Healthcare Behavioral Group Therapy (Group - Code-Blooded)

Project Status Update: In this project we are trying to implement a small **Healthcare Behavioral Group Therapy application** using F# APIs. Our plan is to complete the backend logic first which accounts for the coding part majorly in F# and then we proceed on to the UI part that won't take much time as it would be on Express.js, Currently, we are progressing as expected and almost nearing the completion of our backend logic. In the 1st week of November, the team will start focusing on the frontend logic and we are targeting to complete our project on or before November 15th.

A little something about our project:. The primary objective of this project is to develop a comprehensive application to support group therapy sessions in a secure, compliant and effective manner. Key platforms include:

- An Express js application to handle session scheduling, patient management.
- A business API layer using F# to handle patient data, session management and group therapy workflows.
- A MongoDB database for secure, flexible and scalable patient and session data storage.

Ensure that the application is HIPAA-compliant and meets all necessary security and privacy standards for healthcare applications.

Core API endpoints for patient registration, session scheduling and therapist session management have been implemented and successfully tested in the development environment.

Key Achievements:

- Completed design and implementation of RESTful F# API
- Established authentication and authorization flows
- Unit and integration tests show 95% passing rate

Next Steps:

- Begin integration development of frontend probably through express js
- Conduct initial user acceptance testing with a sample dataset

Risks:

- Data privacy compliance (HIPAA review pending)
- It is a simple project on F#, hence not many risks were encountered until now.

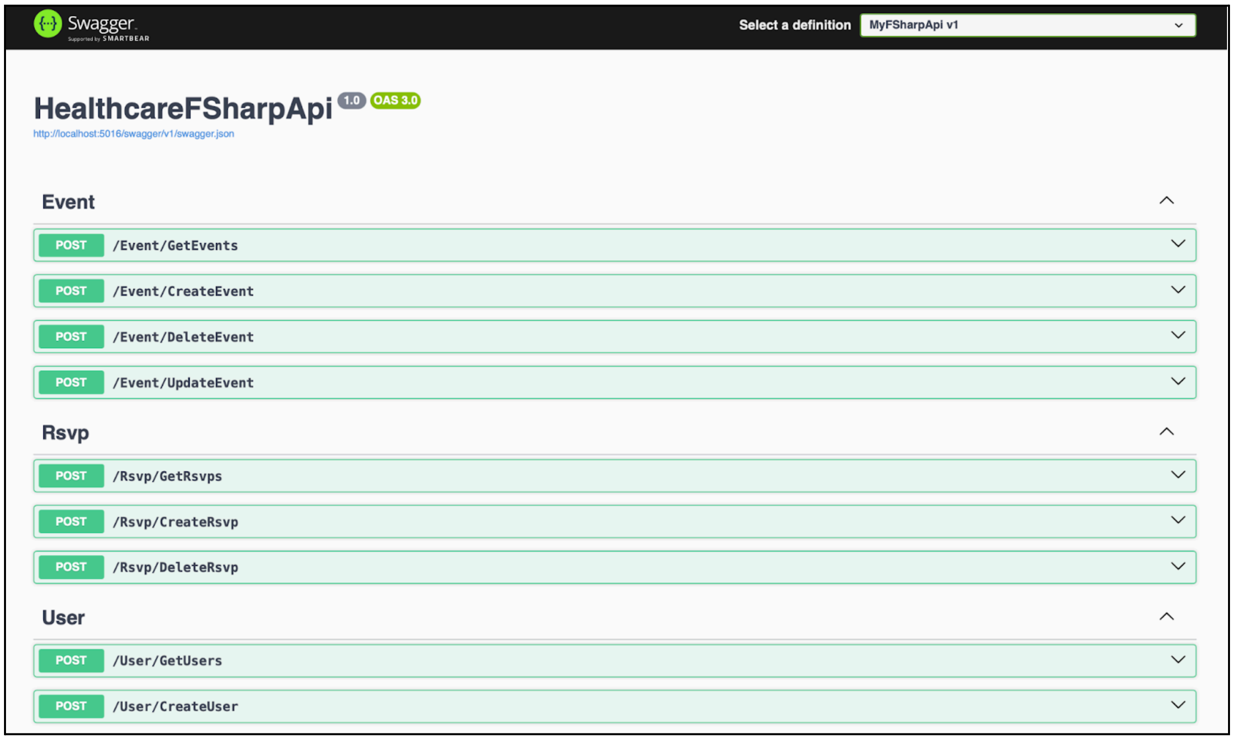
Current status for each of the use cases:

