Design

The final design of VenView synthesizes the map view of open venues with venue listings.

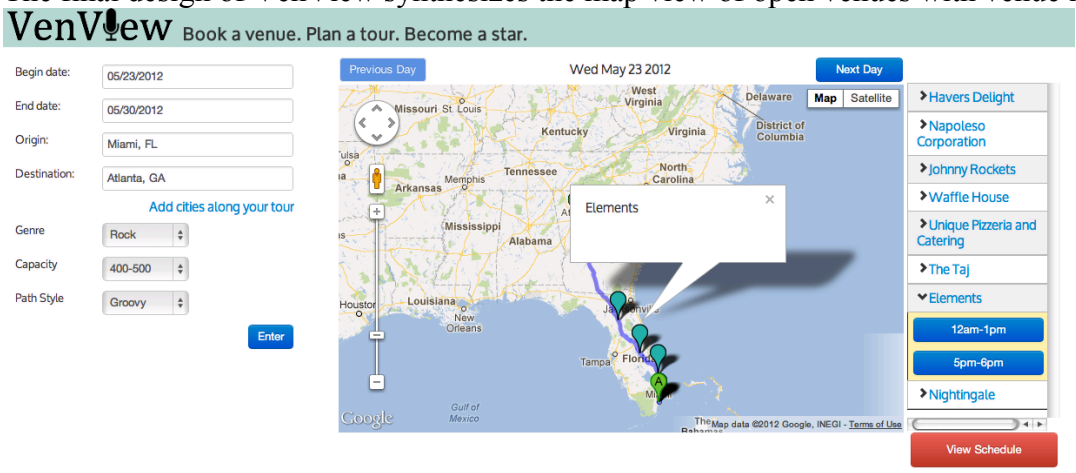

Figure 1: Final design of our project combines map view of open venue with venue listing per day of trip.

Process

Previous iterations cross referenced these separate views. However when we implemented an older design, we found the screen space limiting.

Figure 2: GR4 design included a graph and a grid of buttons.

This grid of buttons was partially inspired by HipMunk's booking interface. The plan was to cross reference venue locations (pinpointed in the map view) with rows in this grid that indicated (over the course of the music tour's length) available gig openings. Clicking one of the buttons in this grid view would trigger the booking step (task 2).

We decided for GR4 to additionally indicate (to the user) how much each available booking would pay by labeling the "to book" buttons with one to three $'s. The idea behind providing it was to avoid labeling all the buttons with redundant information (eg. "book-this-opening", "click-here", etc.). It was also to guide the user to select which venue to book based on money earning given all the openings displayed. Unfortunately by the time of our paper prototype, we had not yet considered this level of fidelity for this part of the design. As a result we were unable to test whether our dollar sign symbol scheme would work. However feedback from our heuristic evaluation of GR$ showed that users were confused by the grid of $ buttons We considered instead to indicate the number of "slots" available for each day at a particular venue on the booking button label.

Ultimately, we eliminated the grid of buttons altogether. Instead we display a map and list of venues day by day in the schedule. This was motivated by our decision to consolidate calendars. It also solved many of the usability issues with booking found in the heuristic evaluation and saved a lot of screen real estate. Feedback from our paper prototype also showed that users tended plan their trip day by day so displaying this information day by day made sense.

Previously, we had envisioned animating the columns designating a particular day so that once the user clicked a particular booking-button, the column for that day would flip and display specific opening times in addition to gigs the user had previously booked for that day. In addition to this, we had planned to provide a scheduled view so that the user could view his entire schedule after booking all desired gigs (task 3).

Our new approach uses only the full schedule view because it was counter intuitive to show a single day and the entire tour schedule separately. Additionally feedback from our paper prototype suggested that users wanted to be able to compare the booking they were currently making with those they had already booked on previous days.

