheduling.

The software is hosted through google firebase, private hosting service with built-in database storage. A user must simply navigate to the URL [virginiacity-players.web.app](http://virginiacity-players.web.app). There the website requires them to log in with their admin credentials which we will set up. We will need the user's email to add to a white list that validates their account. Once this is done, the site will prompt them to enter their email. Once these steps have been completed the user has full access to all the features of the site. There isn't any installation or use of third-party tools when operating our website. There won't be anything that the user has to run. All the processes will be run in the background, the user will simply interact with an interface.

After the user has logged into our site, they will need to navigate to the shows page. Here they will have two options: create a new show, or view already scheduled shows. If they choose to create a new show, they will be prompted to enter the date, time, and name of the show they plan on creating. This data will be cross-checked with the existing database of shows to make sure there isn't a scheduling overlap. If there isn't any overlap, this creates a data entry into the database that will store the seating chart of the show.

Once the user has created a show, they will have three ways of interacting with the new show. First they can view the seat plan of the tickets purchased for this show. Because this show was just created the seat plan will be blank. The user can also enter tickets for this new show. This will navigate them to the ticket creation page. Lastly the user may delete the entire show from the database.

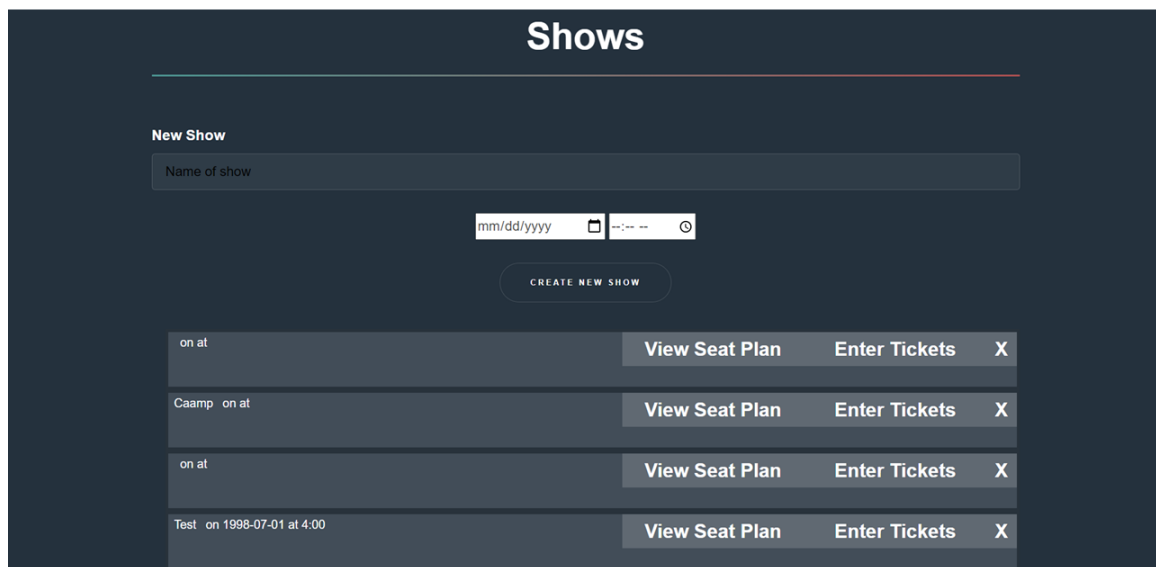
After navigating to the ticket creation page the user will then begin the process of entering tickets. They will be asked to enter the information for the customers ticket. After filling out their information the ticket they created will be input to the database. Then the user will have four options: print the ticket, assign the seat, edit the ticket, and finally delete the ticket. If the user selects print a print window will appear provided the

ticket has been assigned a seat. If the user chooses to assign a seat then the seat planner will open allowing the user to select unoccupied seats. If the user selects edit, then they will be able to update the information of the ticket. Finally the user can delete the ticket if the customer no longer wants a ticket.

In order to report a bug in our software, we created a ticketing system that will get sent to our project manager's email. The ticketing system will ask the user for their contact information, the feature they found an issue on, and to describe the issue that they are dealing with. Our team will then work to recreate the bug and fix it. We will need as much information from the user as possible because it will quicken the process of recreating the bug.

