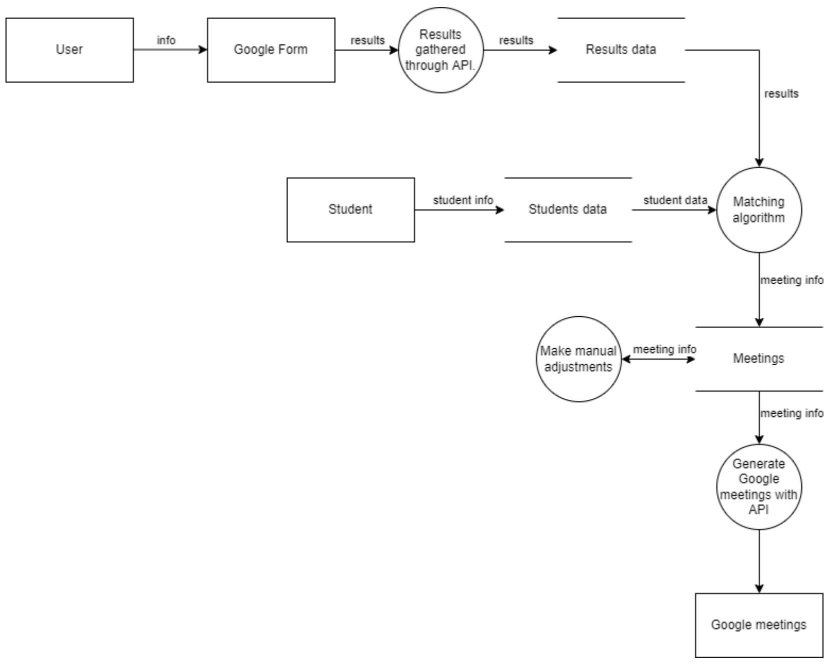
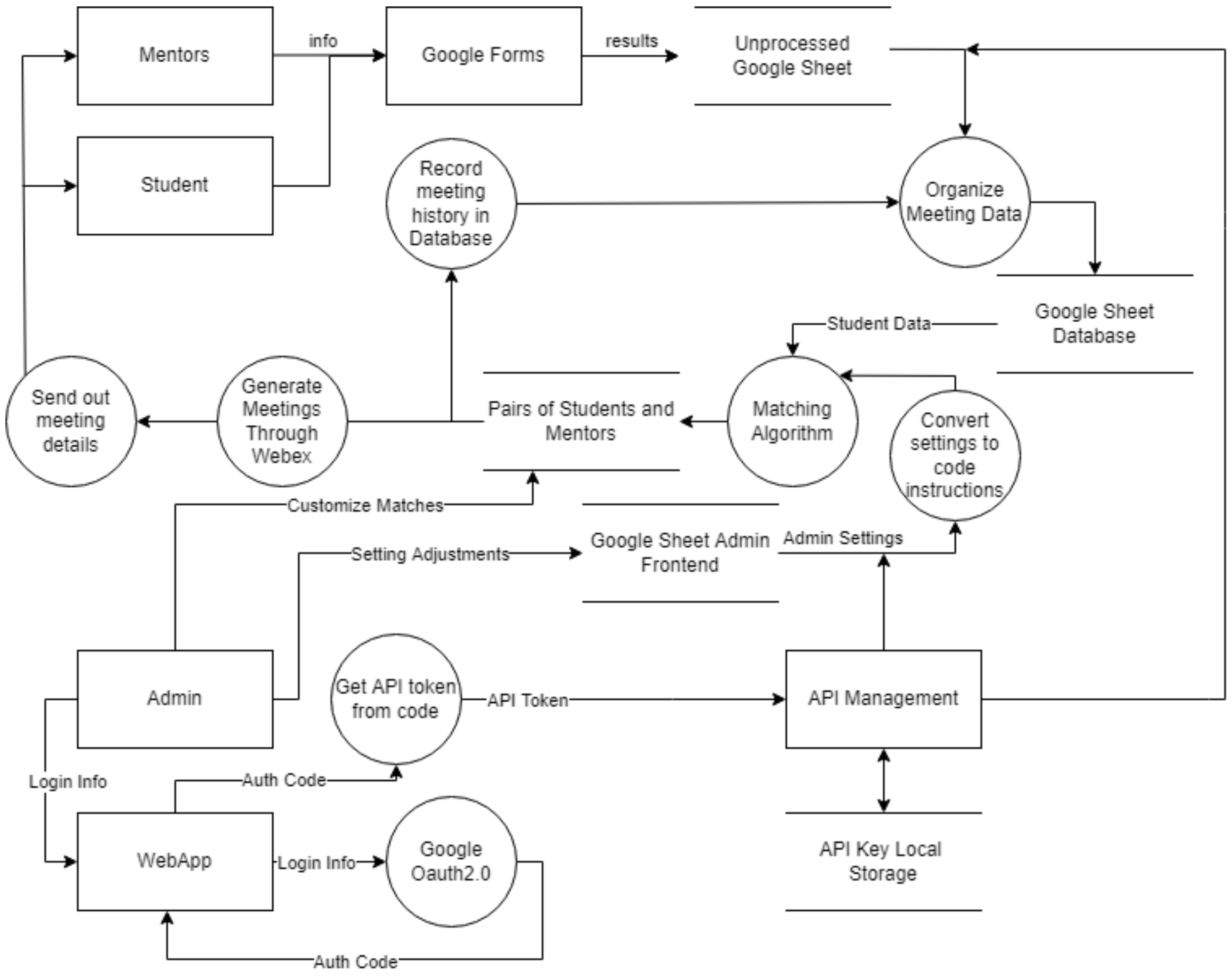
MeetEng Design

