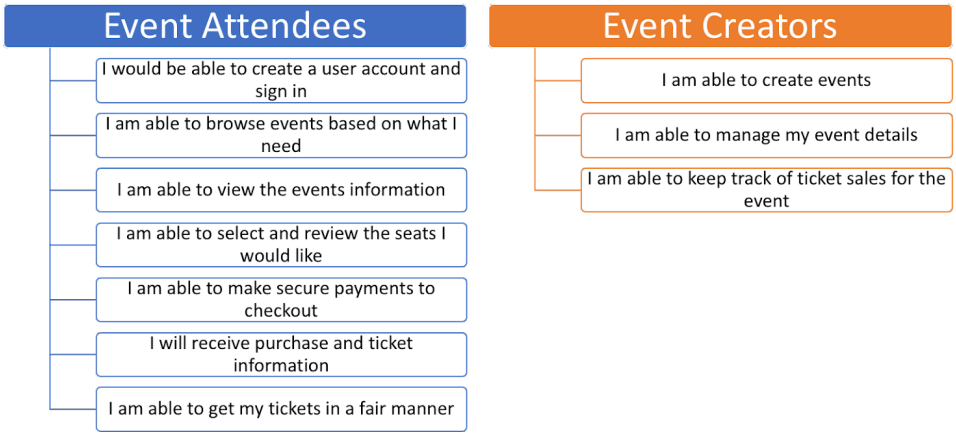
Introduction
Bluetix is an event ticketing platform designed to cater to both customers and event organisers. What sets it apart is its innovative use of blockchain technology within its ticketing system that brings several key advantages, including enhanced security, transparency, and protection against spoofing. Bluetix aims to streamline transactions and create a more trustworthy environment by leveraging blockchain technology to mitigate scalping. This will work alongside a priority queuing system to ensure fairness in ticket allocation. All of this is executed with a user-centric design in mind, while ensuring scalability.

Software Development Process
Our project adopts the Scrum methodology for its agility and flexibility that fits this large scale project into a tight time frame. By focusing on feature delivery and producing tangible products, we can have instant feedback and prevent misdelivery. Our weekly meetings, led by Scrum Master Favian, are held to discuss the user stories to target. The team is divided into frontend (Favian, Si Wei, Izzat) and backend (Timothy, Boon Jhee, Axel), both actively contributing ideas and ensuring achievable deliverables. Each sprint is estimated to last 3 weeks and tackle at least 3 user stories each. Expecting 3-4 sprints spanning 12 weeks.

i. User Stories
We have split our user stories into 2 different epics, Event attendee and Event creator.

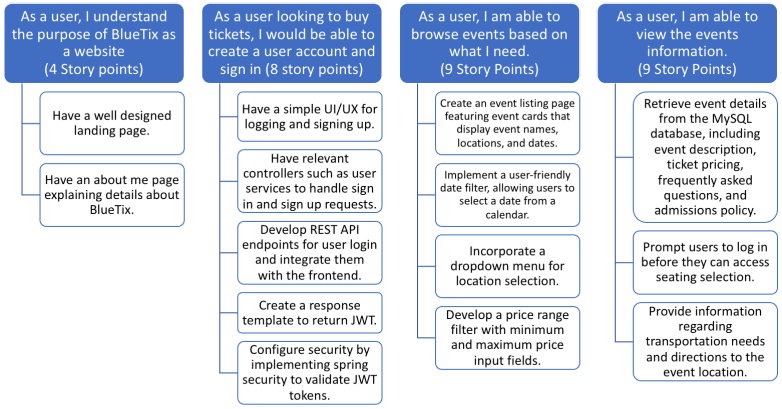


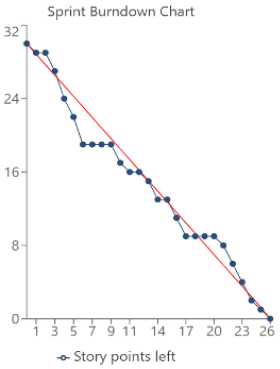
Ii. Backlog Progress
The following table captures the addition & removal of user stories over time sorted by date.