### Site Visual Guide - Step by step

## SHOWS PAGE – ACCESSED FROM THE HOME PAGE



on at	View Seat Plan	Enter Tickets	X
Caamp on at	View Seat Plan	Enter Tickets	X
on at	View Seat Plan	Enter Tickets	X
Test on 1998-07-01 at 4:00	View Seat Plan	Enter Tickets	X



## SHOWS PAGE – CREATING A NEW SHOW

### Shows

---

**New Show**

Name of show

mm/dd/yyyy  --:-- -- 

CREATE NEW SHOW

on at	View Seat Plan	Enter Tickets	X
Caamp on at	View Seat Plan	Enter Tickets	X
on at	View Seat Plan	Enter Tickets	X
Test on 1998-07-01 at 4:00	View Seat Plan	Enter Tickets	X



## SHOWS PAGE – VIEWING A SHOW

### Shows

---

**New Show**

Name of show

mm/dd/yyyy  --:-- -- 

CREATE NEW SHOW

OPTIONS FOR A CREATED SHOW ARE HERE

on at	View Seat Plan	Enter Tickets	X
Caamp on at	View Seat Plan	Enter Tickets	X
on at	View Seat Plan	Enter Tickets	X
Test on 1998-07-01 at 4:00	View Seat Plan	Enter Tickets	X

# SEAT PLANNER PAGE – VIEWING A SEAT PLAN

Seat Planner
HOME SEAT PLANNER

### SELECTED SHOW SEAT PLAN

Virginia City Players  
Seating Chart

Rows: A, B, C, D, E, F, G, H, J, K, M, N, O, P, Q, R, S, T, U

Seats: 1 through 30

RED SEATS ARE ALREADY OCCUPIED

EMPTY SEATS ARE IN WHITE

UNIQUE TO EACH SHOW, WITH UNIQUE TICKETS

# TICKET PAGE – ADDING/EDITING/DELETING TICKETS

## Editing Tickets for Test

### Create a New Ticket

First Name
Last Name
Phone Number
Email
Number of Attendees
Additional Information

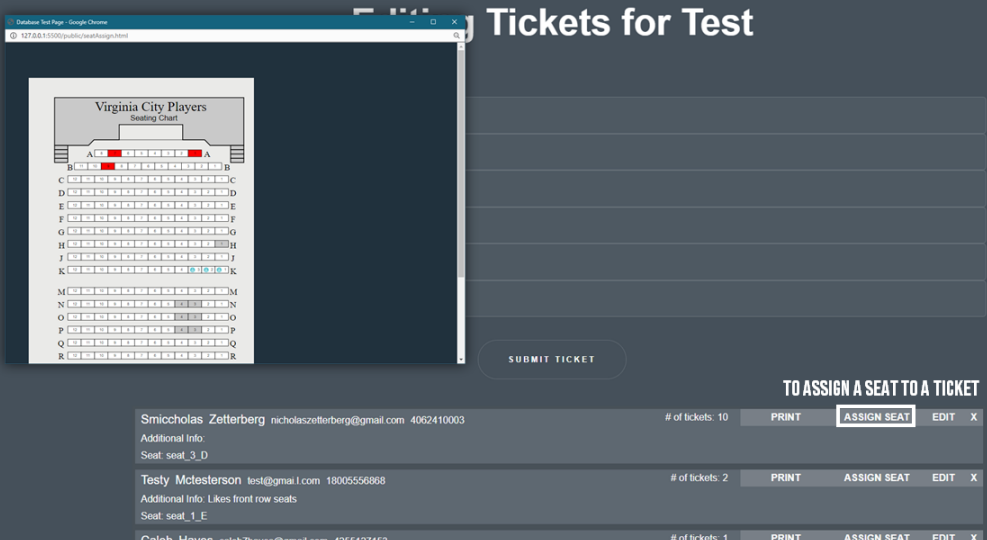
[Cancel](#)
[SUBMIT TICKET](#)

### UNIQUE TO EACH SHOW

Smiccholas Zetterberg nicholaszetterberg@gmail.com 4062410003 Additional Info: Seat: seat_3_D	# of tickets: 10	<a href="#">PRINT</a> <a href="#">ASSIGN SEAT</a> <a href="#">EDIT</a> <a href="#">X</a>
Testy Mclesteron test@gmail.com 18005556868 Additional Info: Likes front row seats Seat: seat_1_E	# of tickets: 2	<a href="#">PRINT</a> <a href="#">ASSIGN SEAT</a> <a href="#">EDIT</a> <a href="#">X</a>
Caleb Hayes caleb7hayes@gmail.com 4255127153	# of tickets: 1	<a href="#">PRINT</a> <a href="#">ASSIGN SEAT</a> <a href="#">EDIT</a> <a href="#">X</a>