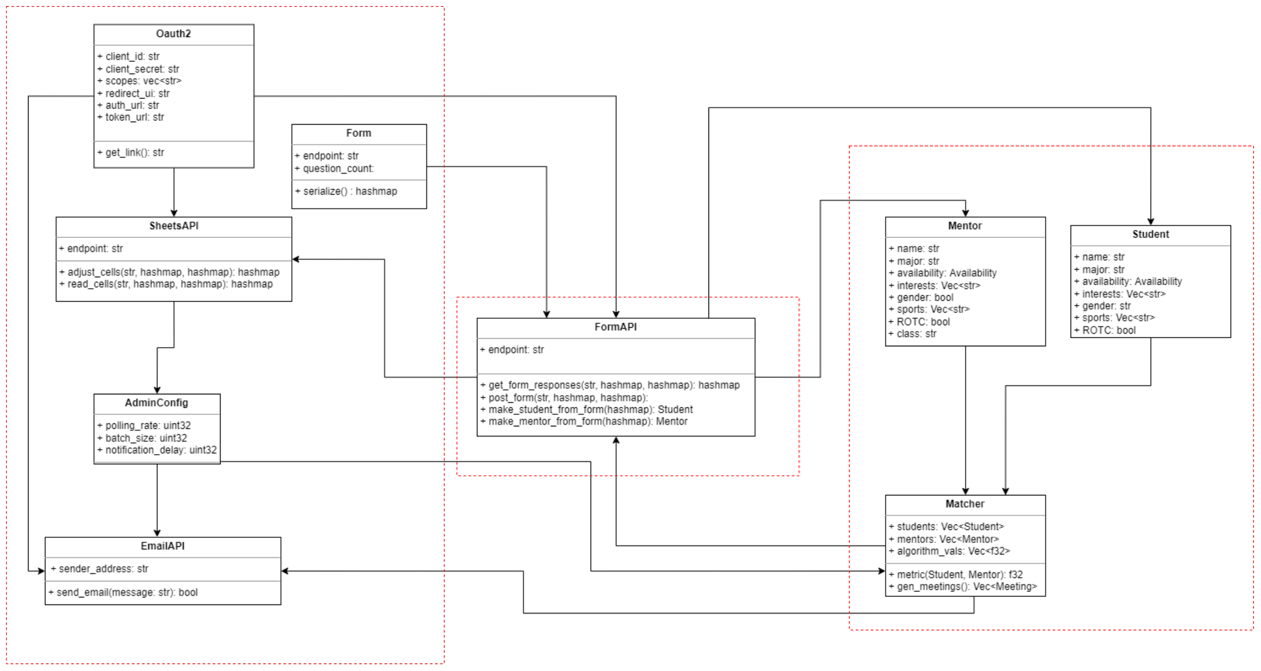
- MeetEng Design Review #1 Minutes: Google Document.
- Original DFD:



- Revised DFD:



- Function Diagram:



- Summary:

  Our goal is a minimum viable product that is simple for users, maintainable for developers, and reliable for the administration. To achieve this, our MeetEng design utilizes Google as a front-end that is controlled by a Rust server, which is connected to Google services through their API.
  The most critical advantage of Google products is that they allow us to create a substantially complete system in the least possible time. The scale required for MeetEng to service the entire target population (an incoming class of engineers) is well within the capabilities of Google, even if we overestimate the number of users at 10,000. The bonus is that the cost to use the Google APIs for our purpose will be practically nothing: the service is free, below two million API calls a month.