User Story	Action	Date	Reason
I want to easily sell and list my tickets for sale	Removal	30/8/23	Out of scope; an optional extra feature.
I want to be able to offer feedback	Removal	30/8/23	Out of scope; an optional extra feature.
I want to be able to monitor ticket attendance through a dashboard	Removal	2/9/23	Out of scope; an optional extra feature.
I want to have a seamless verification process for the ownership of tickets.	Addition	26/9/23	Crucial for blockchain product's functionality. Encompasses the verification of NFTs when queuing to enter the event.
I want to have a seamless verification process for the ownership of tickets.	Removal	14/10/23	Tight deadline, decided that it was an optional feature.

Iii. Sprints
Below are user stories separated into sprints and tasks.

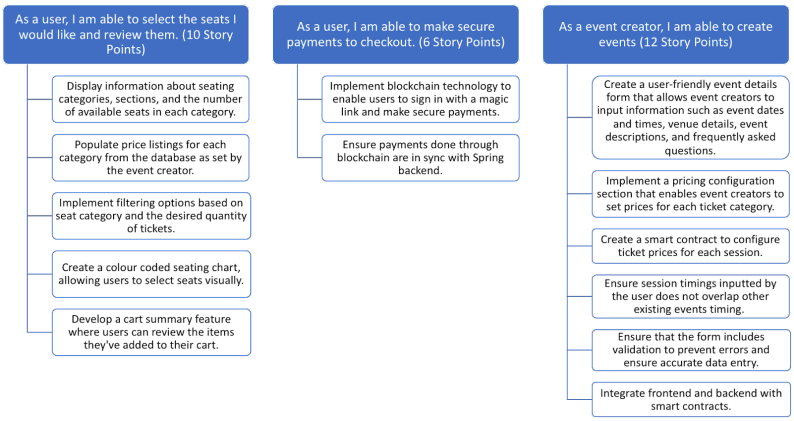
Sprint 1 Review:

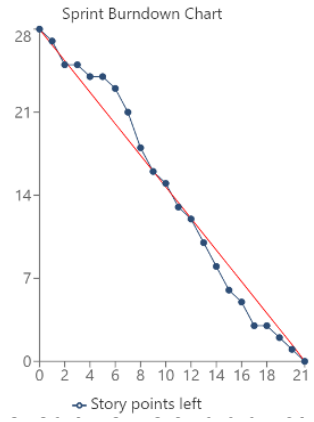




Based on our current progress, we are punctual in keeping with our established timeline and did not deviate much from our schedule based on the burndown chart above. During this sprint, we recognised the need for multiple iterations on our landing page, introducing design enhancements and innovative aspects. Initially, our design on Figma was static with just an image background. Consequently, we incorporated dynamic elements into our design, including a video composed of stock footage and an interactive carousel, improving the visual appeal of the page.

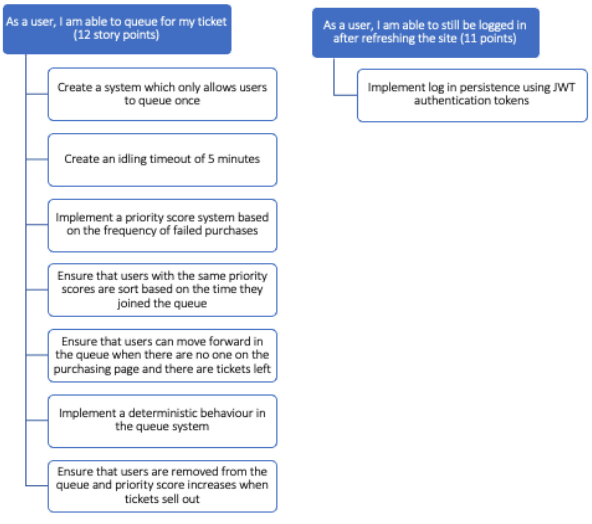
Sprint 2 Review:

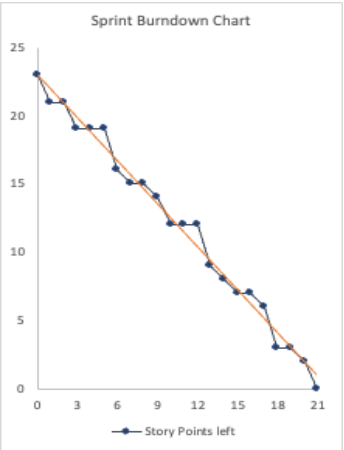




Based on our current progress, we are punctual in keeping with our esta deviate much from our schedule based on the burndown chart above. A slight deviation in the burndown chart is due to a challenge we faced which was the seating chart. Seating chart was something new to us and much research was needed to create a selectable UI with seating section details.

