# ELE 39211 Web Applications Development project
# Team ViSi

# LINKS

Github repository: https://github.com/Valiant-GitHub/ELE-3921-Web-app-dev.-Project

Video presentation: https://youtu.be/0LtNpT7IMoo

# PURPOSE AND FEATURES

## Purpose

The purpose of our website is to be a one-stop-shop for organizing, booking and ticketing for venues and artists so that they can cater to their fans. Our inspiration is Ticketmaster and we were motivated to create a possible competitor to it, or at the very least create a similar service. With our service we are targeting customers who have not been acquired by Ticketmaster, such as more niche artists. They do not have many competitors, at very least ones that are as big. As such this presents us with an opportunity.

### The target audience:

**Venues**: primarily small venues but not limited to. Venues such as bars, community centers, cafes, and other similar settings seeking to host events and simplify their verification of tickets
**Artists**: primarily independent artists such as musicians and performers looking for opportunities to showcase their talent.
**Fans**: Enthusiasts of niche and lesser-known music scenes who want to discover and attend unique events. Not limited to music fans, as it could also be fans of standup comedy.

## Key Features

Most of the features are shown in the video we made, but here we explain them more generally and include give a brief explanation of the workflows. The features are also covered more in depth in the views and models sections under architecture.

### Artist & Venue user workflow outline

- Artist and venues create availability listings with a message, date and time.
- Then Artist or Venues can send a booking request on an availability listing. When you receive a booking request it will show on your calendar on the dashboard.
- When an artist or venue receives a booking request they can either reject or accept the booking, if accepted, it deletes the availability listing and the user proceeds to an event creation form. When the event is created it shown on the calendar. If the artist or venue reject the booking request, the booking request is deleted, and the availability listing is kept up.

## Fan & doorman user workflow outline

- Fans can follow their favorite artists, select favorite genre, and find new events to enjoy. On their home page they will find both future and past events from favorite artists and their purchased tickets.
- Doorman users will see their upcoming events to control on their home page. Will be able to verify tickets for events they are designated for.

## Event creation system

- Event Creation: Artists and venues can create events, including setting ticket prices, description, image, datetime and assigning the doorman.
- Ticket Purchase: Fans can browse events, purchase digital tickets,
- Progress Bar: A visual indicator showing the number of tickets sold, helping attendees gauge crowd size.

## Artist/Venue Matching

- Artist Availability: Artists can post their availability for specific dates, allowing venues to inquire about booking them.
- Venue Opportunities: Venues can post opportunities to host artists, and artists can express interest in performing.
- They can form an event by one of the parties sending a booking request to an availability, then user that made the availability posting can accept the booking and fill in the event form.

## User Authentication

- Registration and Login: Users can register with a username, password, first name, last name, and email address. Implemented using Django's built-in authentication module.
- A User can be a Customer (Fan), an Artist, or a Venue owner, or doorman, but only one of these roles.
- Profile Management: Users can manage their profiles, including updating personal information and viewing past events.

## Ticket verification

- Doorman user was created for ticket verification; they need to be assigned to an event when it is created so they are eligible to verify the tickets.
- When fans buy a ticket, they will receive a QR code. Which they can then use to verify their ticket with a doorman at the event.

### Event Management (Admin Interface)

- Admin Controls: Site administrators can add, update, and delete events, artists, and venues using Django's admin interface.
- Data Integrity: Ensure that deleting items does not disrupt relationships with past events and bookings.

### User pages

- Each user has their own profile page. Each user type's page communicates different kinds of information
- Artist and venues have one private and one public profile. Public profile is the ones other users see. Venues are also able to utilize the photoreel to showcase the venue.
- Doorman profile is barebones, as they solely serve the function of verifying tickets
- Fans have a profile where they can select their favorite genres, artists and set profile pictures.

# ARCHITECTURE

## Views:

1. home: Authenticates and checks the type of user. Dashboard for artist/venues and home for doorman/fans showing different info depending on user. For the fans it shows different data depending on which artist they follow. For Doormans it shows upcoming shows they need to check tickets at. Shows a landing page with a message and additional signup/sign card for unauthenticated users.