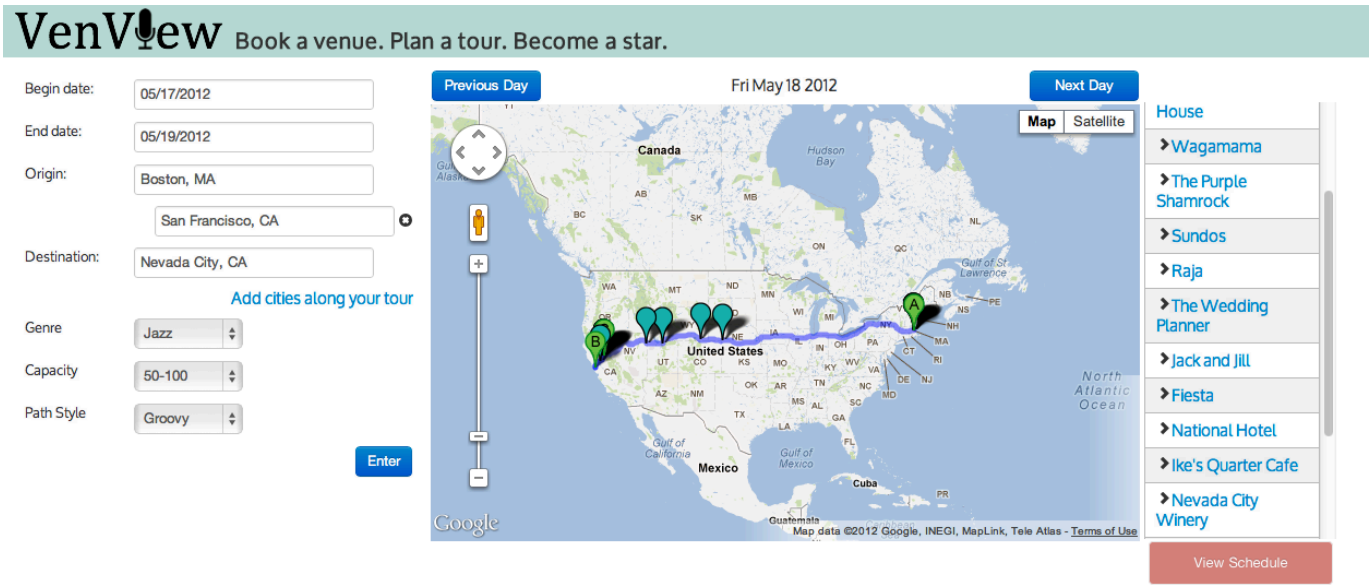

Figure 3: GR5: Map and a list associated with each day

Additionally, up to GR5 , the map view had become more of an accessory to the user's action. The user was mainly engaged with the listing view which only provided schedule information. This detracted from our original objective which was to inform musicians' tour schedule based on geography.

When we paper prototyped VenView we extensively tested how intermediate locations would be added and how to recognize the list of venues. The first element that really clicked for our final implementation was the left column where users provided the dates of their tour and locations. Since what made the interface up to this point exciting, was seeing the map populated with available venues it made sense to demonstrate this responsiveness in our interface more often. Therefore, our final design re-populates the map for each day of the tour. The pins only indicate venues with available bookings for that day within the specifications set by the user in the left hand column (genre, venue capacity, path style) .