  Students are familiar with the Google Suite, often using it for their entire high school careers. We will use Google Forms to gather the necessary information on prospective and current students, such as when they are available for meetings, major, interests, gender, etc. MeetEng then manages this unprocessed data. Our service will transform it into a more readily usable format and store it in a Master Google Sheet. This Master Sheet will serve as the server's database and control panel. The database will store the matching data of currently available students and a history of past and upcoming meetings. The control panel section of the Master Sheet will make use of the interactive features available in the Google Sheets software. The administration can manually adjust meeting times and match pairs with the control panel. When a match is made, an email will be sent out to both the prospective student and current students that includes the meeting time and a WebEx meeting link.

  As mentioned before, Google will be used as the primary front-end (forms and sheets); however, an access point is needed between the user and Google authentication. This access point will be a simple web page developed using Rust web assembly. On this web page, the user will be asked to sign into their Google account and allow MeetEng access to certain scopes of Google Cloud.

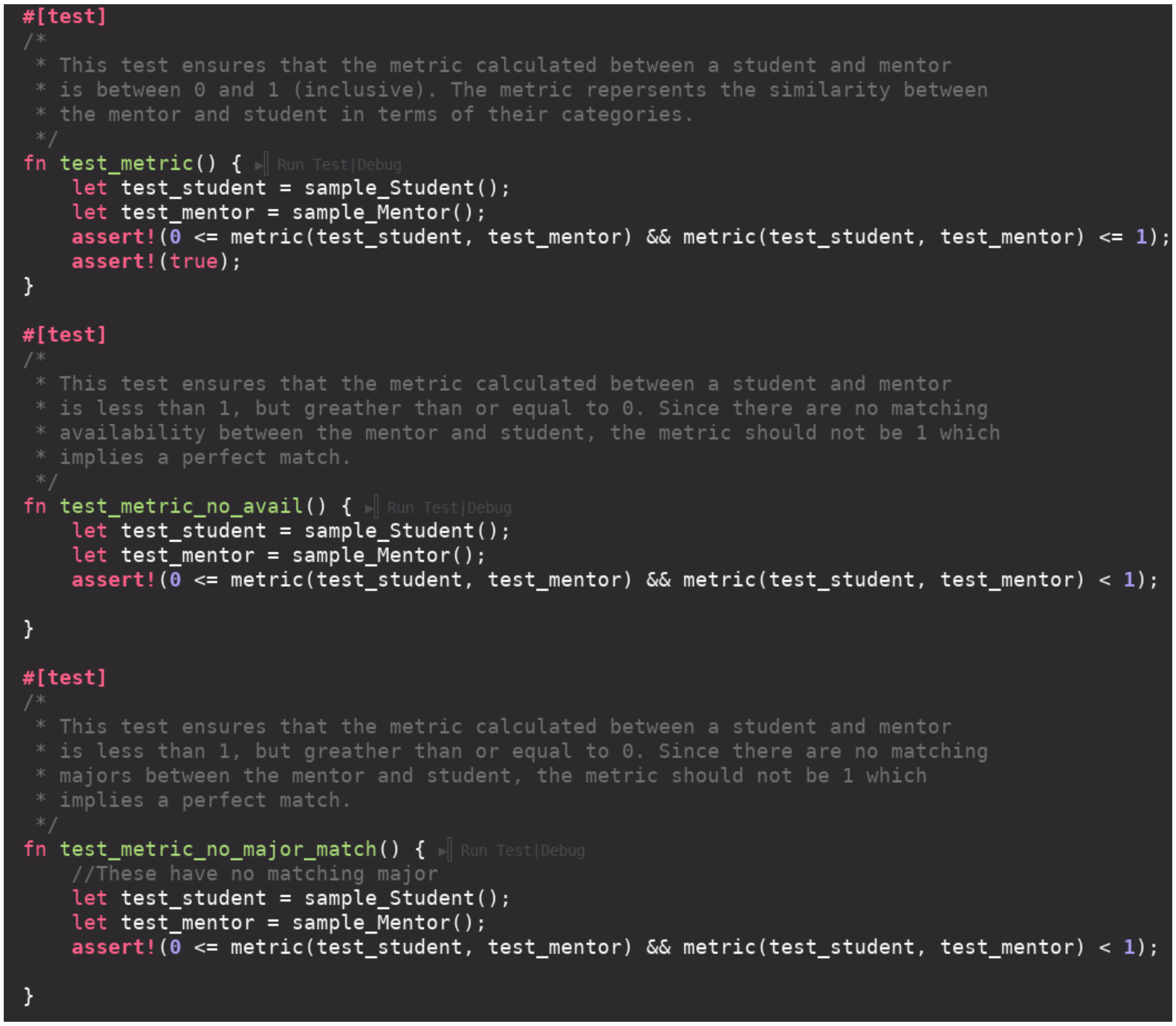
To improve user experience, in the future, this webpage might be further developed to include account logins and a dashboard/console to display matching statistics, matches, and other valuable details.