Delivery date	**use cases**	**status**
Monday, October 6, 2025	register new user	completed
Tuesday, October 7, 2025	login / authenticate user	completed
Wednesday, October 8, 2025	get current user profile	completed
Thursday, October 9, 2025	reset user password	completed
Friday, October 10, 2025	list all users	completed
Monday, October 13, 2025	create event / group therapy session create RSVP	completed
Tuesday, October 14, 2025	list all event / group therapy sessions update RSVP	completed

	exam preparation break	
Wednesday, October 22, 2025	get event / group therapy details delete RSVP	completed
Thursday, October 23, 2025	update group therapy session encrypt user credentials	completed
Friday, October 24, 2025	cancel group therapy session	completed
Monday, October 27, 2025	list RSVP for an event / group therapy	completed
Tuesday, October 28, 2025	list RSVP for an user	completed
Wednesday, October 29, 2025	models/new event	completed
Thursday, October 30, 2025	models/Get Event, Update Event and the RSVPs	completed
Friday, October 31, 2025	models/Designing user models	In progress
Tuesday, November 4, 2025	UI view / page to register new user	Currently in the planning phase and majority of work is done, will move to the implementation phase

		soon.
Wednesday, November 5, 2025	UI view / page to login user	Currently in the planning phase and majority of work is done, will move to the implementation phase soon.
Thursday, November 6, 2025	UI default index page of group therapy application.	Currently in the planning phase and majority of work is done, will move to the implementation phase soon.
Friday, November 7, 2025	UI view / page to list group therapies	Currently in the planning phase and majority of work is done, will move to the implementation phase soon.

Screen capture of F# API:



Swagger Supported by SMARTBEAR

Select a definition MyFSharpApi v1

HealthcareFSharpApi 1.0 OAS 3.0

http://localhost:5016/swagger/v1/swagger.json

Event

POST /Event/GetEvents

POST /Event/CreateEvent

POST /Event/DeleteEvent

POST /Event/UpdateEvent

Rsvp

POST /Rsvp/GetRsvps

POST /Rsvp/CreateRsvp

POST /Rsvp/DeleteRsvp

User

POST /User/GetUsers

POST /User/CreateUser

Challenges faced:

One challenge we did not anticipate was ensuring smooth integration between the F# backend APIs and the Express.js frontend. We faced some data format mismatches and dependency issues during testing, which required extra debugging. Adapting to F#'s functional programming style and finding relevant online resources also took longer than expected. While authentication and encryption were implemented successfully, completing the HIPAA compliance review is still in progress. Managing multiple controllers connected to the same MongoDB database occasionally caused model update conflicts. In addition, the tight project timeline added pressure during development and testing phases. Overall, these challenges helped us strengthen our coordination, debugging and time management skills.

In short, all of us helped each other to reach this phase, the challenges were sometimes typical to debug but with patience and each other's knowledge sharing, we managed to overcome a lot.

All 5 of us contributed to F# coding logic in the backend and hence we were able to achieve the target in the expected date as shown above:

Below mentioned are the contributions of each of the team members:
Rucha and Nidhi - They worked on the **EventController** that contains logic to connect to a MongoDB database, performing operations on an event collection. After the GetEvent endpoint is called, it fetches all the event specific records from the database, converts it first to JSON and then returns it successfully to the client. The controller is also responsible for providing endpoints to create, update and delete events, in which each of the functionality logs an action, interacts with the MongoDB collection and then returns a result. The CreateEvent function also performs some preliminary validation and creates a small summary structure to include in its response. Overall, the controller behaves as the API layer that is responsible for communication between the client and the MongoDB database, handling data fetch and modification through HTTP requests.

Sanjyot - She worked on structuring the **UserController** that is responsible for connection to the same MongoDB database and works with the user collection. The GetUsers endpoints reads all the user related data records from the database, converts them to JSON and returns them in the response. The CreateUser method allows a new user to be added in the database by insertion of the already received data. The Login method in this implementation also has an insert operation which means it is somewhat same in behavior as user creation instead of verifying the credentials. Throughout, the logging functionality is used to record whatever the data controller handles. Overall, this controller is responsible for management of user specific data by taking care of simple database interactions through HTTP requests.

Chetan - He worked on the **RsvpController and the data models** that are responsible for managing the RSVP records stored in the MongoDB database. The GetRsvps endpoint fetches all the RSVP entries from the rsvps collection, also responsible for logging the results, converting them to JSON and finally returning them. The CreateRsvp functionality is responsible for insertion of a new RSVP entry into the database. The DeleteRsvp method removes an RSVP on the basis of its id value. Throughout the controller, logging functionality is implemented to track the actions and processed data. Overall, this controller handles simple create, read and delete functionalities for RSVP data.

Sudeepta - She worked on the **data models for Event** that are used to represent the event information when interacting with the MongoDB. Each type of record actually corresponds to a separate stage of working with the data so newEvent is responsible for creation of a new event before it gets an assigned database ID, while the Event corresponds to a fully stored data that includes the MongoDB _id field. The GetEvent type is useful in case of fetching events and includes the host as a string rather than ObjectId, making it easier to display or return in responses. The Update Event type is responsible for updating an event by sending the if and changed fields back to the server. Binary JSON attributes are used to correctly relate the fields between F# objects and MongoDB document structure.

Testing scenario: Since we are currently on track to build our UI, hence distinct testing results cannot be updated in this document, however as mentioned above integration tests have been performed and it has a good percentage of passing rate.

The Github link that contains our current progress is:
https://github.com/sbal2911/SPL_Final_project
