Code Review

Background

Our team is using ChatGPT AI as model to automatically perform code review to changed files every time a pull request is created. The OpenAI model is gpt-4-turbo and it will perform only to Kotlin, JavaScript, CSS, and JSON files.

Along with AI assisted model, our team also has more traditional code review. Each task has assignee and reviewer. Every time pull request is created, assignee must assign a reviewer for their code, and the reviewer must be a different person from the assignee. Both reviewer and assignee must within the same development group, meaning both must be Frontend dev team or Backend dev team.

With the combination of AI assisted code review and traditional code review, our process is more robust, as ChatGPT essentially becomes another reviewer to help review the code.

Note on Sprint 2 Timelines

During Sprint 2, most, if not all, of our code does not involve with ChatGPT, due to the fact that we are unable to use it until May 1st, which during that time we have finished our sprint 2 coding process. An automated code review did occur to 3 files, which is linked below:

https://github.com/COMP90082-2024-SM1/ME-Wombat/pull/68 Connect your Github account

Files Reviewed

Selection Criteria

Backend

- Is an API controller that is accessible to the open internet
- Deals with complex data managing, such as file reading
- Not understood very well - tech areas new to the team

Frontend

- Three seperate files were reviewed for each of the web app and the mobile app
- For each, we evaluate a file that:
  - Handles the rendering of several complex UI components
  - Handles the navigation between screens
  - Handles the API requests to the backend

Timeframe

- Given the issues with GitHub actions being out of resources, we were only able to run the automated code review once
- In future we aim to regularly use the tool to see how sensitive pull requests compare
- Given that we have identified some useful points, we have made action items below which we will implement in the next sprint to address issues

Note

- We encountered issues using the GitHub code review on the frontend files, so they were reviewed directly through ChatGPT with the prompt provided on Ed

Table

*refer to Github for full GPT review details

File	Area	GPT Summary	Takeaways (green text for useful suggestions)	Reviewer
Appointment Controller	Backend	**Summary of Recommendations:** 1. **Add Comments:** Include comments to explain complex logic or business decisions, especially for public APIs. 2. **Refactor Long Lines:** Break long lines into multiple lines for better readability. 3. **Optimize Database Queries:** Modify the `getAppointmentsById` method to perform more efficient database queries rather than fetching all records and filtering them in-memory. 4. **Enhance Input Validation:** Extend validation to cover all aspects of the DTOs to ensure they meet all predefined business rules before processing.	1. We aim to make our code self-documenting rather than using comments, so will not be implementing this recommendation 2. We will attempt to keep lines short in future, for clarity 3. Given the timeline, we will not be able to edit this right now, but have investigated and will update to take the advice in the next sprint 4. We will not be changing the validation strategy at the current time, as it validates all fields we feel are required (AI agent doesn't seem to fully understand how validation works given it is in a different class)	@AndreSoetrisno
User Controller	Backend	**Summary of Recommendations:** - **Add Comments:** Improve documentation within the code to describe the functionality and any complex logic. - **Method Renaming:** Rename methods like `getPatient` to more descriptive names such as `findPatientById`. - **Reduce Long Lines:** Break long lines into multiple lines for better readability. - **Abstract Common Patterns:** Consider abstracting the validation and persistence pattern used in `createPatient` and `createAdmin`. - **Enhance Input Validation:** Implement more comprehensive validation in methods like `createPatient` to ensure robustness.	1. We aim to make our code self-documenting rather than using comments, so will not be implementing this recommendation 2. We will rename this function for clarity 3. We will attempt to keep lines short in future, for clarity 4. We will not be changing the validation strategy at the current time, as it validates all fields we feel are required (AI agent doesn't seem to fully understand how validation works given it is in a different class)	@AndreSoetrisno