MeetEng Implementation Plan

- Primary language:
  Rust
- Language use:
  Rust will be used for interacting with Google API and WebEx API to create, read surveys and form a meeting space. Rust will also be used for all internal logic such as the matching algorithm and manual administration adjustments. We currently see no need for any other language.
- Style Guide:
  Follow Rust formatting guidelines documented here and here, which are fairly extensive:
    - Use spaces, not tabs.
    - Each level of indentation must be 4 spaces.
    - The maximum width for a line is 100 characters.
    - Separate items and statements by either zero or one blank lines.
    - Prefer line comments (//) to block comments (/* ... */).
    - There should be a space after ":" and on both sides of "=".
    - CamelCase for "type-level" constructs, snake_case for "value-level" constructs
    - Getters for variable foo should be foo(&self), setters should be set_foo(&self, val)
    - etc.
- Build Tools:
  The project management and build tools required for writing rust are all included in the Cargo program provided by the Rust installation. Cargo includes a configuration file where all dependencies are listed. Cargo will maintain the dependencies across any platform. Additionally, Cargo includes a unit testing framework.

MeetEng Unit Tests

1. metric(Student, Mentor): float



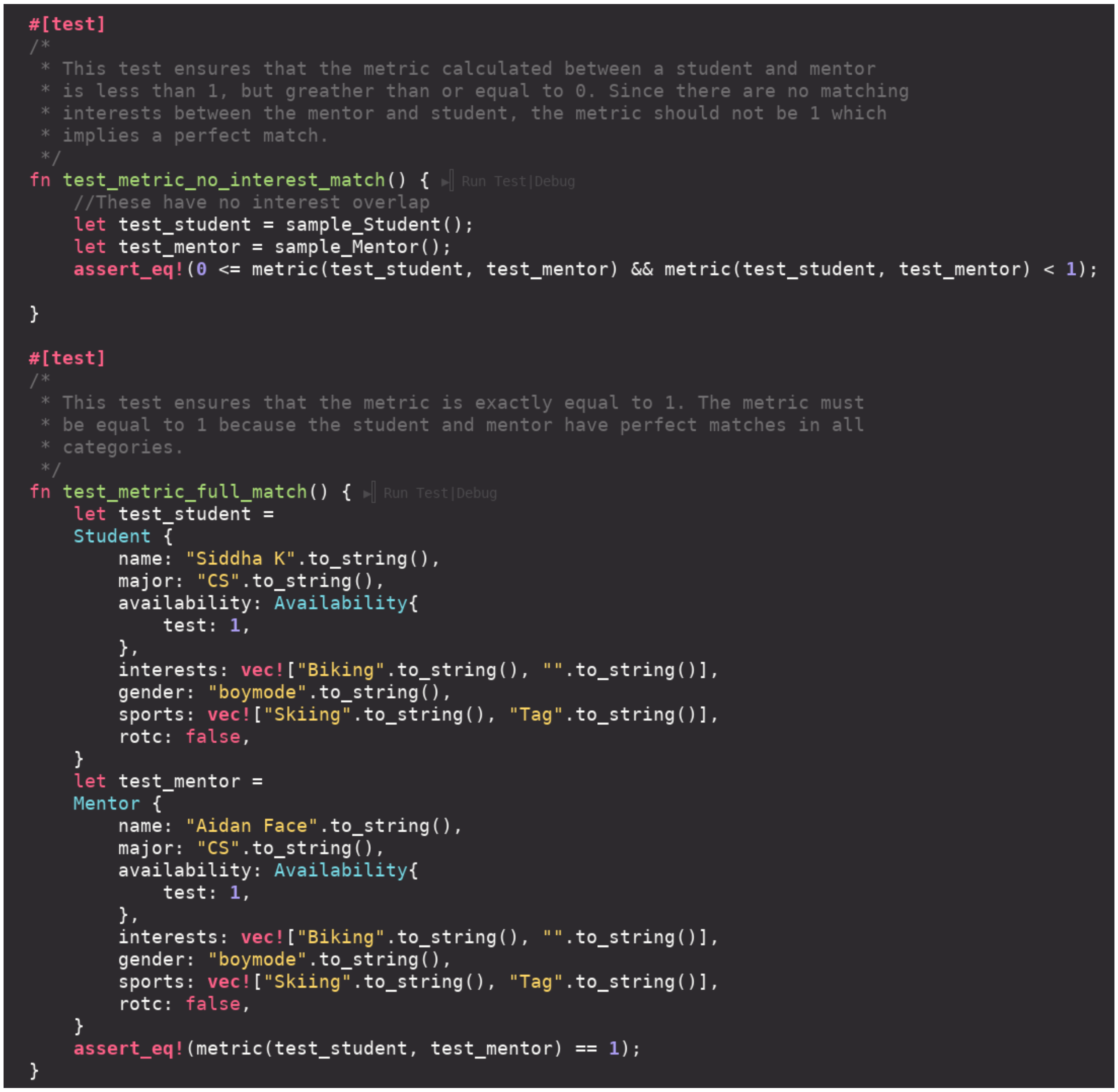
```
#[test]
/*
 * This test ensures that the metric calculated between a student and mentor
 * is between 0 and 1 (inclusive). The metric repersents the similarity between
 * the mentor and student in terms of their categories.
 */
fn test_metric() { ▶ Run Test|Debug
    let test_student = sample_Student();
    let test_mentor = sample_Mentor();
    assert!(0 <= metric(test_student, test_mentor) && metric(test_student, test_mentor) <= 1);
    assert!(true);
}

#[test]
/*
 * This test ensures that the metric calculated between a student and mentor
 * is less than 1, but greather than or equal to 0. Since there are no matching
 * availability between the mentor and student, the metric should not be 1 which
 * implies a perfect match.
 */
fn test_metric_no_avail() { ▶ Run Test|Debug
    let test_student = sample_Student();
    let test_mentor = sample_Mentor();
    assert!(0 <= metric(test_student, test_mentor) && metric(test_student, test_mentor) < 1);

}

#[test]
/*
 * This test ensures that the metric calculated between a student and mentor
 * is less than 1, but greather than or equal to 0. Since there are no matching
 * majors between the mentor and student, the metric should not be 1 which
 * implies a perfect match.
 */
fn test_metric_no_major_match() { ▶ Run Test|Debug
    //These have no matching major
    let test_student = sample_Student();
    let test_mentor = sample_Mentor();
    assert!(0 <= metric(test_student, test_mentor) && metric(test_student, test_mentor) < 1);

}
```