2. events: shows all the events, seperated into upcoming and past events

3. event: shows a specific event

4. buyticket: handles logic for fans to purchase tickets. Checks for event id, checks if it sold out by looking at venue capacity and the number of tickets sold, checks if the event has already passed, checks if the user is a fan. It creates a 16-digit number with random.randint() until it finds one that doesn't exist in the database. When a ticket is purchased the fan is linked to the event and the increments the ticketsold counter. Returns the user to the purchased ticket if successful and an error message if there is an issue.

5. profile: renders the profile template, different for different users.

6. signup: handles both GET(handles by else:, re-renders if invalid due to the form containing validation errors) and POST(creates a form instance with data and files, validates it and if it is valid saves the new user and automatically logs them in, then continues into the view where they set up their profile.

7. ticketdetails: shows information for a ticket, secured with an ownership check by comparing ticket.fan.user with request.user
8. createprofile: logic handling for new users to create their profile based on the selected role directly after signing up with dynamic form selection. It checks if the user has an existing profile, preventing duplicate profiles. creates a form instance with the submitted data, then if valid creates the profile, associates it to the current user and then saves the profile to the database. It then renders the template with the form in context.
9. availability: handles creation of availability records. Layered approach to access control, following the principle of defense in depth. So that only the appropriate users can create availability listings. Standard pattern for django form processing.
10. editprofile: updating role specific user profiles. Dynamic form selection and has dual form setup for the profile picture and role specific fields.
11. availability_success: renders success page for successfully creating an availability listing.
12. artistprofile: public profile for artist showing the artists events.
13. venueprofile: public profile for venues showing their events and a photoreel
14. available: retrieves all Availability objects and orders them in a chronological order passing them in the respective template.
15. requestbooking: handles the process of users booking availability listings. Retrieves availability listing, checks if the user is trying to book their own availability. We purposefully allowed the venue or artist being able to book another venue or artist, if they wish to create a collaboration.
16. bookingsuccess: renders a template when successfully submitting a booking.
17. dashboard: personalized, it retrieves availability, booking and event info relevant to the current user. Bookings and events are also showcased in the calendar with colors. It is also where the users accept or reject bookings. Renders 3 templates: availabilities, bookings and calendar_events.
18. listingdetail; allows the users to open up an availability or booking listing enabling them to read the message/description of the listing. Maybe a small oversight on our part is that we enabled the deletion of other users' bookings that were made to their availabilities, which essentialy is the same as rejecting although it can be a bit misleading.
19. handlebookingaction; handles rejecting or approving booking requests and creating an event
20. myevents: displays lists of events associated with the current user (artist or venue)

21. createevent, after a booking has been approved, this view is accessed to create an event, it prefills a portion of the fields based on existing information from the bookings, availabilities and user information. It also cleans up related records after succesful event creation. It is our most complicated view.
22. photoupload; handles upload form and processing photo submission
23. deletephoto: handles the deletion from the photoreel
24. ticketvalidation: 1st step in ticket verification, it retrieves the ticket using the provided number, verifies that the doorman is able to verify the ticket for the event it belongs. Redirects to Validateticket if no errors
25. validateticket: 2nd step, retrieves ticket, checks if ticket has been used, if unused marks as used and provides the appropriate feedback message.