# TICKET PAGE – ASSIGNING A SEAT TO A SPECIFIC TICKET

**Tickets for Test**



**Virginia City Players Seating Chart**

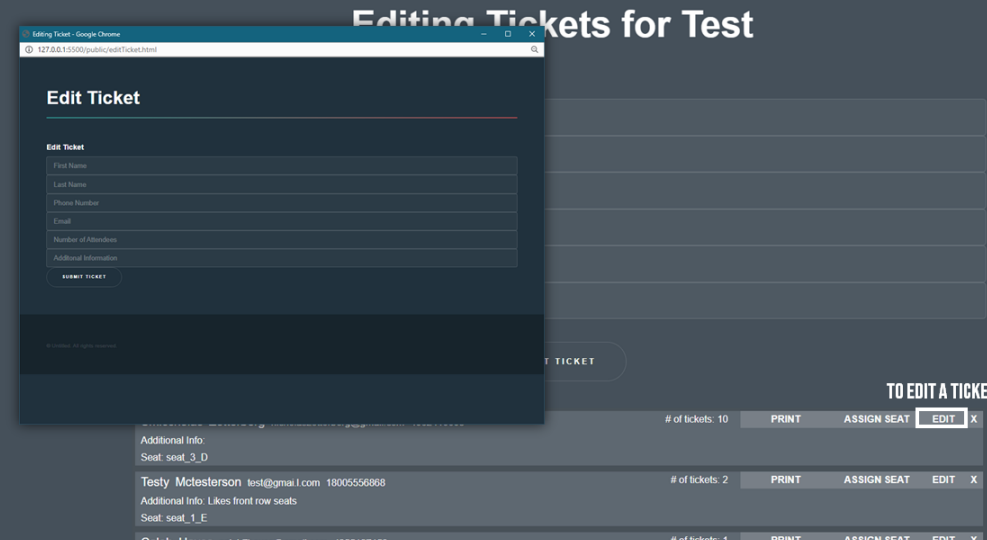
**SUBMIT TICKET**

**TO ASSIGN A SEAT TO A TICKET**

User Information	# of tickets	PRINT	ASSIGN SEAT	EDIT	X
<b>Smicchnolas Zetterberg</b> nicholaszetterberg@gmail.com 4062410003 Additional Info: Seat: seat_3_D	10	PRINT	ASSIGN SEAT	EDIT	X
<b>Testy Mcclersterson</b> test@gmail.com 18005556868 Additional Info: Likes front row seats Seat: seat_1_E	2	PRINT	ASSIGN SEAT	EDIT	X
<b>Caleb Hayes</b> caleb7hayes@gmail.com 4255127153	1	PRINT	ASSIGN SEAT	EDIT	X

# TICKET PAGE – EDITING A TICKETS INFORMATION

**Editing Tickets for Test**



**Edit Ticket**

**EDIT TICKET**

First Name  
Last Name  
Phone Number  
Email  
Number of Attendees  
Additional Information

**SUBMIT TICKET**

**TO EDIT A TICKET**

User Information	# of tickets	PRINT	ASSIGN SEAT	EDIT	X
<b>Smicchnolas Zetterberg</b> nicholaszetterberg@gmail.com 4062410003 Additional Info: Seat: seat_3_D	10	PRINT	ASSIGN SEAT	EDIT	X
<b>Testy Mcclersterson</b> test@gmail.com 18005556868 Additional Info: Likes front row seats Seat: seat_1_E	2	PRINT	ASSIGN SEAT	EDIT	X
<b>Caleb Hayes</b> caleb7hayes@gmail.com 4255127153	1	PRINT	ASSIGN SEAT	EDIT	X

# TICKET PAGE – PRINT A SINGLE TICKET

**Editing Tickets for Test**

**Create a New Ticket**

First Name
Last Name
Phone Number
Email
Number of Attendees
Additional Information

**SUBMIT TICKET**

**TO PRINT THIS TICKET**

Smicolas Zetterberg nicholaszetterberg@gmail.com 4062410003	# of tickets: 10	<b>PRINT</b>	ASSIGN SEAT	EDIT	X
Additional Info: Seat: seat_3_D					
Testy Mclesteron test@gmail.com 18005556868	# of tickets: 2	<b>PRINT</b>	ASSIGN SEAT	EDIT	X
Additional Info: Likes front row seats Seat: seat_1_E					
Caleb Hayes caleb7hayes@gmail.com 4255127153	# of tickets: 1	<b>PRINT</b>	ASSIGN SEAT	EDIT	X

## Developer Documentation

In order to start contributing or editing this product one must have have a few CLI packages installed:

- Nodejs
- git
- Npm
- firebase-tools

Next you need to make sure you have access to the VirginiaCity-Players firebase project at <https://console.firebase.google.com/u/0/project/virginiacity-players> and the git repository at <https://github.com/Quackbar/VirginiaCityPlayers>.

