PASSION PALS

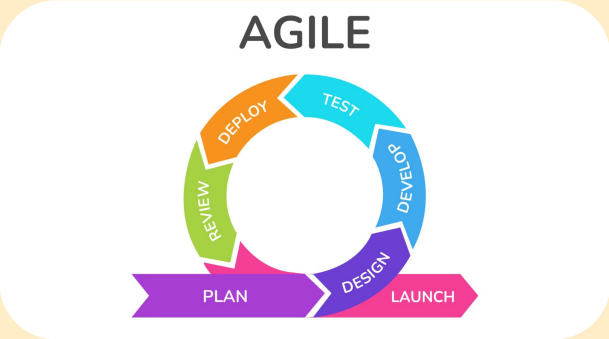
Connecting Passions, Creating Friendships

DEMO

PROCESS DISCUSSION

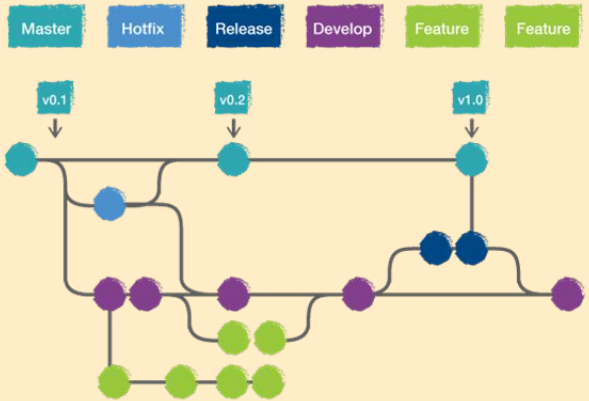
TEAM COLLABORATION AND COMMUNICATION

- Held regular sprint planning sessions, stand-up meetings, and sprint reviews to ensure efficient progress
- Set up a dedicated Discord server for seamless team discussions and meetings
- Posted standups on Slack



EFFECTIVE USE OF GIT/GITHUB

- Version control played a pivotal role in our software development process
- Utilized the branching feature so our team could work on various features concurrently without worrying about conflicts arising between the changes made by different team members
- GitHub's pull requests streamlined our code review process, ensuring high-quality code before merging
- Collaborative workflow not only improved code quality but also fostered a sense of accountability and shared ownership of the project



DIVIDING TASKS

- Utilized Jira to create user stories and assign tasks, keeping our workflow organized and manageable
- This division of tasks allowed us to work on multiple fronts simultaneously, ensuring timely progress
- Implemented user story points for task estimation, ensuring an equal distribution of work among team members and promoting balanced contributions
- We made sure to divide tasks during team meetings, as it ensured alignment and contributed to a smooth and efficient development process

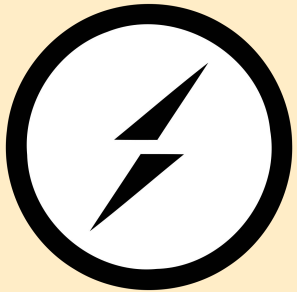
NAVIGATING SIGNIFICANT DECISIONS

1. **User Story Points in Jira (Good)**
    - Decision: Adopting the user story point estimation system in Jira for better planning
    - Rationale:
        - This choice allowed us to estimate the complexity of tasks more accurately and prioritize tasks effectively within each sprint.
        - The ability to visualize work progress and allocate tasks efficiently enhanced team coordination and productivity.
2. **Effective Communication with Discord (Good)**
    - Decision: Creating a Discord server for better communication.
    - Rationale:
        - Opting for Discord improved communication within the team, providing a platform that accommodated larger group calls for stand-ups and discussions compared to Slack
        - The server's channels facilitated progress updates on different features and components, allowing us to stay organized and informed despite varying schedules.
3. **Excessive Use of GitHub Branches for Development (Bad)**
    - Decision: Creating numerous branches for every small task or bug fix.
    - Rationale:
        - The rationale behind this decision was to minimize conflicts, but in practice, it led to a complex branch hierarchy and difficulties tracking changes across various branches.
        - This led us to realize that a more balanced approach to branch creation and management was needed for smoother collaboration and version control.

TECHNICAL DISCUSSION

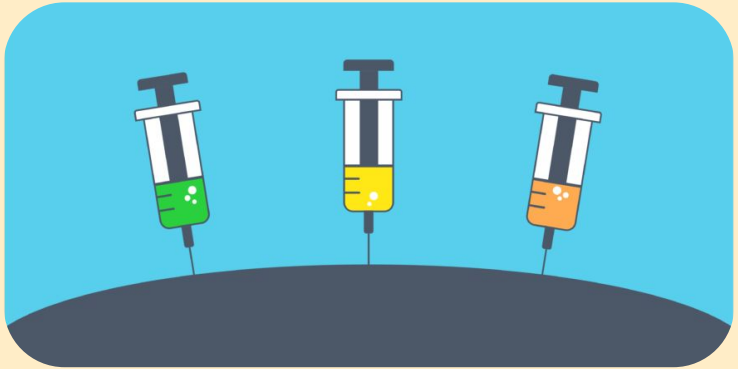
ITEM#1:
REAL-TIME INTERACTION

- A technical item that came up during our project was the use of sockets, as we used them heavily for the chat, event, and waitlist functions
    - There were several challenges working with sockets, as we had to ensure that the data being emitted would be consistent with the database, and that our client would be updated with correct data at the correct time