## Models:

1. User(AbstractUser): extended the default django user to include fields such profilename role and enable profile pictures. Serves as the base for authentication. Models 3,4,5,6 have a 1to1 relationship with user.
2. Genre: for events, artists, fans. Fans can show their favorite genre on their profile, while artist can show what genre they perform on their profile, and artists also use it with venues to indicate the genre of the events
3. Fan: can follow artists, buy tickets, attend events, set favorite genre
4. Doorman: barebones user, can verify tickets if authorized for that event.
5. Artist: artist image for promo, be followed by fans and indicate which genre they play.
6. Venue: location(Foreign key), capicity(number of tickets available for events in that venue), venue image for promo. Venuefans(not heavility utlized), genres that are usually played at that venue
7. Photoreel: allows venues to upload images to their public profile to show off the venue. Could also be extended for other user profiles if needed but primarily created for venues.
8. Location: location usable by venues, and thereby in events. Location is a seperate model as to adhere to best relational database principles, instead of being in each model as a number of variables.
9. Availability: for creating the availability slots, either an artist or venue can create one, the model has a filter for accessing availabilities by a specific venue or specific artist (the related name). Followed by a constraint in the meta class that indicates one availabality entry in the database can only be associated with 1 artist OR venue, never both or neither.

10. Events: for individual events, including all the necessary variables (too many to list, and self-explanatory).
11. EventArtists: used to connect artist to an event, future work would include making the relation ships many to one, i.e. many artists can perform at one event. But as our service would focus on smaller, more niche artist and events, we deemed it that it wasnt necessary for the purposes of this project.
12. Tickets: indicates for which: fan, event it is and shows the ticketnumber and if it is used.
13. Booking: used to connect users with availabilities and tracks the booking status. Has Cascade, so if a user is deleted all their booking will be deleted as well. Bookings have a message with a limit of 1000 chars. The bookings have 3 statuses: pending, approved, and rejected. Also includes a created_at field which can be used for sorting in a chronological order.

## Templates:

The templates are all associated with their respective views, so we see no reason to expand further upon what the templates are for other than what is written about the views. The templates are as follows:

1. artistprofile
2. availabilityform
3. availabilitysuccess
4. available
5. base
6. booking
7. bookingsuccess
8. createevent
9. dashboard
10. editprofile
11. event
12. events
13. home

14. listingdetail
15. login
16. mapspartial
17. my events
18. profile
19. profilecreation
20. signup
21. ticket
22. ticketvalidation
23. uploadphoto
24. validateticket:
    a. Used mebjas' Html5-QRCode library to handle QR-code scanning functionality
25. venueprofile

# REFLECTION ON CHALLENGES

When we started with writing the project proposal for our event management system, we were quite ambitious with our project. while we knew this at the time, we believe it served as a motivator. We outlined 8 key features, 5 of which we successfully implemented. The 3 that we did not implement Shopping cart, Merchandise listings, order management. As they were not that essential, we were able to simplify them. For example, instead of a shopping cart and order management, we implemented the tickets to be purchased on the event page with a click of a button, this simplifies the ticket purchasing experience which can convert more people to customers.

Regarding the "personal touch" aspect of our project, we didn't fulfill all the ones listed in project proposal or as initially described. However, we did naturally come across different ways we could add a personal touch, for example the scanning of QR code for tickets is a personal touch, as well as the dashboard with the calendar we created.

Lastly we realized that while we did a good job as 2 people, we could have an even better project if we were 3. however, we still benefit from this in that we learned more of the course.

# REFLECTION ON FUTURE IMPROVEMENTS

What we primarily focused on in the development of our website was the technical aspects of it, and we therefore spent very little time styling it. What we can improve further in the future is styling it nicer. We could have expanded more on the base CSS styling, so we won't have to write styling from the top for many of our templates.

For further improvements we could implement an actual messaging system so that venues and artists can send messages back and forth instead of what we have now, which is to initiate the first point of contact and for them to continue discussions via other platforms.

We could also implement some sort of automated email-service that sends the tickets via email to the buyer. This automated email-service could also serve as a way to notify users of upcoming events from their favorite artists for example.

On a final note, one more thing that could be a good thing to implement is a sorting system for events, meaning a way for users to limit search area within a certain radius, the genre of the event, etc. This isn't something we spent time on since the main point of our website is to connect artists and venues, and we kept the search and ticketing quite bare bones.