Clone the git repository to a safe location.

~ git clone <https://github.com/Quackbar/VirginiaCityPlayers>

Then enter into the project and install the required npm packages, then test npm

```
~ cd VirginiaCityPlayers  
~ npm install  
~ npm start
```

Then you may make changes inside the public repository and deploy them.

But first make sure your destination is correct within the firebase.json file. It should look like this:

```
"hosting": {  
  "site": "virginiacity-players",  
  "public": "public"  
}
```

Then,

```
~ cd public  
~ firebase deploy
```

If this doesn't work you may need to initialize your own firebase project and copy in the code.

```
~ firebase init
```

If you'd like to deploy to a test domain instead you may change the "site" in the firebase.json file to one of the other domains in the firebase project. Then deploy the same way.

For more troubleshooting resources checkout the docs at:

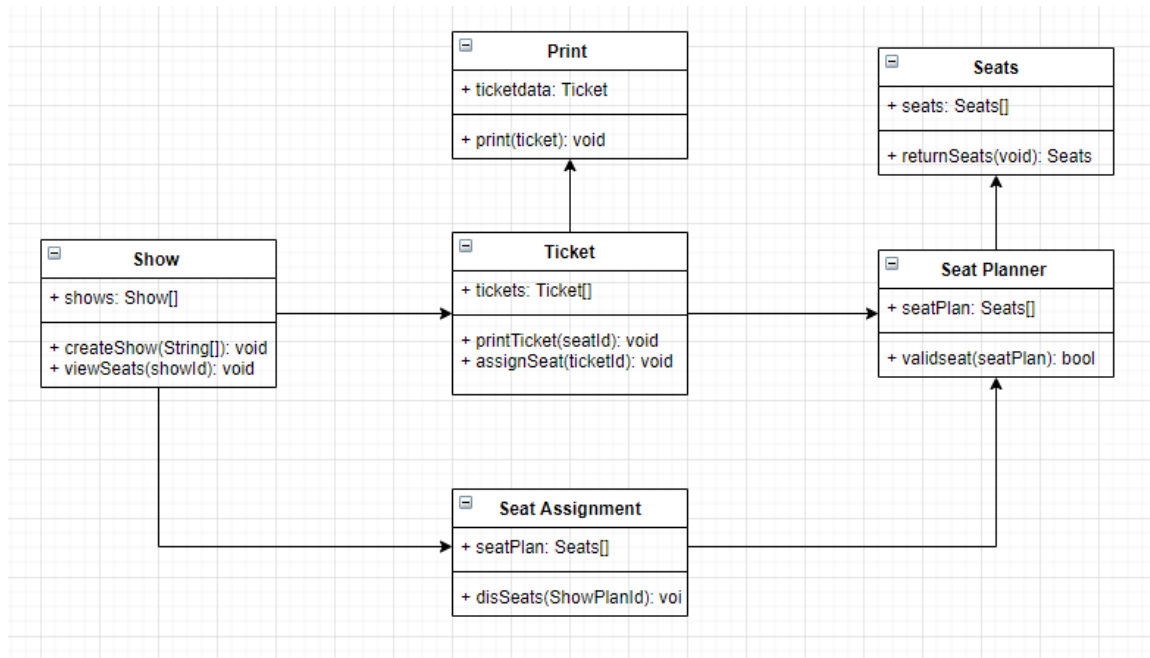
<https://firebase.google.com/>

Or

<https://www.npmjs.com/get-npm>

## 5 UML

A UML diagram showcasing the functions in our app and the database elements that are utilized for each function.



## 6 Design Trade Offs

Throughout the design process, especially at the beginning, some design concessions had to be made. Our completed project does have all of the main features we wanted to include. The biggest design trade-offs we made were on how the database was put together.

In our completed database we utilized two different Firestore collections. One is for the shows and one is for the ticket entries for each show. We made design trade offs here because we could've used either one firestore collection to hold all of the tickets and then use a unique show identifier to hold the show information in the same firestore collection. We also could've used a collection for each item within a tickets component, and tied them together with a unique identifier. We chose our method of using two collections, one for shows and one for tickets and then tying each show to a ticket by its



unique reference id. We chose to use this method because it allows for us to search through shows and tickets faster in each individual collection. If they were all housed under the same collection, any time we queried the database we would always have to search the entire database. While with split shows and tickets we can search for shows in the show collection and tickets in the ticket collection thus decreasing the time it requires to find either item.