ITEM#2:
DEPENDENCY INJECTION FOR BUTTON HANDLING

- Dependency Injection
  - Injected functions to handle button presses for event cards, so it became more portable and easier to test
  - Same with some pop-ups, created a barebones skeleton for a pop-up, and let caller provide the contents for the contents, as well as managing the state of the popup



ITEM#3:
DESIGN CHANGES

- We made several design changes during development
    - Did not include our “Calendar” page, which would show all upcoming dates in a calendar. It was found redundant as our listed enrolled events already included dates
    - Changed the UI and general theme from our original Figma design, as we decided our new approach was overall more aesthetically pleasing and functional
    - Between sprint 3 and 4 we retooled our dashboard, merging it with our Find Event page to create the homepage, as we found that the dashboard’s purpose was too similar to Find Event’s

SOFTWARE ARCHITECTURE

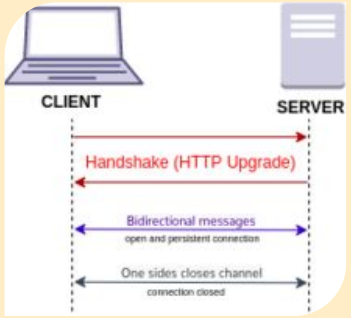
ARCHITECTURAL OVERVIEW

- Passion Pals' architecture consists of three core components: the front-end, the back-end, and the database.
- Front-End Component
  - Responsible for the user interface and user experience of Passion Pals
  - It encompasses everything that users interact with directly on the website
  - Users can navigate through various sections, subscribe to themes, browse events, enroll in events, and engage in chats
- Back-End Component
  - Manages user authentication, event creation, recommendations, messaging, and much more.
  - The back-end ensures that users' interactions with the front-end are accurately processed and responded to.
  - It communicates with the database to fetch and update data as needed.
- Database Component
  - Stores all the data crucial for the functioning of our app
  - This includes user profiles, event information, chat messages, and user enrollments
  - The database ensures that data remains consistent and readily available for use by the other components

Interaction between components

- Interaction between these components is orchestrated through API calls and data flow
- When a user interacts with the front-end, such as subscribing to themes or enrolling in an event, the front-end sends API requests to the back-end
- The back-end processes these requests, applies the necessary logic, and communicates with the database to retrieve or update data
    - For instance, when a user enrolls in an event, the back-end updates the user's enrollment status and the number of spots remaining in the event itself

TECHNOLOGIES AND TOOLS



- **Front-End Framework: React**
  - Developed dynamic and interactive user interfaces with React.
  - Utilized React components for modular design and code reusability.
- **Back-End Server: Node.js**
  - Built a robust back-end server using Node.js for handling requests and managing data.
  - Employed Express.js framework for streamlined routing and middleware management.
- **Database Management: MongoDB**
  - Chose MongoDB for secure and scalable data storage.
  - Designed efficient database schemas for organizing user profiles, events, and interactions.
- **Real-Time Communication: WebSockets**
  - Implemented WebSockets for real-time communication and instant messaging during events.
  - Enhanced user engagement by enabling seamless interaction among participants.
- **SMS Notifications: Twilio**
  - Integrated Twilio for sending SMS notifications to users about event updates.
  - Enabled effective communication and timely reminders via text messages.

twilio

M E R N

INDIVIDUAL CONTRIBUTION

ROGER

- Created and Joined event display (old dashboard)
- Navbar
- Locations
- Location, theme, recommended filter and optimizing concurrency between search options
- Homepage UI
- Login signup UI
- Friend page UI
- Theme selection UI
- Location popup
- Friend request popup
- Interest Popup

NISHU

- Profile and edit profile
- Direct and group chat
- Event enrollment
- Logout
- Email verification
- Theme filter on home page
- Front-end general touch-ups

AMITOZ

- Create Event
- SMS Notifications
- My created events
- Delete events
- Profile setup page

MUSTAFA

–Basic Set-up for the project with initial dependencies
–Basic backend for login and signup
–Front end and back end for themes subscription
–Backend for add/delete friends
–Backend for friend request acceptance
–Backend for creating direct message object for current friend
–Backend for creating Group chat conversation object

NIC

- UI for event cards, and searching for events
- Popup skeleton code
- Update/Edit event
- Pagination for searching event pages
- Learned how to patch data
- Filter which friends are in an event, to show on event details

DARREN

- Early UI Design on Figma
- System Design Diagrams
- Event Enrollment
- Live event-spot updates
- Implementing Waitlisting
- Front-end touch-ups

THANKS!