1. File & Project Structure

- Folder names: PascalCase
- File names: PascalCase
- Package organization: Use the following layers: `presentation`, `logic`, `persistence`, `object`. Subpackages for specific categories of implementations are allowed. Tests will have the same layers

---

2. Naming Conventions

- Class names: PascalCase
- Variable and method names: camelCase
- Constants: ALL_CAPS_WITH_UNDERSCORES
- Abbreviations are allowed, but not prefered

---

3. Formatting & Style

- Tabs will be used to create all indentation
- Line length limit is 120 characters
- Braces go on a new line after method/if statement
- Do not use braces for single-lined if statements
- There must be one blank line between methods

---

4. Commenting Practices

- Required class/method headers
- Inline comments for complex logic only (variable names should be self-explanatory)
- For single-line comments, always use //
- For multi-line comments, both /* */ and // are accepted (former preferred)

---

5. Code Organization

- Order of class members:
    - constants > fields > constructor > public methods > private methods
- Static variables should come first, then instance variables
- Only make methods public if they will be called by other classes

- Variables should not be public and should have getters/setters

---

6. Error Handling

- Only catch an exception if it is an expected behaviour from a method that we have not written (i.e. from a package of some kind), otherwise throw the exception (unless you are in UI with nowhere to throw it, in which case you may catch and display the exception)
- Always use custom exceptions where applicable, with the format “<detailedTitleOfWhatWentWrong>” (this may change in the future; we accept this technical debt)

---

7. Version Control Practices

- Branch names: cooked/<camelCase> (should be relevant and clear to what the purpose is)
- Your commit comments should start with a single-sentence explanation of the change made in the commit. This should then (optionally) be followed by a line break followed by any additional feature details in list form, then finally followed by another line break then a (optional) developer notes section with any technical information that needs to be known
    - Ex.:
    - Added ability to filter searches by allergies
    -
    - - Can filter for nut, dairy, and seafood allergies
    - - Has the ability to filter for any number of allergies at once
    -
    - Dev Notes:
    - - Added filter menu to allow for additional filter options
- Push to the dev branch after every completed commit (pull first) and submit a merge request if necessary
- Merge requests are to be completed by the person they are relevant to if possible (the people whose code are having conflicts)
- Every commit must go through a code review by one person who did not make the commit. Any recommended changes should be sent to the person who made the initial commit to complete

---

8. Testing Standards

- Unit test coverage must meet 80% overall

- Tests and test methods should follow the standard naming conventions, and should be of the form “test<ClassName><SpecificsOfTest>” (e.g. testIngredientSearchbarHistory)
- Test files should be organized based on the package organization (e.g. all persistence tests should be in their own directory)
- Tests should work with the built in data in the persistence data

---

9. Dependency Management

- All libraries and frameworks are allowed, as long as they work in an offline setting
- Do not use external APIs
- We will all use JUnit (must work with JUnit 5) for testing

---

10. Collaboration & Workflow Norms

- If you are working as a pair on some section of the code, you should designate which parts of the code each of you will be doing, and work on different files if possible (to avoid merge conflicts)