Another major design trade off we made was with our site design. We used a free preset instead of building our own site from scratch. This came with a lot of benefits but also some major drawbacks. It allowed us to skip the work of building a site and get straight to working on the web apps functionality, it also allowed us to have a professional looking site. Though later on it proved to be somewhat annoying to work around when some team members needed to modify css for specific site elements and had to modify a css file with thousands of lines. Overall the benefits outweigh the negatives as we were able to start working on the hard stuff early and get ahead of the curve.

There were other design trade-offs, mostly quality of life features that we decided would take more time to complete than they were worth. These features included the ability to create seasons or recurring shows, and the ability to upload custom images to tickets.

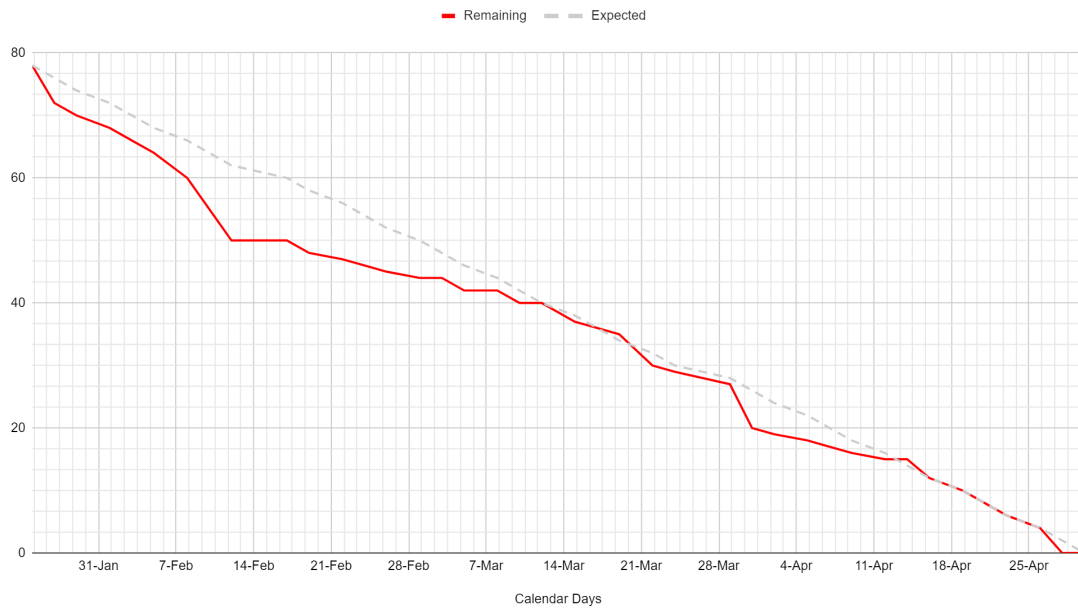
## **7 Software Development Life Cycle Model**

We used an agile development life cycle for the creation of our project. This life cycle used two week sprints to its advantage. This method was useful in defining goals and deadlines in a rapid development environment.

At the start of every spring our group had a scrum meeting which was used to define the tasks for the next two weeks. This was helpful in keeping all members accountable and on the same page. We all had an understanding of what each team member was doing and how to assist them. We also had scrum meetings halfway through each sprint for a reconsideration of tasks and roadblocks that members encountered. This strategy ensured that we would complete all tasks by the end of the sprint. Agile worked well for our team and we were happy with the pace of the life cycle.

The agile lifecycle method worked well for the most part except for one scenario. The agile life cycle caused our development to be sinusoidal in terms of progress. We would make lots of progress near the end of every sprint, however after we finished a sprint there would be little progress made for the next couple of days. This hindered our

VCP Group 2 Burndown Chart



consistent progression. Agile is about producing consistent progress but ended up producing burst progress. This outcome is not necessarily a bad thing because we were able to finish our project effectively. However, this outcome would not be sustainable in the long run. There will be sprints in which there are extreme roadblocks that pop up right before the end of a sprint that cannot simply be powered through. This is where we could have lost a lot of time.

Many of the points that have been discussed above can be visualized in our burndown chart and scrum meeting task lists.

## 8 Appendix

### Relevant Links:

[VCP Seat Planner Site](#)

[GitHub](#)

[Burndown Chart](#)