KU Dating App (JayMatch) – Project 3 Plan

Synopsis:	A KU-verified dating web app. Students sign in with @ku.edu, create profiles, get daily match suggestions, and chat when there's a mutual like.
Stack:	Backend: Rust (Axum or Actix) + SQLite. Frontend: Angular. Realtime chat using backend as middle man.
Deploy targets	API: Fly.io or Railway or Render.
	DB: Sqlite
	Frontend: Vercel or Netlify or Cloudflare Pages (Angular static build).
	Domain and TLS: provider of choice with HTTPS.

Team 1 members:

Ibrahim	Lead frontend
Maren	Lead backend
Zack	Scrum Master / Backend / Frontend
Abdullah	Frontend
Nick	Backend
Saurav	Front end

Non Functional Requirements:

NFR1 Security: Argon2 + JWT + HTTPS
NFR2 Performance: suggestions p95 < 300 ms with cache
NFR3 Privacy: only @ku.edu can sign up
NFR4 Usability: mobile-friendly UI

Functional Requirements:

FR1 KU email sign-up + verify
FR2 Login + session
FR3 Create/edit profile (name, year, major, bio, interests, 1 photo)
FR4 See “Daily 10” suggestions
FR5 Like/Pass
FR6 Mutual like → creates a Match
FR7 1-to-1 chat (WebSocket)
FR8 Block/Report

Agile Reference Stories:

Story ID:	**Story Summary:**	**Story Points**	**Story Source:**
1	Designed a basic interactive web page using HTML, CSS, and JavaScript that reacts to user input via buttons. Implemented one button to show an alert message and another to update paragraph text dynamically. Ensured proper functionality and simple UI.	1	EECS 468
2	For a stock data processor written in python, we needed to update the rho value calculations to include the variation parameter. This involved making the change and testing that the calculated output was as expected.	1	Internship
3	Add a requirement to the CICD gitlab stack file to require that all unit tests pass before a merge can be accepted.	1	Internship
4	Created SQL queries to retrieve and aggregate library records using GROUP BY, HAVING, and JOIN clauses. Verified output with sample schema.	2	EECS 447
5	Search through a large python financial repo and replace a legacy table with a newer table with the same data but modern headers. This involved in depth testing to ensure the functionality remained the same after the changes.	2	Internship
6	Wrote function as part of basic minesweeper using PyGame that populates the grid and determines which numbers go in which cells. .	2	EECS 581
7	Developed an interactive web page using HTML, CSS, and JavaScript with four tabs. Each tab updates the displayed text and changes the active button's color when clicked. Ensured UI consistency, proper functionality, and maintainable code.	3	EECS 468
8	Built a configuration file parser in JavaScript for Minesweeper grid generationm reading from JSON input and redering cells dynamically in React.	3	EECS 581
9	C program that reads monthly sales data from a file, generates a monthly report, calculates summary statistics and six-month moving averages, and produces a sales report from highest to lowest. Includes error handling for file reading and data validation.	3	EECS 348
10	Composed a Python program using recursion and backtracking to simulate a blob moving through a city grid. Program handled errors, tracked visited positions, and calculated total “people eaten” while printing the final grid.	5	EECS 268
11	Implemented a multi-page HTML site with interactive content, including a profile, customizable color box, and password form. Used HTML, CSS, JavaScript, and PHP for dynamic updates and clear navigation.	5	EECS 348
12	Trained and compared multiple clasification models (SVM, KNN, Naive Bayes) on the Iris dataset using scikit-learn; computed confusion matrices and accuracy.	5	EECS 658
13	Constructed a C++ template-based doubly linked list class with full iterator support. Implemented insertion, deletion, push/pop, reversal, list appending, and adjacent element swapping. Managed memory safely, handled edge cases, and supported const-correct access. Tested class with multiple scenarios for reliability.	8	EECS 330
14	Developed backend integrations for an internal chatbot that routed IT suport queries to APIs. Implemented data masking for member into and validated JSON responsed.	8	Internship
15	Helped build a device event aggregation API in Python that collected IoT telemetry and stored it in SQL. Added dashboards and Snyk vulnerability scanning.	8	Internship
16	Assisted IT security team in adding JWT-based authentication to internal APIs, implemented data-masking middleware, and tested via Postman.	13	Internship
17	Built an undirected graph class with adjacency lists using MyVector and MyLinkedList. Supported vertex and edge insertion/deletion, integrated existing data structures, and ensured proper memory management.	13	EECS 330
18	Upgrade an ancient angular project from angular version 9 to angular 19. This involved rewriting the entire project because it was so outdated. We had to research new libraries because the old libraries were not updated.	13	Internship

Project Sprints

Requirement ID	**Description**	**Story Points**	**Priority**	**Sprint No.**
1	Create a basic Angular App. It should hold the skeleton of the project and have things added to it.	1	1	
2	Add buttons for login which sends a request to the backend and add validation to reject non-@ku.edu emails. This should provide a MFA code to the KU email to prevent fake accounts.	1	1	
3	Create basic backend in rust for handling API events.	1	1	
4	Allow users to edit their profiles, update pictures, and modify preferences.	1	5	
5	Build backend endpoints to communicate with the database to both store data and retrieve data as the user uses the app.	2	2	
6	The frontend should send API requests when the user likes a profile or sends a message to another user. All messages should go through the backend and stored.	2	3	
7	Custom UI elements designed in a sprite editor so that we do not need to use stock images or stock widgets. This should be for buttons, logos, and other visual assistants.	2	5	
8	handle navigation between pages such as login, profile, chat, and settings. Ensure URL paths and guards are configured properly.	2	6	
9	Set up custom hosting and possible a custom domain so we do not need to host it locally on someone's computer. This should be a long term location that keeps both the frontend and backend alive.	3	6	
10	Add components to frontend such as profiles, photo albums, dating preferences and the ability to interact with other profiles.	3	6	
11	Integrate with other social media platforms for better matchmaking. For example, matchmake with people that are friends of friends or in the same social network.	3	8	
12	Set up sqlite database to store user information and dating profiles. Build tables to store messages, hashed passwords, user information, user preferences, and pictures.	3	2	
13	Test database connections and verify that backend routes (login, profile fetch, message send) correctly read and write data	5	4	
14	Make the frontend look professional with interactive styling, modern themes, and be free of bugs. This task should also encompass making the quality of life changes to the previously bare-bones frontend.	5	3	
15	Matchmaking algorithm to suggest people who match both lifestyle preferences and other requirements set by the user. It should be build on the backend and provide the frontend with user recommendations.	5	5	

16	Integrate a help chatbot that can handle user questions and provide support. We could use the openai api.	8	7		
17	Create a paid plan to allow users to pay to win at dating. This should involve either credit card payments or a shopify like app integration to handle payments. The payments will also need to be stored on the backend.	8	7		
18	Implement Block and Report safety features. Blocking should hide profiles and disable chat across the system. Reports capture reason, free-text notes, and offending message references.	8	4		
19	Ensure the app is fully responsive on mobile and meets accessibility standards. CSS elements should dynamically resize to fit both a PC window and a phone screen.	13	7		
20	Create an onboarding flow to help new users understand app features; include optional tooltips or tutorial modals.	13	5		
21	Add notification badges or alerts when a user receives a new message or like. These should come in the form of push notifications to phone if possible, otherwise as an email.	13	5		
22	Multithread the backend to handle large traffic loads. This could potentially also include a load balancer or distributed server locations.	21	8		
23	Conduct final end to end testing and debugging to make sure all core features (login, profiles, chat, matching) work smoothly	21	3		