# Team JOHTO Project: Part B - Alumni Network AI Dashboard

## 1. Decompose User Stories into Tasks

-User Story 1: Alumni (Brandon Medina Assigned)

User Story: “As an alumnus of CGU, I want to give back by mentoring a student. I want to be able to create an account, log in, and see a dashboard of students I match with that have similar classes I took, or are interested in the industry I work in, or have the same degree concentration as me, or are in my location, so that I can connect with them.”

Task 1 - Define database requirements - Estimated Hours 3

Alumni, Student, Classes Taken, Industry, Degree Concentration, Job Location

Task 2 - Build database schema SQL - Estimated Hours 6

* Alumni - id (PK), first\_name, last\_name, email (unique), password\_hash, graduation\_year, degree, concentration\_id (FK → degree\_concentrations), industry\_id (FK → industries), job\_location\_id (FK → job\_locations), linkedin\_url, created\_at
* Students - id (PK), first\_name, last\_name, email (unique), password\_hash, enrollment\_year, expected\_graduation\_year, degree, concentration\_id (FK → degree\_concentrations), linkedin\_url, created\_at
* Classes - id (PK), course\_code, course\_name
* Junction Tables: alumni\_classes (alumni\_id, class\_id), student\_classes (student\_id, class\_id)
* Industries - id (PK), name
* Degree Concentrations - id (PK), name
* Job Locations - id (PK), city, state, country

Task 3 - Create HTML skeleton & base - Estimated Hours 5

* main.py
* Index.html - landing page
* Base.html - for a shared navbar and css import
* Login.html - login form
* Register.html - registration form for new users
* Dashboard.html - placeholder for alumni dashboard
* Profile.html - placeholder for populating and updating user information

Task 4 - Connect Flask with database - Estimated Hours 6

* Write seed\_db.py to populate synthetic data for testing
* Write empty\_db.py for clearing synthetic data from the database
* Write helper functions to connect to DB
* Test sample queries to pull from DB

Task 5 - Implement login route - Estimated Hours 5

* Create /login route in [main.py](http://main.py)
* Validate email/password input
* Query DB for user email
* Verify password hash
* Store user\_id and role in session
* flash invalid credentials message for failed login
* Build redirect

Task 6 - Implement logout route - Estimated Hours 2

* Create /logout route in main.py.
* Clear session data (session.clear()).
* Redirect the user to /login page.
* Flash message: “You have been logged out.”

Task 7 - Implement registration route - Estimated Hours 5

* Create /register route in main.py.
* Add register.html form (first\_name, last\_name, email, password, role: alumni/student).
* Validate email is unique in DB.
* Hash password before storing.
* Insert new user into users table + appropriate profile table (alumni or students).
* Redirect to login after success.
* Flash message: “Account created, please log in.”

Task 8 - Build profile route - Estimated Hours 4

* Create /profile route in main.py.
* Add profile.html template to display current profile info.
* Fetch profile details from DB based on logged-in user’s role.
* Show update/edit button (link to profile edit page).

Task 9 - Add profile edit constraints - Estimated Hours 4

* Define editable fields:
  + Alumni: industry, concentration, job location, linkedin\_url.
* Create /profile/edit route.
* Validate form inputs (e.g., location must exist in job\_locations table).
* Prevent edits to non-editable fields (e.g., email, user role).
* Update DB with validated fields only.
* Flash confirmation on save.

Task 10 - Create dashboard page - Estimated Hours 6

* Create /dashboard route in main.py.
* Add dashboard.html template.
* Restrict dashboard results by role
* Display matched students profile cards (name, degree, industry, location).

Task 11 - Implement matching system - Estimated Hours 8

* Define match criteria weights (classes > concentration > industry > location).
* Write SQL query to join alumni ↔ students with filters.
* Write find\_student\_matches(alumni\_id) function.
* Return matches sorted by score.
* Integrate matches into dashboard view.
* (Optional) Implement ML recommender model later.

Task 12 - Implement connection feature - Estimated Hours 7

* Create connections table: id, alumni\_id, student\_id, status (pending/accepted/rejected).
* Add “Connect” button on student cards.
* Route /connect/<student\_id> inserts new row into connections with status=pending.
* Student dashboard shows pending requests (/requests).
* Routes for /accept/<connection\_id> and /reject/<connection\_id>.
* Update DB on response.
* Flash confirmation for both sides.