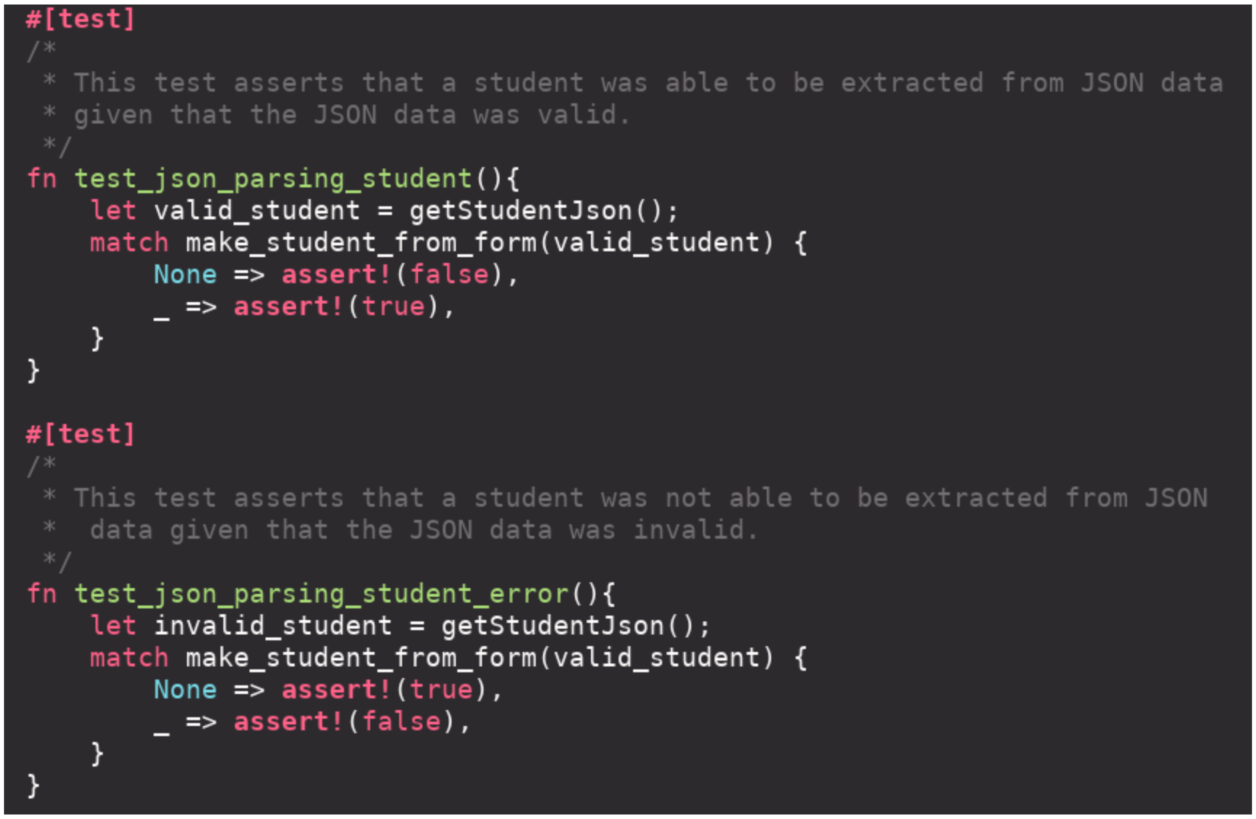


```rust
#[test]
/*
 * This test ensures that the metric calculated between a student and mentor
 * is less than 1, but greather than or equal to 0. Since there are no matching
 * interests between the mentor and student, the metric should not be 1 which
 * implies a perfect match.
 */
fn test_metric_no_interest_match() { Run Test|Debug
    //These have no interest overlap
    let test_student = sample_Student();
    let test_mentor = sample_Mentor();
    assert_eq!(0 <= metric(test_student, test_mentor) && metric(test_student, test_mentor) < 1);

}

#[test]
/*
 * This test ensures that the metric is exactly equal to 1. The metric must
 * be equal to 1 because the student and mentor have perfect matches in all
 * categories.
 */
fn test_metric_full_match() { Run Test|Debug
    let test_student =
    Student {
        name: "Siddha K".to_string(),
        major: "CS".to_string(),
        availability: Availability{
            test: 1,
        },
        interests: vec!["Biking".to_string(), "".to_string()],
        gender: "boymode".to_string(),
        sports: vec!["Skiing".to_string(), "Tag".to_string()],
        rotc: false,
    }
    let test_mentor =
    Mentor {
        name: "Aidan Face".to_string(),
        major: "CS".to_string(),
        availability: Availability{
            test: 1,
        },
        interests: vec!["Biking".to_string(), "".to_string()],
        gender: "boymode".to_string(),
        sports: vec!["Skiing".to_string(), "Tag".to_string()],
        rotc: false,
    }
    assert_eq!(metric(test_student, test_mentor) == 1);
}
```