Sprint 3 Review:

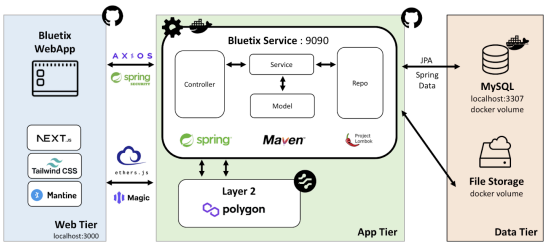




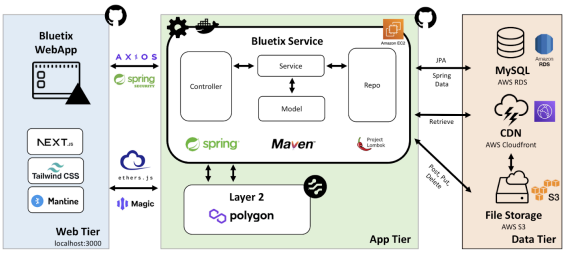
Based on our current progress, we are punctual in keeping with our established timeline and did not deviate much from our schedule based on the burndown chart above.

Software Design

Web Application Architecture - Dev



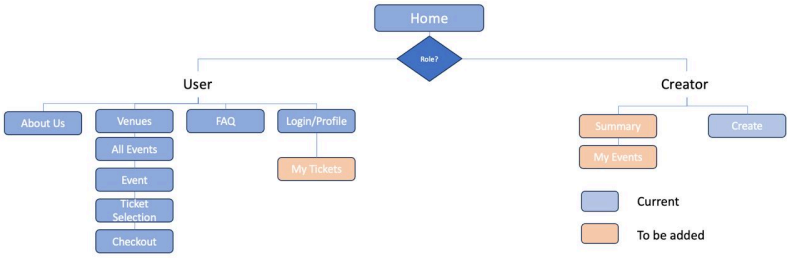
Web Application Architecture - Production



Bluetix employs a microservices architecture, facilitating collaborative development via GitHub and consistent environments through Docker. The application is structured into: Web, Data, and App tiers.

Web Tier

Next.js provides server-side rendering, making it ideal for content-rich websites with frequent API calls and flexible routing. Tailwind CSS offers modular, developer-friendly styling with a mobile-first design principle. Here's the page structure:

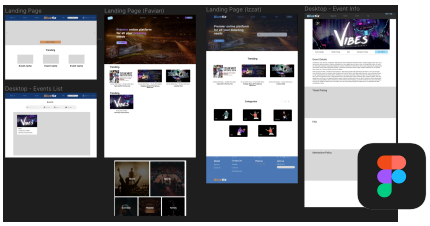


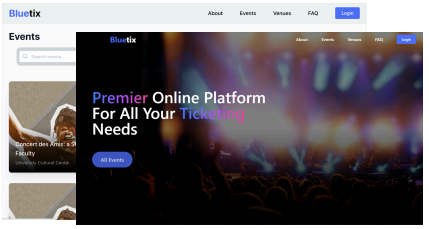
UI Design

Colour Choice: We selected blue as our primary colour to evoke trust and security in users, crucial for any ticketing platform. These qualities ensure users feel confident in their transactions.

Layout Simplicity: Our design adopts a minimalistic layout with creative accents, boosting usability and maintaining sharp focus.

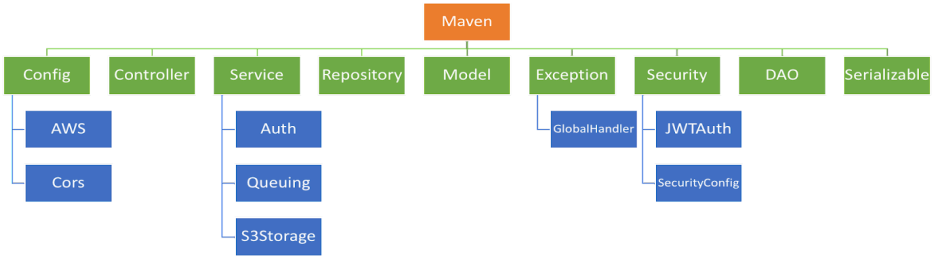
Inspirational Foundations: We draw inspiration from industry giants like Ticketmaster, Eventbrite, and Oatside, known for their minimalist approach and large user base.





