Sprint 3 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.

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 the mobile app
- 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

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
Notification Controller	Backend	**Summary of Recommended Changes** 1. **Add Comments:** Include comments to describe the purpose of each method and any complex logic within the methods. 2. **Improve Input Validation:** Implement additional validation for `NotificationRequest` and `DeviceTokenRequest` to ensure all necessary fields are present and valid. 3. **Standardise Error Handling:** Ensure consistent error handling across methods, particularly in the `registerDeviceToken` and `getDeviceToken` methods. 4. **Specific Naming:** Consider renaming `request` to a more specific name like `notificationRequest` and `deviceTokenRequest` to enhance clarity. 5. **Consistent Use of Annotations:** Review the use of `@Autowired` and consider constructor injection for consistency and testability.	1. We will add comments to explain the purpose of each method and important blocks of code in the upcoming sprint. 2. We will implement additional validation for `NotificationRequest` and `DeviceTokenRequest` to ensure all necessary fields are present and valid in the next iteration. 3. We will ensure consistent error handling across methods, particularly in the `registerDeviceToken` and `getDeviceToken` methods, in the upcoming release. 4. We will rename `request` to `notificationRequest` and `deviceTokenRequest` in the next code refactor session to enhance clarity. 5. We will review the use of `@Autowired` and consider constructor injection for consistency and testability in the next code review.	@Andre Soetrisno
Message Controller	Backend	**Summary of Recommended Changes** 1. **Add Comments:** ◦ Include comments to describe the purpose of each method and any complex logic within the methods. 2. **Improve Input Validation:** ◦ Implement additional validation for `messageRequest` in the `sendMessage` method to ensure all necessary fields are present and valid. 3. **Consider Pagination:** ◦ For `getMessagesByPatientId`, consider adding pagination to handle a large number of messages efficiently. 4. **Specific Naming:** ◦ Consider renaming `messageRequest` to a more specific name like `messageDTO` to enhance clarity.	1. We will add comments to explain the purpose of each method and important blocks of code in the upcoming sprint. 2. We will implement additional validation for `messageRequest` in the `sendMessage` method to ensure all necessary fields are present and valid in the next iteration. 3. We will consider adding pagination for `getMessagesByPatientId` to handle a large number of messages efficiently in the upcoming release. 4. We will rename `messageRequest` to `messageDTO` in the next code refactor session to enhance clarity.	@Andre Soetrisno
Email Service	Backend	**Summary of Recommended Changes** 1. **Add Comments:**	1. We will add comments to explain the purpose of each method and	@Jonathan Latti

		- Include comments to explain the purpose of methods and significant code blocks. 2. **Method Renaming:** - Rename `getPatients` to `findPatientsByEmail` for clarity. 3. **Refactor Duplication:** - Extract the passcode generation and persistence logic into a separate method to avoid code duplication. 4. **Simplify Complex Lines:** - Break down complex lines in `sendActivateRecoverEmail` for improved readability. 5. **Email Validation:** - Add validation for the email format to ensure valid input. 6. **Batch Processing:** - Consider implementing batch processing for email sending to handle larger volumes efficiently.	important blocks of code in the upcoming sprint. 2. We will rename `getPatients` to `findPatientsByEmail` in the next code refactor session. 3. The passcode generation and persistence logic will be extracted into a separate method in the next refactoring cycle. 4. We will break down complex lines in `sendActivateRecoverEmail` in the next sprint to improve readability. 5. Email format validation will be added in the next development cycle to ensure valid input handling. 6. We will discuss and potentially implement batch processing for email sending in the next sprint to improve performance and scalability.	
AppointmentDetailScreen	Front end - Mobile app	**Summary of Recommendations:** 1. **Add comments** to explain complex logic and non-obvious functionality, particularly in the `handleSaveNotes` function and conditional rendering sections. 2. **Break down long lines**, especially in the `styles` object, to improve code readability. 3. **Remove the commented-out code** related to document functionality and status badges if it's not needed. 4. **Extract common functionality**, such as the rendering of "Add Notes" and "Update Notes" buttons, into separate components or functions to reduce duplication. 5. Implement **user input validation and error handling** to ensure the app can handle invalid or unexpected user inputs gracefully.	1. We will add comments to explain the purpose of each method and important blocks of code in the upcoming sprint. 2. We will attempt to keep lines short in future, for clarity. 3. We will attempt to remove the commented-out code if it's not needed. 4. We will try to extract common functionality into separate components or functions and will update to take the advice in the next sprint. 5. We will attempt to apply more robust error handling strategy in the future and validate all fields we feel are required.	@Sophie von Doussa
NotificationService	Front end - Mobile app	**Summary of Recommendations:** 1. **Remove console.log statements** if they are no longer needed for debugging purposes. 2. **Add comments** to explain the purpose of certain code blocks, especially in the `registerForPushNotificationsAsync` function, to enhance code readability and maintainability. 3. **Break down long lines**, particularly in the `registerForPushNotificationsAsync` function, to	1. We will add comments to explain the purpose of each method and important blocks of code in the upcoming sprint. 2. We will attempt to keep lines short in future, for clarity. 3. We will be able to remove the commented-out console.log statements.	@Yuan

		improve code readability.	4. We believe the current error handling is sufficient and will not be adding more user input validation at this time.	
Calendar	Front end - Mobile app	**Summary of Recommendations:** 1. **Add comments** to describe the component's responsibilities and any important logic to enhance code readability and maintainability. 2. **Break down long lines**, particularly in the `markedDates` object and the `theme` prop, to improve code readability. 3. Consider **adding prop type validation** using PropTypes or TypeScript to ensure the correct types of props are passed to the component.	1. We will add comments to explain the purpose of the component and any important logic in the upcoming sprint. 2. We will break down long lines in the next iteration to improve code readability. 3. We will consider adding prop type validation in a future to make more robust error handling. 4. We will not add any more input validation at this time as it will be handled by the parent component.	`@Sophie von Doussa`

✅ Action Items

- [ ] Explain the purpose of each method and complex logic across all components.
- [ ] Implement additional validation for `NotificationRequest`, `DeviceTokenRequest`, and `messageRequest` in `sendMessage`.
- [ ] Ensure consistent error handling, particularly in `registerDeviceToken` and `getDeviceToken`.
- [ ] Use specific names like `notificationRequest`, `deviceTokenRequest`, `messageDTO`, and `findPatientsByEmail` for clarity.
- [ ] Review `@Autowired` usage and consider constructor injection for consistency and testability.
- [ ] Extract common functionality, such as passcode generation and button rendering, into separate methods or components.
- [ ] Improve readability by shortening long lines in all components.
- [ ] Eliminate commented-out and unused code.
- [ ] Add email format validation and consider batch processing for large volumes.
- [ ] Implement pagination for handling large data sets, like messages by patient ID.