		• **Optimize Performance:** Implement pagination in `getAllPatients` to handle large datasets efficiently.	5. We will use abstract patterns for good practice 6. Given the generally small scale of the task, we don't expect pagination to be particularly helpful in improving performance here	
User Mapper	Backend	**Summary and Recommendations:** • **Improve Comments:** Enhance comments to explain complex logic, especially in methods like `mapHtmlPatient`. • **Refactor Duplication:** Consider refactoring the creation of `PatientEntity` in `mapPatientDTO` and `mapHtmlPatient` to reduce code duplication. • **Enhance Input Validation:** Strengthen input validation and error handling to manage different types of input more robustly. • **Break Long Lines:** Improve readability by breaking long lines into shorter ones. • **Use Standard Libraries:** Where possible, use standard libraries for common tasks like date parsing to increase code reliability and reduce potential bugs.	1. We aim to make our code self-documenting rather than using comments, so will not be implementing this recommendation 2. We will not be performing this refactor, as it is important to separate these two functions in our code, as they perform very separate duties. (Adding patients is also a test-only endpoint in the final product, so we want this to not impact the rest of the functionality) 3. We will not be changing the validation strategy at the current time, as it validates all fields we feel are required (AI agent doesn't seem to fully understand how validation works given it is in a different class) 4. We will attempt to keep lines short in future, for clarity 5. We will move the method call discussed in the full GPT results (stringToInstant) to an inbuilt standard library for robustness	@Shaolong Xu
File Service	Backend	**Summary and Recommendations:** • **Refactor duplicated code** for parsing HTML tables into a separate method to reduce duplication and improve maintainability. • **Improve variable naming** for collections to reflect that they are plural. • **Add error handling and input validation** to manage malformed HTML and ensure robustness. • **Address the TODO comment** to handle unlinked appointments, potentially expanding on what specific conditions need to be checked. • Consider **performance optimizations** if handling large datasets, possibly by streamlining HTML parsing and object mapping.	1. We will not be performing this refactor immediately, as it is important to separate these two functions in our code, as they perform very separate duties. (we are aware of this note and are already considering this change as we go) 2. We will fix the variable names mentioned in the full GPT report 3. We will not be changing the validation strategy at the current time, as it validates all fields we feel are required (AI agent doesn't seem to fully understand how validation works given it is in a different class) 4. We will address the todo in sprint 3	@Jonathan Latti

			5. We will generally discuss optimisations for html parsing in next spring also	
Security Configuration	Backend	**Summary of Recommendations:** 1. **Expand Comments:** Provide more detailed comments explaining the security choices and configurations, especially related to role-based access controls. 2. **Refactor Long Lines:** Break long lines of code into multiple lines to enhance readability and maintainability. 3. **Validate User Input:** Ensure that there is adequate user input validation elsewhere in the application to complement the security configurations set here.	1. We aim to make our code self-documenting rather than using comments, so will not be implementing this recommendation 2. We will attempt to keep lines short in future, for clarity 3. We will not be changing the validation strategy at the current time, as it validates all fields we feel are required (AI agent doesn't seem to fully understand how validation works given it is in a different class)	@Jonathan Latti
Home Screen	Front end (mobile app)	**Summary of Recommendations:** 1. Add comments to explain complex logic or non-obvious decisions, especially in functions like `showAppointmentDetails`, `closeAppointmentModal`, and `handleMessagePress`. 2. Refactor the duplication in the "Show More" functionality into a reusable function to enhance maintainability. 3. Consider implementing user input validation to ensure data integrity and security. 4. Evaluate performance optimizations, especially for handling large datasets like upcoming appointments or recent messages. This could involve techniques like pagination or lazy loading.	1. We aim to make our code self-documenting rather than using unnecessary comments, so will not be implementing this recommendation 2. We will refactor this for good practice as we believe this function can also be used on other screens 3. The only user input comes from button pressing, the validation of this is abstracted by react-native so we feel this point is unnecessary 4. In the scope of this project we do not anticipate upcoming appointments or recent messages to ever be large enough datasets to justify these optimisations	@Sophie von Doussa
Tab Navigator	Front end (mobile app)	**Summary of Recommendations:** 1. Add comments to clarify the purpose and functionality of different sections, especially related to navigation and route handling. 2. Refactor the repetition in setting initial parameters for each screen to reduce redundancy and improve maintainability. 3. Consider implementing input validation for user-provided route parameters (`userId`) to ensure data integrity and security.	1. We aim to make our code self-documenting rather than using unnecessary comments, so will not be implementing this recommendation 2. Each screen requires unique parameters, and we would like to maintain flexibility to change this further, so we do not implement this point 3. Validation of the userId parameter is handled during login	@Sophie von Doussa
use Facilities Details	Front end(mobile app)	**Summary of Recommendations:** 1. Add comments to explain the purpose of the hook, its parameters (if any), and how it should be used by consuming components.	1. We aim to make our code self-documenting rather than using unnecessary comments, so will not	@Sophie von Doussa

