Team 14
Adrian Chang (adrianlc)
Xuanye Li (xuanyel)
Henry Lin (hlin1)
Muhammad Yusuf (muhammay)

API Design Final Project: Spoontacular

For our final project, we decided to fix the Spoonacular API, which allows users to search for and get information about recipes. The result of our work is Spoontacular, a Java library that allows users with a Spoonacular API key to easily search for and analyze recipes.

Design and Development Process

We began by individually brainstorming use cases we would personally run into when using our API. Because users of this API would be anyone who cooks and we all cook, we felt this would be sufficient to get a solid scope of user behavior to start designing.

We then collectively determined which use cases we wanted to support for our first version of the API, which would be best left for future versions, and which were out of scope of the project entirely. Once we had our target use cases, we collectively laid out requirements for our first version and decided on the shape of our API.

We then split up the design work for the classes and enums and collectively reviewed the initial designs. We then shuffled the classes around and each implemented a different set of classes from the ones we designed. While implementing, we iterated on the class designs where necessary.

We then split up the work of testing (both unit testing and writing client code against the API), debugging, and documenting our API. Once we had a working implementation, we revisited some design decisions we made and adjusted our design appropriately. We then sought out external feedback, determined which feedback we ought to address in the first version of our API, and made some major adjustments accordingly.

Work Attribution

We generated requirements and made major design decisions by committee in group meetings. The majority of implementation and testing was done individually, with the team checking in and providing input regularly. We then split up work on the report and presentation, which we then reviewed and finalized as a group.

Person	Attribution
Adrian	● Implemented the Parser, Restful API requester helper and Restful API unit test ● Designed and drafted the prototype of RecipeSearcher ● Fixed minor errors and make sure the code base is always compilable ● Interviewed with friends to gather feedbacks for V1 ● Wrote and presented the Example Use Case, Requirement and Solution slides of the presentation
Muhammad	● Designed Recipe class and Diet, Intolerance, Cuisine, and Nutrient enums (w/ Henry) ● Implemented Recipe, RecipeSearcher, and RecipeSearcher.Builder classes ● Wrote sample client code ● Presented live demonstration with explanation ● Authored initial report draft
Xuan	● Designed the Equipment, Ingredient, Step, and Amount classes ● Implemented the Nutrient enum, Step, and Equipment classes ● Wrote sample client code ● Interviewed a friend to gather more feedback for V1 ● Wrote and presented the Evaluation and Future Work section of the presentation
Henry	● Designed Recipe class and Diet, Intolerance, Cuisine, and Nutrient enums (w/ Muhammad) ● Implemented and took lead in the Amount and Ingredient classes ● Wrote the initial API javadoc ● Authored the design rational and requirements documentation ● Wrote and presented the Introduction and API Flaws slides of the presentation

Challenges and Lessons

More than anything else, we learned how much iteration it takes to go from an initial API design to a final implementation that actually meets the goals set at the beginning of the process. While we were thoughtful with our initial design and it laid a solid foundation for our work, there were many flaws and missteps in our original design that weren't revealed until well into the implementation process. Our finalized version of our API was not significantly different from our

initial design, but there were enough differences in the detail to show the evolution of our design as we implemented it.

On a related note, we learned that a large chunk of the work in designing an API lies in the small details. There were many details that we didn't consider as part of our initial design, and we spent a lot more time than we thought we would thinking through those details as they arose while we implemented our solution.

Spoontacular vs Spoonacular

Our API makes some much needed improvements on the original Spoonacular API. Unlike the original Java API, our API is actually usable, and our API is easier, safer, and cheaper to use than the original REST API.

The original Spoonacular Java API is an autogenerated port of the REST API, and none of the class names or method signatures have been made human readable. On the other hand, our API was built from the key abstractions behind the Spoonacular product. We defined clear classes and enums to represent tangible things like recipes, ingredients, diets, and nutrients, meaning there is a smaller representational gap between the key concepts a user will have in mind and the code they will write.

Our API also provides many benefits over the Spoonacular REST API. Because we abstract away the organization of Spoonacular's REST endpoints, users can focus on writing simple programs to deal with the recipes. Our API also uses batch requests that aren't well documented by Spoonacular to more cheaply get users the same information they would have gotten by making the REST API calls themselves. For example, a naive version to request chicken recipes will need 20~30 points. However, using our API, it only costs 1.5 points. Additionally, our classes and enums provide for much stronger typing than the JSON returned by the REST API and the URL strings used to make the REST calls, making our API safer and more reliable to use.

Future Work

While we're happy with where our API is right now, we know that there is more work to be done to completely fix the Spoonacular API.

While our current implementation works well, it could be more cost-efficient and provide even more details about ingredients (like price and nutritional information) if we implemented caching.

There are also many use cases supported by Spoonacular, like meal planning and more advanced recipe searching, that we deemed out of scope for this initial version of Spoontacular. These use cases would need to be addressed to truly fix Spoonacular.