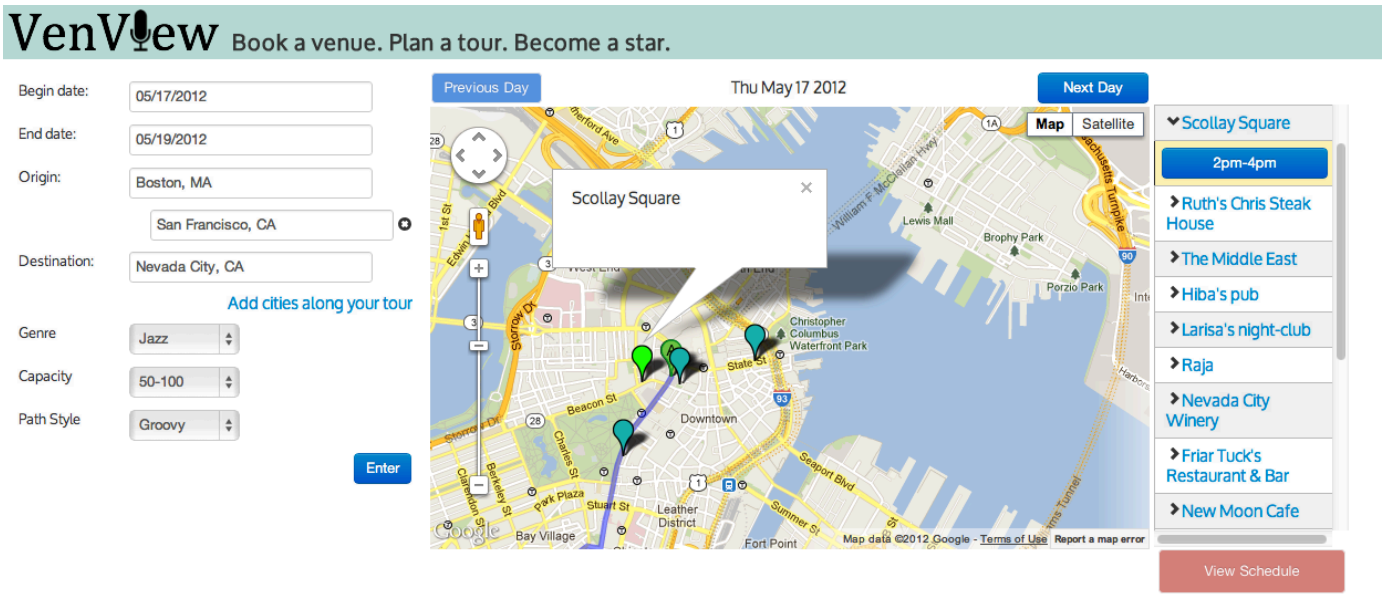

Figure 4: GR5 Design: Clicking on the marker in the map, or the venue name on the list, highlights it, and displays opening as a button.

When we originally interviewed musicians they mentioned that a longer tour in addition to intermediate cities they'd also constrain where they traveled to based on particular venues they would want to hit. Since in the the map view, a user can only reveal the name of the venue by clicking on a specific pin we decided to accompany the view with the list of the venue names. Users book a venue by clicking the time label button beneath the venue name in this list. After a user has booked a specific slot this button changes into an un-book button for that slot.

We decided to break the calendar view into fifteen minute intervals based on our preliminary interviews with musicians who said this was the most minimum time that a gig could last. This abstraction allowed for us to constrain slots so that only exactly one gig could be booked per slot.