		2. Consider adding error handling for cases where the API call fails due to network issues or other reasons. 3. Depending on performance requirements, consider implementing caching mechanisms or optimizations to reduce unnecessary API calls.	be implementing this recommendation 2. We currently catch all errors in a single call, which we are satisfied with. However, we remain open to updating this approach if we identify a need for the backend to send more specific error information. Where possible, we do not want to display errors to the patients as this could lead to confusion 3. We have not yet implemented caching mechanisms, but we do currently call the API requests at the head of the navigation components (i.e, in the Resource List screen). This approach ensures that the data is fetched when the user navigates to the resources section, rather than re-calling the API each time the user selects a different facility.	
Doctors	Front end (web app)	**Summary of Recommendations:** 1. Add comments to explain the purpose of each function, especially those related to handling user interactions and API requests. 2. Refactor the filtering and sorting logic into a reusable function to reduce redundancy and improve maintainability. 3. Improve input validation for user-provided filter values and counts to prevent unexpected behavior. 4. Enhance error handling to provide better feedback to users when API requests fail or when invalid data is provided.	1. We aim to make our code self-documenting rather than using unnecessary comments, so will not be implementing this recommendation 2. We will refactor this functionality and make it reusable as similar logic is used in several other screens 3. We plan to implement validation for the address field to ensure it is compatible with the users native maps app 4. API request fails are handled in the API utility function files	@Sophie von Doussa

FileUploaad	Front end (web app)	**Summary of Recommendations:** 1. Add comments to explain the purpose of each function, especially those related to handling user interactions and file upload logic. 2. Refactor repeated logic into reusable functions to reduce redundancy and improve maintainability. 3. Consider adding additional error handling for edge cases, such as network failures or server errors during file upload. 4. Enhance the user interface with clearer instructions or visual feedback during file selection and upload progress.	1. We aim to make our code self-documenting rather than using unnecessary comments, so will not be implementing this recommendation 2. The only duplication we identify is in the button logic, where each (of two) buttons require seperate names and associated checks and functions so we doo not feel that we should refactor this 3. The API request fails are handled in the API utility function files. We do plan to implement more robust error messages initiated from the backend in the case of incorrect file upload format or other processing issues 4. We believe the visual feedback is sufficient. Instructions for how to upload a file are displayed, along with error, success and progress messages where relevant	@Sophie von Doussa
doctorsAPI	Front end (web app)	**Summary of Recommendations:** 1. Add comments to explain the purpose of each function and the overall flow of operations. 2. Refactor repeated logic for retrieving JWT tokens and setting headers into a separate function to reduce redundancy and improve maintainability. 3. Consider enhancing error handling to handle specific edge cases more explicitly and provide better feedback to users in case of failures.	1. We aim to make our code self-documenting rather than using unnecessary comments, so will not be implementing this recommendation 2. We will refactor this repeated code into a reusable function 3. We plan to implement more robust error messages for the webapp	@Sophie von Doussa

✅ Action Items

- [ ] Look at refactoring long lines to multiple lines in backend implementation
- [ ] Write custom JPA for get all appointments by id instead of filtering
- [ ] Rename getPatient to findPatientById
- [ ] Use abstract patterns for messages in controllers
- [ ] Use standard library for stringToInstant
- [ ] Use plural name for Ids in file service
- [ ] Refactor repeated code blocks in frontend files, specifically HomeScreen, DoctorsScreen and API calls
- [ ] Implement more robust error messages for API calls in webapp (e.g. network errors)
- [ ] Implement more robust error messages in the case of incorrect file upload format/type on the webapp File Upload page
- [ ] Implement input validation of the address field on the Doctors webapp page