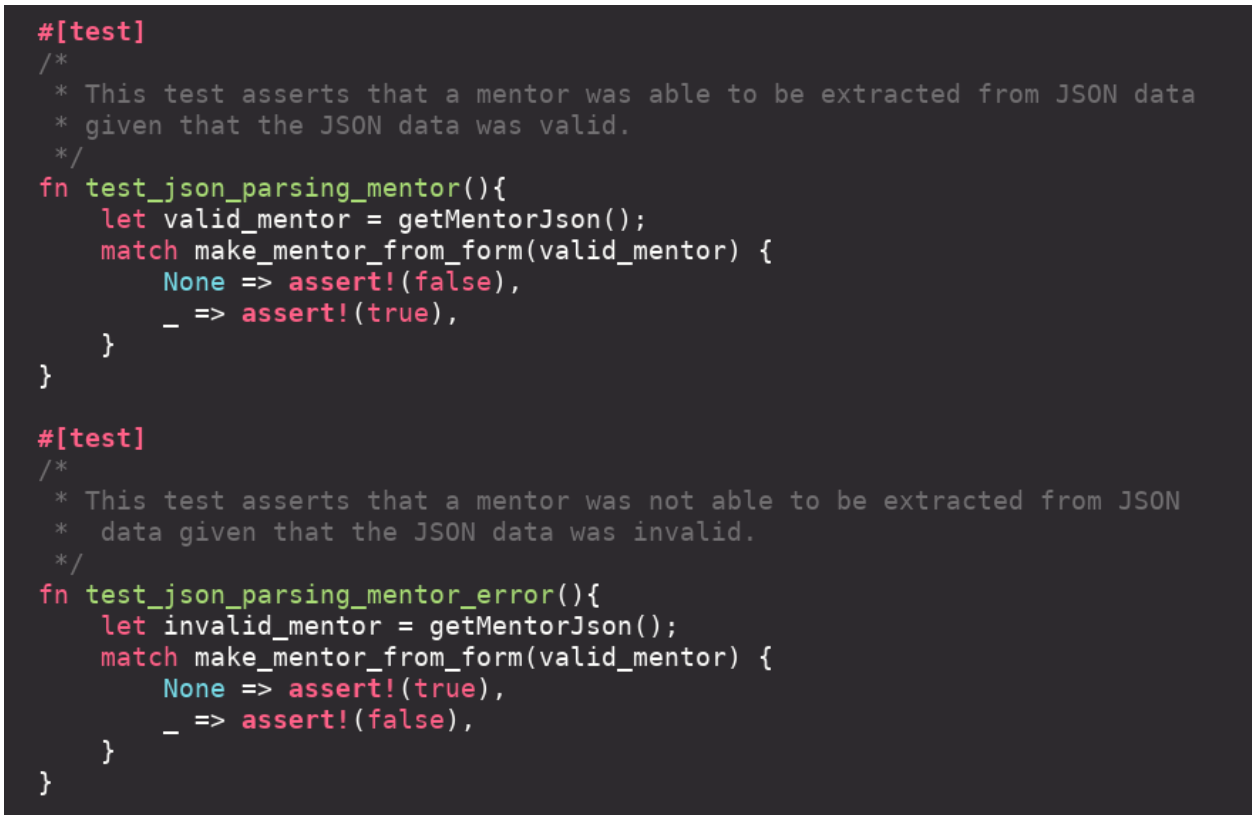
2. make_student_from_form(json): Student



```
#[test]
/*
 * This test asserts that a student was able to be extracted from JSON data
 * given that the JSON data was valid.
 */
fn test_json_parsing_student(){
    let valid_student = getStudentJson();
    match make_student_from_form(valid_student) {
        None => assert!(false),
        _ => assert!(true),
    }
}

#[test]
/*
 * This test asserts that a student was not able to be extracted from JSON
 *  data given that the JSON data was invalid.
 */
fn test_json_parsing_student_error(){
    let invalid_student = getStudentJson();
    match make_student_from_form(valid_student) {
        None => assert!(true),
        _ => assert!(false),
    }
}
```