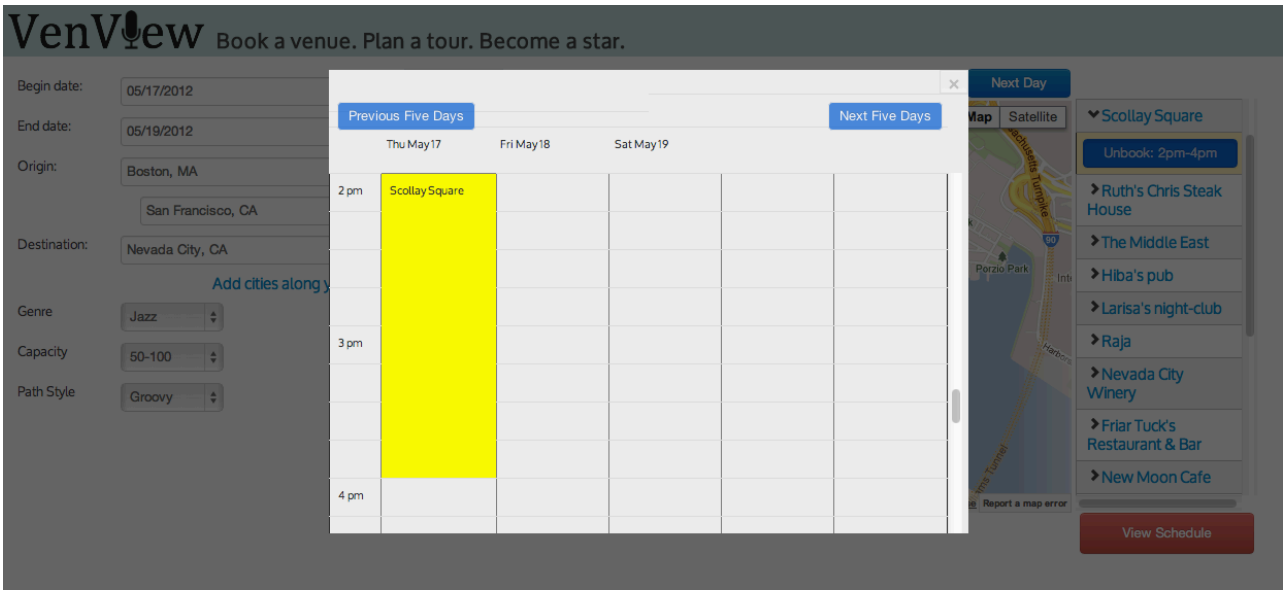

Figure 5: GR5 design: clicking on book immediately pops up calendar showing the scheduled event. We divided the calendar view into fifteen-minute slots motivated by our interviews with musicians.

Implementation

Route and Venues along the route

- Our application used the Google Maps API to get routes. It used the intermediate cities entered by the user as Waypoints to request from the Google direction renderer.
- The number of intermediate cities had to be limited to three to reduce calculations. The calculations ensure right behavior when removing the first city and adding a new one. i.e. The new one would be placed at the end and not as the first. It also ensures that the cities are fed to the waypoints in a route in the right order.
- The application used the Google Places API search to get venues within a 500 radius around the origin and destination. The initial design was meant to use this search service to get venues along the path, using the (longitude, latitude) of points along the path. However, Google restricts users from iterative calls to a service and the data along the path had to be hardcoded. That problem took a long time to figure out, it should of have been specified in the Google API documentation.

Venues: map and list

- The system used random algorithms to choose venues open in a certain day, and similarly to choose the time openings of those venues ensuring no collision in time openings, yet allowing venues to be open on more than one day. This information was saved in a list that resets every time the application is restarted.

- The application used Google markers to indicate venues that are open along the path at a given day. It used different colored markers to indicate venues: that are open, that have been booked in another day, and venues that are currently selected.
- The behavior between the list and the map: the application stored a list of venue names per day, and the longitude and latitude of a respective venue. This information was used to calibrate the venues on the list and the venues on the map. i.e. selecting a venue on the map would make the list scroll to it, highlight it and show its available bookings. Vice versa, selecting it on the list would make the marker on that venue change color to green and an information window would appear on the map, to give the user feedback of the venu's location on the map.

Standard Behavior

The application was made consistent with other applications by having any selected element be deselected if the user clicked on another element. i.e. if the user clicked anywhere on the map and a marker was selected, it would be unselected and all expected behavior would propagate to be reflected on the list. Additionally, when the user is in the "view schedule" mode and he clicks outside the schedule, the schedule automatically closes.

Booking

After the user enters the travel information, we generated random number of booking objects for each venue, on each day of that trip. These bookings are all assumed to be open and are associated with the venue object and its marker on the map. These bookings are then displayed in the html table for each venue on that day. Upon clicking on a venue, the marker on the map is then highlighted, and the available slots to book are displayed. Vice versa, when a marker is clicked the associated venue in the table is also highlighted, and the available times are displayed.When a user selects the booking, the opening slot is immediately designated as 'booked', and the schedule pops up showing the booking. In addition the marker is highlighted in a darker color to designate that the venue is booked.