Task 13 - Testing & dataset validation - Estimated Hours 8

* Write unit tests for:
  + Registration (unique email enforcement).
  + Login/Logout session handling.
  + Profile creation/edit constraints.
  + Matching function returns expected results.
  + Connection request lifecycle (send → accept → reject).
* Write integration tests with Flask test client.
* Seed synthetic dataset (5 alumni, 10 students, multiple industries/locations).
* Perform manual UI testing (login → dashboard → connect → accept).

Task 14- Create tasks/issues on JIRA-Estimated Hours 2

* Create a new project on JIRA
* Create a Kanban board for the project
* Create tasks/issues based on the above tasks with dependencies and due dates
* Create user story for the project to track the progress

As of October 1st

| Milestone | Due Date | Required Hours | Weekdays Available | Hours per Day |
| --- | --- | --- | --- | --- |
| 1.0 (Tasks 1-9) | Oct 23 | 40 | 17 | 2.35 |
| 2.0 (Tasks 10-13) | Nov 20 | 29 | 20 | 1.45 |

- US2: Student mentor suggestions → Tasks: Create synthetic student profiles, implement AI matching logic, connect RAG/SQL backend, test queries.

Task 2: PJ, Nihaad

* Define assistant **system prompt** (role, goals, privacy rules, response style) and few-shot examples.
* create service endpoint to return ranked mentors for a logged-in student (name, degree, concentration, industry, location, score, “why this match”)
* create service endpoint to send a connection request (pending status)
* wire the student dashboard: **Find Mentors** button → ranked list; **Connect** button → request sent
* handle login/role checks and friendly error/empty states
* add **test queries** to validate results and the connect flow

User story 3 : Alex(system admin)

User Story: “As a university administrator I want to view alumni data visualizations on a dashboard using synthetic data during development so I can explore and validate how the system will display insights before connecting to real alumni records.”

Task 1 - Setup environment (3-5 hours)

* Create a frontend folder for dashboard code(flask)
* Install required visualization libraries(plotly, dash, matplotlib, and seaborn)
* setup initial dashboard layout with placeholder charts

Task 2 - synthetic data organization 8 hours)

* Obtain synthetic data from developer(Brandon)
* Load synthetic dataset into backend service(flask/fastAPI)
* Create REST API endpoints:
  + List of alumni
  + Return career/industry trends
  + Filter queries
* Add synthetic updates(new alumni records every x seconds ) to simulate real time

Task 3 - Dashboard Visualizations(10 hours)

* Alumni distribution by industry (bar chart/pie chart)
* Geographic alumni spread (map visualization)
* Career path timeline (Linechart/sankey diagram)
* Mentorship opt-in stats (gauge chart or progress bar)
* Engagement trends over time (line chart showing growth of student-alumni connections)

Task 4 - integration & interactivity(10 hours)

* Connect front end charts to backend API instead of static Json
* Add filtering (by major, graduation year, location)
* Add tooltips and drill–downs for interactive exploration
* Implement dashboard tabs(overview, mentorship, career paths, trends)

Task 5 - testing and validation(10 hours)

* Write unit tests for synthetic data generator
* Validate API endpoints return correct filtered data
* Test chart rendering with sample and large datasets
* Check responsiveness

Task 6 - Documentation(10 hours)

* Document how to regenerate synthetic data
* Document how to run the dashboard locally
* Add screenshots of charts in readme

- User Story 4: Jin, Nihaad– Project mgmt → Tasks: Schedule syncs with AI Squared, document progress, manage GitHub repo.

User Story: As a project developer responsible for analytics, I want to integrate alumni and student data into a dashboard prototype so that administrators and alumni can visualize distributions, trends, and mentorship participation during development, ensuring the system’s insights are validated before connecting to real data.

Task 1 – Environment Setup & Structure (4 hrs)

* Create frontend/ folder for dashboard code.
* Set up Flask templates (base.html, dashboard.html).
* Install visualization libraries (plotly, dash, matplotlib, seaborn).

Task 2 – Dashboard Prototype Layout (6 hrs)

* Build dashboard.html structure with placeholder charts.
* Add navbar and tab switching (Overview, Mentorship, Trends).
* Ensure responsiveness in Bootstrap/CSS.

Task 3 – Visualization Components (8 hrs)

* Implement Alumni Industry Distribution (bar/pie).
* Implement Geographic Alumni Spread (map visualization).
* Implement Mentorship Opt-in Stats (gauge/progress bar).

Task 4 – Data Integration (6 hrs)

* Load alumni/student synthetic dataset from /data.
* Create Flask API endpoints for charts.
* Add filtering (graduation year, degree, location).