3. make_mentor_from_form(json): Mentor



```
#[test]
/*
 * This test asserts that a mentor was able to be extracted from JSON data
 * given that the JSON data was valid.
 */
fn test_json_parsing_mentor(){
    let valid_mentor = getMentorJson();
    match make_mentor_from_form(valid_mentor) {
        None => assert!(false),
        _ => assert!(true),
    }
}

#[test]
/*
 * This test asserts that a mentor was not able to be extracted from JSON
 *  data given that the JSON data was invalid.
 */
fn test_json_parsing_mentor_error(){
    let invalid_mentor = getMentorJson();
    match make_mentor_from_form(valid_mentor) {
        None => assert!(true),
        _ => assert!(false),
    }
}
```

MeetEng System Tests

Test Case ID	Associated Requirement ID(s)	Summary	Initial Setup	Steps/Inputs	Expected Outputs	Expected Side Effects	Notes
TEST-1	MATCH-1	Test that a mentor has a match with one student.	One mentor and one student in the spread sheet.	Run the matching algorithm, but don't send emails.	We see the matching		
TEST-2	FORMS-1	Test mentor forms.	No data in the sheet.	Fill out the form with sample upperclassman data	Google form completion page	The database is populated with the sample data	
TEST-3	FORMS-2	Test student forms.	No data in the sheet.	Fill out the form with sample student data	Google form completion page	The database is populated with the sample data	
TEST-4	ADMIN-6	Adding new Admin user	System is initialized.	Share Master sheet with new admin user	None	New Admin can now access data and settings configuration	
TEST-5	**ETHICS-1**	Testing that alternate translation links are functioning.	An alternative form is created with translated text.	Fill out the form with sample data	The form shows text in the desired language	The database is populated with the sample data	
TEST-6	MG-13	Test notification functionality.	A mentor and student with compatible schedules were run through the	Notifcations are sent out	Mentor and student should both receive details on their meeting.	none	

			matching algorithm.				
TEST-7	**SCALE-1**	Test the capacity of our system.	System is initialized	Automatically generate sample data for 200 Mentors and 1000 Students	Meetings should be generated for to maximize the number of served Students	Data on the Mentors, Students, and the generated meetings is stored in the database	
TEST-8	**SCALE-2**	Test the admin capacity of our system.	System is initialized with sample data	Share the master Sheet with 100 sample admins	none	All sample admins have access to master Sheet	
TEST-9	**SCALE-3**	Test meeting notification throughput.	System is initialized with 10,000 sample meetings	Send out notifications	none	Meetings notifications are sent to both participants of all 10k meetings	
TEST-10	**SCALE-4**	API Scaling	None	Make 10000 API requests	none	Data is returned from each API call	
TEST-11	**ETHICS-2**	Matching Region Bias	System is initialized with 1000 Students and Mentors with randomly distrubuted matching factors	Perform matching on sample Data	Of the Students matched last, there should not be any Hometowns over represented	None	We want to ensure Students are not disadvantaged in getting their prefered time slots because they come from an uncommon place.