# REFLECTION ON FEEDBACK

We had a meeting with the group "Ugoing?" and got the following feedback:

1. *implementation of live map services, for location of events.*
2. *explore the possibilities of a testing API payment system.*
3. *add a constraint on the message system in order to prevent spam and extreme length on messages.*
4. *populating the event cards with either images, number of people, artist descriptions, possible map location.*
5. *Validating the Artist users for credibility. (could be seen as an extra condition depending on the timeframe) when a user is created send an approval request to the admin.*
6. *Some restructuring of the interface for user friendliness would help with navigation, event setup, and overall experience of the website. adding placeholder pictures for aesthetics.*

7. *For the code itself organize the views by dividing them into sections.*

1) With this feedback in mind, we implemented map-services using google maps-API, which we use for choosing location for venues. The map-services significantly simplifies the location creation process.

2) We considered implementing a testing payment API but decided against it. We could use Stripe's API which seemed to offer a good sample payment system. The reason we didn't implement it is because it wasn't within the scope of our project or the scope of the course. As such, it was not integral to the main goal of the website.

3) Adding a constraint on message length was done to prevent spam. This was a good catch by the other team since we had added length constraints to all other models except the messages which was an oversight on our part.

4) We added event pictures to the event cards as suggested.

5) We didn't add a manual review of artist creation. This wouldn't have been a hard thing to implement. However, we felt that a manual review goes against the proposed idea of the website, which is to lower the barrier of entry for event booking. We want to simplify the whole process of finding venues and creating events, and having manual reviews of artists is in opposition to this.

6) We found a default event picture online which is used for all events that don't choose a picture manually. We added purchased tickets to Fan homepage, which the other team believed is what would be the most important for accessibility.

7) We commented in headers in the views with their name/function. We also used built-in VS code functionality to fix spacing and indentation in the code of the templates, views, utils, forms and models.

# SAMPLE CREDENTIALS

| USERNAME | PASSWORD | ROLE |
|---|---|---|
| **admin** | admin | Admin |
| **venue** | WebDevExam123 | Venue |
| **artist** | WebDevExam123 | Artist |
| **fan** | WebDevExam123 | Fan (normal user role) |

# AI USAGE

We used AI to a limited extent throughout the project, and it was mainly used for debugging and small improvements to code, some styling as well as for writing some commit comments faster.

We used AI to debug issues related with the validation view and with doorman selection logic in event creation. It was also used to uncover an issue where the link to media appended twice in the url for uploaded pictures.

For styling we used AI to generate simple layout suggestions which we edited and expanded on to make it more suitable to our project. For example, we used it to generate some short paragraphs for the homepage, create a basic card and create a two-column flexbox for the profile page. We also had it write a script that uses the address selected via the google maps API to autofill that information in the location text fields and then modified it to actually work as intended in our project.

When working on the booking system we used AI to make sure we hadn't missed any important pieces in the models, views and form related to booking. We also used it to find flaws and inefficiencies in the code and using that feedback we implemented fixes to it. We also used it to generate a default profile picture.

In summary, we used generative AI as a supporting tool – not to write the code for us. It helped us find bugs quicker than doing it manually, test ideas and save time on simple tasks such as styling. Everything generated using AI was reviewed and reworked as needed to fit the structure and style of our project. In addition to AI as support material, we also spent significant time reading on stackoverflow and Django documentation.

## commit messages (relating to AI):

Used AI for debugging both the validation view and debugging doorman selection.

For non logged in users i had ai write a few paragraphs of text with a call to action

Used ai to write a script to use the maps location for location form input and modified it to actually work for our project.

Used AI to create a flexbox with 2 columns in profile page.

Added booking functionality with associated models, forms, and views. event creation is still WIP

> resources utilized: various online tutorials and resources, common patterns, and error messages, solved by me with understanding of them. Utlized AI to ensure my code has no gaps, by asking for feedback and checking for possible areas of improvement, which I then implemented by myself.

Worked on a template for profile, still WIP. Made url for media. Removed media/ from url for image upload for all models, so as to not append media/ twice, once in the model and second in django automatically appending.

Used copilot for debugging the issue with media/media.

Used AI to generate default profile picture