App Tier

Spring Boot is chosen for its flexibility, scalability, and cross-language integration. Maven is used for well-structured project management, while Lombok streamlines annotations.



The Spring application Dockerised, and uploaded to DockerHub which the AWS EC2 will pull, build and run automatically using the CI/CD from GitHub actions.

API Endpoints

The API endpoints use Spring's @RestController and individual repository, and service. For functions requiring calls to multiple services, such as third user story in Spring 2 for event creation, a dedicated

controller class (e.g. CreaterController) is created. In addition, custom SQL may be made. To provide security, each controller goes through the SecurityConfig controller, which provides authorisation depending on user roles.

Sample API Endpoints:

HTTP	Sample API Endpoints	Service Called	Description
GET	/sessions/	sessionService.findAll()	Retrieves all sessions
GET	/sessions/byEventId/{eventId}	sessionsService.findById(event_id)	Retrieves by FK Event Id
POST	/storage/upload	storageService.uploadFile(file, fileName, filePath)	Creates a new image file in S3 bucket
DELETE	/queue/leaveQueue/{eventId}/{sessionId}	queueService.leaveQueue(eventId, sessionId, user)	Removes users from queue

Blockchain

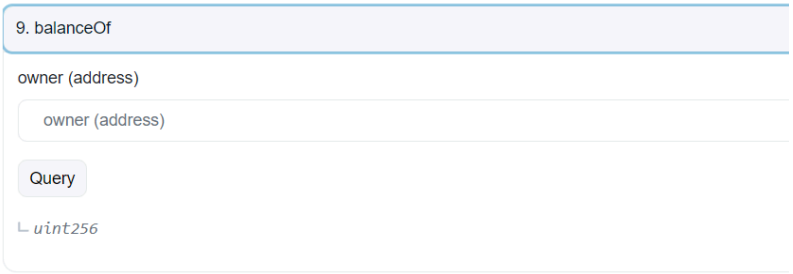
Smart contracts largely inheriting the ERC-1155 multi-token standard have been deployed on the layer 2 network, Polygon proof of stake, for scalability and cost-effective transactions. Polygon can host up to 65,000 transactions per second with an average gas fee of 0.0001 USD. Also, event tickets are tokenized as NFTs for several reasons, mainly:

Pros	Description
Programmable tickets	Specific business rules can be encoded into tickets e.g. ensuring the price cannot go beyond a certain value or ensuring a portion of resale value contribute to event creators.
Ownership verification	Since every ticket is mapped to an owner's address, owners only need a signature of the message. This message with their public key can be verified to prove they own the address.
Immutability	Ticket details are immutable and cannot be changed. This prevents centralised entities or any fraudulent actor from sabotaging the tickets.
Interoperability	The interoperable nature of NFTs make it easy for integration with other platforms all while maintaining its immutability. For instance, NFTs can be easily integrated with Opensea, an NFT marketplace platform for NFT trading.

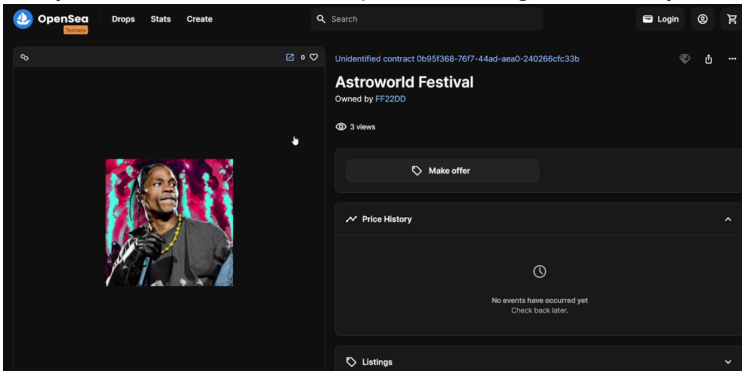
To facilitate smooth transactions for users with and without cryptocurrency wallets, we've integrated the Magic API. This enables users without crypto wallets to effortlessly create a public wallet on Magic using just their email address. Any assigned tickets will be linked to this public identifier, allowing easy retrieval using the same email address in the future.