TEST-12	**ETHICS-3**	Delete Student data	System is initialized with sample data	Performa Matching on sample Data	none	Meetings should be formed. After the meeting has elapsed the Student data should be removed from the database.	
TEST-13	**ETHICS-4**	Delete Mentors data	System is initialized with sample data	Mentor fills out Removal Form	none	The Mentor's data is removed from the database and will no longer be matched with students	
TEST-14	**ETHICS-5**	Purge all data after a year.	System is initialized with sample data marked as more than a year old	Nothing(Periodic checks will occur)	none	All personal data submitted more than a year ago will be deleted.	
TEST-15	MATCH-2	Test that 1 mentor has a max of 4 students.	System is initialized with 1 mentor and 10 Students	Perform matching on sample Data	none	No more than 4 Meetings are generated, no matter the availability of the Mentor and Students	

TEST-16	ADMIN-4	Test that admin can override a match.	System is initialized with sample data	Admin rewrites a meeting entry in the database	none	The update meeting is processed and notifcations are sent	
TEST-17	ADMIN-6.5	Test that additional admins can make changes.	System is initialized	1. Sub-Admins adjust config settings in the control panel 2. Maintainer runs config diagnostic	The diagnostic will show the updated settings input by the Admin	New settings are saved in the control panel	
TEST-18	ADMIN-8	Test that admins can download and view an excel sheet of student info.	System is initialized with sample data	Admin exports the Database Sheet	An excel file is downloaded		
TEST-19	MATCH-5	Test that reminder emails are sent 24 hours prior to meeting time.	System is initialized with sample data	Runs the matching algorithm	none	Reminders are sent out 24hrs in advance of scheduled meetings	
TEST-20	MATCH-6	Test that students are rematched when they have a time conflict.	System is initialized with sample data	Student with meeting scheduled resubmits the information form with updated availabliity	none	Upcoming meetings are adjusted to match the new data.	
TEST-21	UI/UX-6	Test that meeting emails are sent at least 3 days before the meeting time.	System is initialized with sample data	Runs the matching algorithm	none	Reminders are sent out 3hrs in advance of scheduled meetings	

TEST-22	UI/UX-4	Test that all outgoing emails are from SoEhub@rpi.edu.	System is initialized with sample data	Runs the matching algorithm	none	Reminders are sent out from SoEhub@rpi.edu	
TEST-23	UI/UX-3	Test that student and mentor forms include clubs and interests as check boxes.	None	Access Student and Mentor submission forms. Fill out interests and clubs sections	none	Data is recored in the database	
TEST-24	SECUR-1	Test that admins are the only ones that can access the student data.	None	Attempt to access the Database Sheet URL from a non-authorized account	no access	none	
TEST-25	MG-6	Test that there is some form of bot protection.	None	Fill out forms from suspiscious account	none	Suspicious account is flagged	
TEST-26	MG-6	Test that there is some form of bot protection.	None	Repeatedly fill out forms from the same account	none	Many repeat submissions within a short time frame are rejected	
TEST-27	MG-7	Test that there is a admin dashboard where parameters can be changed.	System is initialized	Admin makes changes to control panel parameter	none	New settings are saved in the control panel	
TEST-28	SOCIAL-3	Test that a feedback form is sent after a meeting ends.	One meeting is queued	Wait until the meeting has passed	none	A satisfaction survey is sent to the student	

TEST-29	MG-8	Test that profiles are deleted if a user graduates or leaves.	Stored sample Mentors	Nothing	none	Mentors are removed after a period of time	
TEST-30	MATCH-7	Test that a notification is sent to an admin if an error occurs.	Set of mentors and students such that a student cant find a match	Run matching	none	Error notification is sent to an Admin	
TEST-31	UI/UX-7	Test that in the form time in 30 minutes intervals from 2-7pm weekday can be selected.	No data in the sheet	Fill out the form with for each time slot	Google form completion page	The database is populated with the sample data	
TEST-32	STRETCH-12	Test that the admin panel displays statistics.	History of meetings in the database	Nothing	Statistics are displayed in the control panel	None	