Scheduling

Scheduling was done by just drawing formatted html table as a calendar. When a venue is booked, the cells associated with that time in the table are just highlighted and formatted to reflect that booking. The schedule pops up immediately as feedback for booking.

Implementation problems affecting usability:
The inability to detect and warn of conflicts in scheduling really affected the security of using application.
Additionally, in order to allow for multiple window sizes and because we used blanket, we were unable to create schedule rectangles of set width/height/location. This meant that schedule

rectangles representing a booked-gig could not be interactive as we had originally planned. Instead we filled in corresponding cells in the schedule table and made the schedule purely a view (the user could not unbook slots from the schedule-popup).

Evaluation

"Describe how you conducted your user test. Describe how you found your users and how representative they are of your target user population (but don't identify your users by name). Describe how the users were briefed and what tasks they performed; if you did a demo for them as part of your briefing, justify that decision. List the usability problems you found, and discuss how you might solve them."

Testers were mostly friends willing to test our UI. One user was a musician we had interviewed in GR1 and was happy to look at our final design. Tests were conducted by just briefing the user verbally with a quick description of the UI, and what it can accomplish. Next they were given three tasks to accomplish: 1) Find the venues along a tour they are planning. 2) Book at least one of the venues. 3) Check their schedule. Then the user was given the freedom to explore the UI while the observer took notes, and the facilitator just guided the user at some points if they had questions or were stuck.

Tester 1

User was a pianist friend. She does not perform very often but was quick to impersonate the role of an independent musician on tour.

Heuristic evaluation

Did not notice date at top of the map: Heuristic: Safety, Learnability, Severity: Major

User was unable to distinguish at first that the map was for only one day. She ended up booking two venues that are very far apart on the same day.

Solve this by doing some user testing on how best to display the date to the user.

Had trouble locating venue: Heuristic: Efficiency. Severity: Minor

User spent a long time zooming in after selecting the venue to locate it on the map.

Solve this by: Zooming the map view into the venue whenever it is selected in the list adjacent to the graph.

Users double booked: Heuristic: Safety. Severity: Catastrophic

User double booked an evening because the application did not have the capability to report a conflict. They were then confused about what happened. "Where did my event go?"

Solve this by reporting conflicts in scheduling to the user. We would have to research what the best way of reporting conflicts.

User was surprised the calendar was static. Heuristic: Matches the real world.

User tried desperately to click on the calendar to view more details or to see if they can edit the booking.

Solve this by trying to create a more interactive schedule that matches the real world display of calendars often allowing users to see details of the event and edit them.

Unclear what path style refers to in the design. Heuristic: Help + Documentation, overall Learnability. Severity: Major
Unfortunately this was a suability problem detected earlier which we didn't get a chance.

Tester 2:

MIT course 6 friend in concert choir. Not exactly a touring artist but a performer and familiar with concerts.
Placement of "View Schedule" button. Heuristic: Efficiency, Severity: minor.
User wasn't able to see it. User reported that it would have been better if it was placed on top of the list and not under it.
Solve this by doing user testing to see the best place for the "view schedule" button and place it there.
Legend for marker colors. Heuristic: Learnability, Severity: minor
Solve this by inserting one under the "enter" button when the map appears
Color Choice. Dark Blue not best color for permanent reservations. Heuristic: Learnability. Severity: Cosmetic
Solve this by doing user testing to see the best color and use it
Cross-application Calendar. Be able to download scheduled events into Google Calendar. Heuristic: Efficiency. Severity: major.
Solve this by implementing google calendar in the application. That way it's easier for the user.
Lack of geography knowledge. Doesn't know cities between two states to enter as intermediate cities. Heuristic: Efficiency, Severity: major.
Solve this by making the map visible at an earlier stage of entering inputs. As in when the origin and destination are entered, the map appears showing the intermediate cities.
Ordered list. list orders venues from origin to destination [Good: intuitive]