Further, we used Pinata to upload the NFT images and NFT metadata to the InterPlanetary File System (IPFS). This gives us a content identifier (cid) which enforces immutability upon purchased tickets. Any change in metadata, or the image of the NFT would change the CID, rendering it invalid.

The verifiability of the blockchain is delivered through a decentralised ledger that allows anyone to view the owner of a certain NFT. EtherScan or PolygonScan are examples of these ledgers that track all activities relating to the blockchain on which the application is deployed. To verify if someone owns a ticket, individuals can utilise conventional functions such as "balanceOf" or "ownerOf" from the NFT smart contract that checks to see if someone is an owner of one of the minted NFTs.

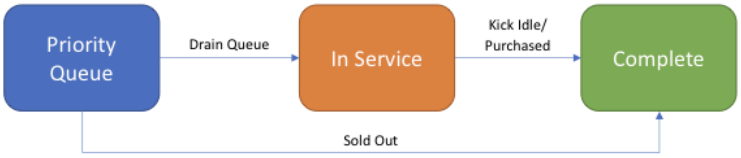


Our NFTs are also integrated with Opensea, an NFT trading platform to provide users a means to trade NFT tickets. This increases liquidity and hypes up concerts for event organisers. Once users mint NFTs from event organisers, they can find their NFT on Opensea through which they can choose to list for sale.



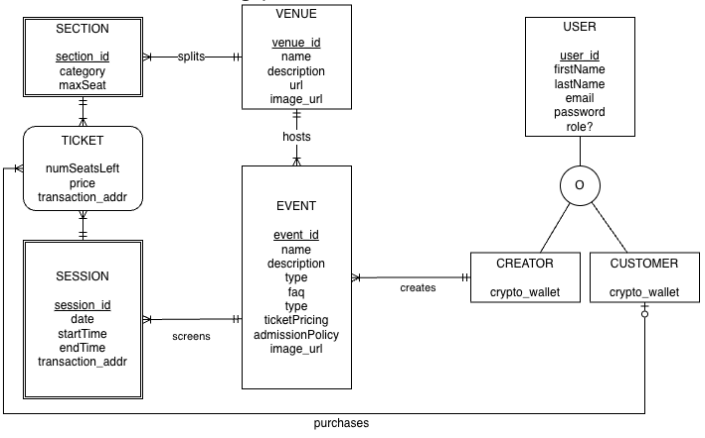
Queuing/Fair Distribution

In our fairness queueing system, we prioritise a deterministic approach, eliminating random ticket allocation. The system operates on a Priority Queue and HashMap to manage users on the payment page efficiently. The User Class implements Comparable, enabling the comparison of priority scores, ensuring higher priority for those who experience more failures. To facilitate smooth processing, various Executor Services are employed to systematically drain the queue, ensuring fair and efficient allocation of resources based on user-specific priority criteria.



Data Tier

MySQL, managed as a Docker image during the testing phase. Persistent data and images are stored as Docker volumes. However, as we are nearing production, it has since moved to AWS RDS and S3.



Security Design

The security design involves implementing JWT-based authentication, offering advantages like scalability due to stateless sessions. However, the risk of compromised tokens exists, so short expiration times are used to limit attacker access. Refresh tokens are omitted to prevent indefinite session extensions. Additionally, JWTs are sent via http-only cookies for protection against XSS attacks.

Role-based authorisation also grants varying API access levels. Input validation covers user emails and relevant fields, while passwords are securely salted and hashed using BCryptEncoder in Spring Security, making it difficult for attackers to access user passwords even in the event of database breaches.

On the frontend, we utilise reusable wrapper components to restrict access to authorised users for specific pages. These components validate user access by checking local state for login status or the presence of

a valid JWT in an http-cookie. Unauthorised users are redirected, and sensitive endpoints require a Bearer token with the request to ensure authorised access.

Testing

Testing is done using the JUnit Tests, and unit tests are written to test the behaviour of the backend. Frontend unit tests were also written and executed with Jest. The JaCoCO Package is used to measure test coverage, in order to determine which part of the code the tests executed on. We prioritised writing tests for more complex features, as those are most likely to fail and have edge cases that we did not consider.