Task 5 – Documentation & Testing (4 hrs)

* Write README.md instructions (run dashboard, dependencies).
* Add screenshots of charts as proof for Milestone 1.0.
* Use Flask test client to validate routes.

- US: Developer RAG integration → Tasks: Set up RAG pipeline or SQL fallback, connect 10 profiles, validate retrieval quality. (Optional)

## 2. Milestone 1.0 Features

- Search functionality (Admin view).  
- Alumni and student profile creation  
- Analytics dashboard prototype (basic charts).  
- GitHub repo with working functional + test code.

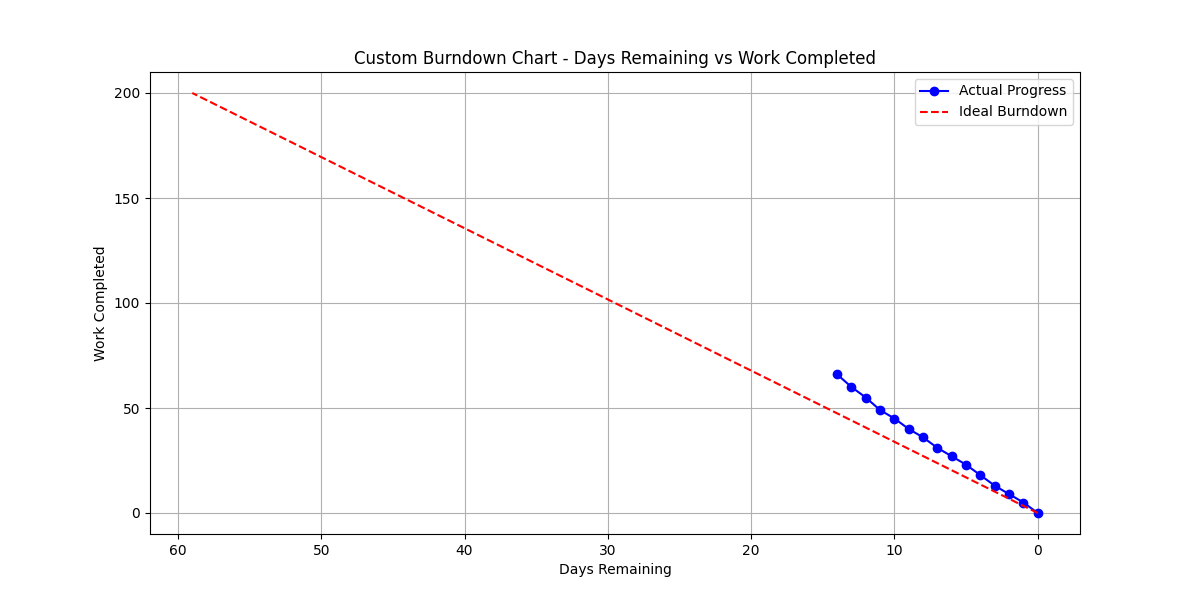
## 3. Iterations for Milestone 1.0

Iteration 1 (30 days,100 hrs total):  
- Database setup  
- API & backend logic  
- Synthetic dataset + alumni/student profiles  
- Basic dashboard integration  
  
Iteration 2 (60 days, 200 hrs total):  
- AI matching logic   
- Dashboard analytics prototype   
- Profile opt-in implementation  
- Testing & refinement  
- Sync with AI Squared  
  
Total work = 200 hrs, completed in 60 days (velocity assumed = 20 hrs/week/team member).

## 4. Task Allocation

- Brandon: Database schema, RAG setup, alumni profiles  
- Alexander: Python backend logic, mentor suggestion engine  
- Haoran: Dashboard integration, analytics prototype  
- Prajwal: API development, AI matching support  
- Nihaad: Schedule syncs with AI Squared, document progress, manage GitHub repo, create synthetic student profiles, implement AI matching logic, connect RAG/SQL backend, test queries

## 5. Burn Down Chart



## 6. Stand-Up Meetings

- Frequency: 2x/week (Tuesdays & Thursdays).  
- Logs/Agenda stored in GitHub repo  
- Example agenda: Progress updates, blockers, task reallocation.  
- Meeting notes uploaded as README files.

## 7. Development & Testing Environment

- GitHub repo initialized with README, requirements.txt, and test folder.  
- Python virtual environment created.  
- Synthetic alumni/student dataset in `/data`.  
- Basic search API and test queries implemented.  
- CI/CD pipeline setup for testing in GitHub Actions.