Tester 3

Swiss saxophonist who we interviewed for GR1 and inspired much of the UI design. We got in touch and he was interested in seeing our work. He tours frequently.
Unclear that available destinations should only be in the USA: No warning exists to indicate to the user that she should only pick locations within the USA. Heuristic: Learnability, Severity: Minor.
Solve by: having some warning message if the user tries to enter locations outside of teh US
Path between pins seemed to indicate time/Unclear that map only indicated 1 day: It was not completely obvious that the displayed pins represented only one day. Since the user picked a long tour time + distance, she inferred that the represented pins must not be all available within the time span, but because the pins were drawn relative to a path, she thought some amount of time > 1 day was being represented.
Heuristic: Learnability Severity: Major
Solve by:

-Incrementally drawing the path once venues have been booked.
-Displaying open venues only a certain radius from the inferred position of the user (eg on day 1 of a five day tour, do not show openings at the final destination).

List and Pins correlate: Number of pins match the number of venues listed. Once a pin is clicked, its corresponding venue is clicked (available times are displayed). Heuristic: Learnability + Efficiency (clicking either view begins "booking" transaction.

Unclear that Names Listed are Venue Names: Heuristic: Learnability, Severity: Major
Solve by: Labeling column of venue names.

Unclear as to how to initiate booking process: User guessed that clicking venue names would display more info about the venue, not necessarily initiate the booking-process for that venue.
Solve by: Labeling Venue-name column with "Book a Venue," or automatically reveal available times for the first venue in the list (display buttons for different available times as if the first item had already been clicked to entice the user to either book those times or click other items in the list to display more information).

Tester 4

MIT friend with a music minor. Performed with gamelan group at MIT but otherwise does not tour.

Unclear location of venues. User doesn't know where the venues are in the list. She prefers headers in the list separating venues in the origin, destination and the ones between cities.
Heuristic: Learnability. Severity: Major.
Solve this by putting headers in the list that are greyed out, just to help the user visualize where the venues are.

Address information. The information window has to have information about the venue.
Heuristic: Learnability. Severity: minor.
Solve this by putting the venue information. That way it is more consistent with Google Maps as well.

Day on top of the map unclear. The user wasn't able to tell that it's a day by day map until a while after she started. Heuristic: Learnability. Severity: minor.
Solve this by making the dates and days bigger and clearer. The user eventually figured it out. It is only a matter of first time using.

Reflection

While we did each step of the spiral design process and tested with users along the way as described in lecture, I think we failed to think deeply enough about the implementation along the way. Had we thought more about the details of our implementation ahead of time, we would have perhaps updated to our final view earlier on in the process and would have been able to iterate on the exact details of it further. Other aspects of our design did benefit from this process (such as the left-hand column view in which additional intermediate cities are added).

When paper prototyping our design in GR3, we found some usability issues but we struggled to solve them. One of our biggest struggles was how to display the two dimensions, time and space (map) for the venues effectively and intuitively. We were unable to let go of any of these dimension, because they were both essential to the tasks. So when GR4 approached we realized the problems but still had not pinned down how to solve them. In addition the design that seemed feasible by paper proved much more difficult on computer because of screen real estate as well as implementation. As a result GR4 had many usability issues. The final design that we came up with for GR5 addressed many of the issues we discovered in our user testing and heuristic evaluation. However, because we came up with it so late in the process, we were unable to prototype it and user test it. As a result many of the details in our final design were not tested and fully fleshed out in advance. The final round of testing thus highlighted mostly new usability issues.

The way we worked on the implementation was basically by building on the user input one module at a time and by ensuring that the new modules communicate with the previous ones properly. If we had decided on specific data structures at the beginning of implementation, we could have been able to work on different modules simultaneously. However, since we were working with APIs we haven't worked with before, it was difficult to anticipate how our data would look.