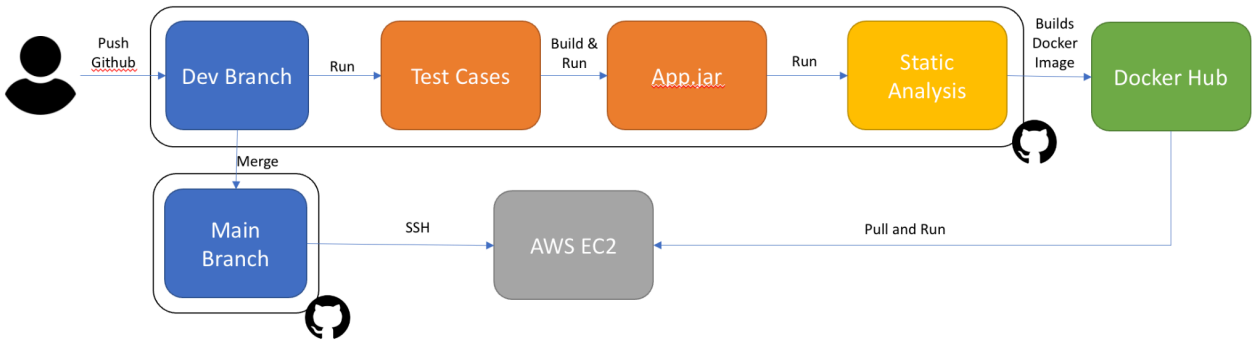
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
bluetix.controller		27%		10%	69	84	152	218	46	61	0	11
bluetix.model		41%		17%	132	190	41	108	85	140	0	10
bluetix.serializable		0%		0%	71	71	20	20	35	35	4	4
bluetix.service		73%		62%	38	109	87	280	23	82	0	12
bluetix.dao.request		32%		0%	40	68	0	14	10	38	0	4
bluetix.exception		1%		0%	26	27	16	17	15	16	4	5
bluetix.dao.response		32%		0%	20	34	0	7	5	19	0	2
bluetix.dto		37%		n/a	27	43	7	23	27	43	1	4
bluetix.security		96%		55%	8	20	1	49	0	10	0	2
bluetix		37%		n/a	1	2	2	3	1	2	0	1
bluetix.config		100%		n/a	0	4	0	11	0	4	0	2
Total	2,716 of 4,893	44%	337 of 404	16%	432	652	326	750	247	450	9	57

Cloud Services

AWS EC2	EC2 instance to host and execute Dockerised containers
AWS RDS	MySQL for efficient scalable database operations
AWS S3	File management service, for storing and retrieving images
AWS CloudFront	CDN for S3, to enhance delivery speed of images
Pinata IPFS	Upload, store and retrieve ticketing NFT metadata

CI/CD Implementation

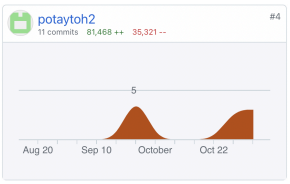
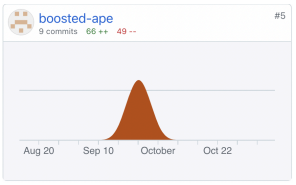
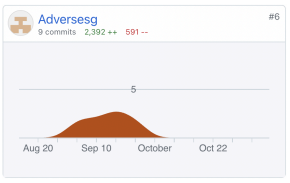
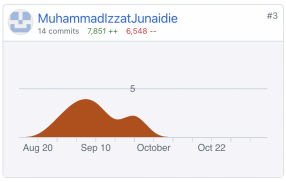
We've implemented a basic CI/CD pipeline, which builds our spring boot application, creates a docker image and pushes it to dockerhub, then ssh's into the EC2 instance and pulls the docker image and runs it in the EC2 instance. All these happen automatically when the dev branch is merged to the main branch via a pull request.



Code Review

Code Review is done whenever a pull request to the main branch, and only upon review by 1 reviewer, will the pull request be merged. The static analysis tool PMD is also used in order to improve the code review process, by recommending code style changes.

Git Commit History

User	Frontend	Backend
Axel - Blockchain		
Boon Jhee -Authentication -Queuing -CI/CD		
Favian -Customers Frontend -Seating		
Izzat -Customers Frontend		
Timothy -Creators Frontend -Generic Backend -Cloud Setup		
Si Wei -Tickets Frontend -Purchasing