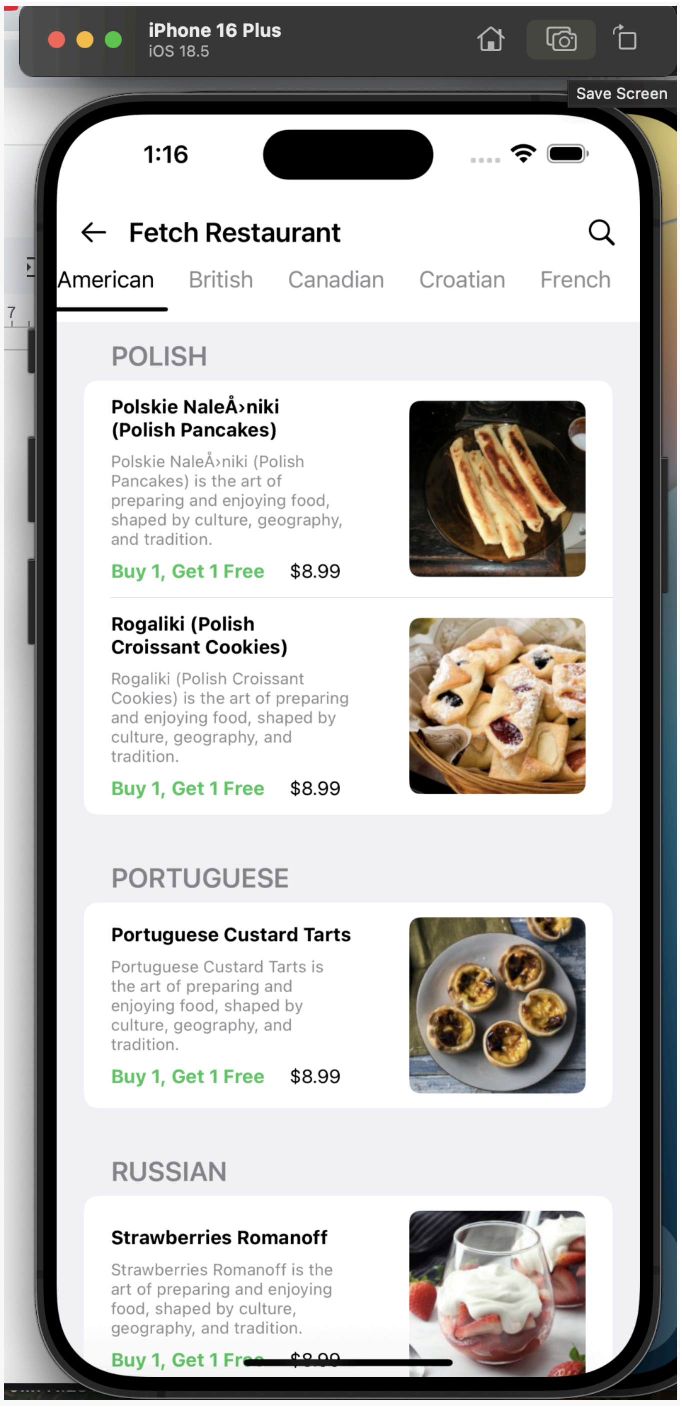
Summary: Include screenshots or a video of your app highlighting its features



POLISH

Polskie NaleÅ›niki (Polish Pancakes)

Polskie NaleÅ›niki (Polish Pancakes) is the art of preparing and enjoying food, shaped by culture, geography, and tradition.

Buy 1, Get 1 Free $8.99

Rogaliki (Polish Croissant Cookies)

Rogaliki (Polish Croissant Cookies) is the art of preparing and enjoying food, shaped by culture, geography, and tradition.

Buy 1, Get 1 Free $8.99

PORTUGUESE

Portuguese Custard Tarts

Portuguese Custard Tarts is the art of preparing and enjoying food, shaped by culture, geography, and tradition.

Buy 1, Get 1 Free $8.99

RUSSIAN

Strawberries Romanoff

Strawberries Romanoff is the art of preparing and enjoying food, shaped by culture, geography, and tradition.

Buy 1, Get 1 Free $8.99

1. This app is showing all the recipes under its own Cuisine
2. You can scroll top menu horizontally
3. Displaying all the pictures
4. All the Cuisine have been sorted

Focus Areas: What specific areas of the project did you prioritize? Why did you choose to focus on these areas?

1. In this project, I prioritized **data architecture**, **state management**, and **user interface scalability**.

2. I focused on these areas because they are the backbone of both **app performance** and **developer productivity**. Clean separation of concerns and reusable UI components make it easier to maintain and extend the app later.

Time Spent: Approximately how long did you spend working on this project? How did you allocate your time?

I spent approximately **4 hours** working on this project over the course of a week.

I broke it down like this:

- 30% – Planning and setup (20):
- 40% – Development (3 hours):
- 20% – Debugging and refactoring (30 minutes):
- 10% – Testing and preview setup (10 minutes):

Trade-offs and Decisions: Did you make any significant trade-offs in your approach?

Yes, I made a few intentional trade-offs to balance simplicity, development speed, and maintainability:

1. Used Simple `@Published` and `@State` Instead of Advanced State Management

2. Grouped Recipes in Memory Instead of on API Level

3. Skipped Pagination or Caching for Simplicity

4. Minimal Error Handling UI

Weakest Part of the Project: What do you think is the weakest part of your project?

I think the weakest part of the project is the **lack of test coverage and error-resilient networking**.

While I added basic unit tests for the `RecipeViewModel`, I didn't yet implement:

- Full coverage for all API failure scenarios
- Mocks or dependency injection for API calls
- UI-level tests to validate behavior under network loss or empty results.

If I were to improve the project further, I'd:

- Add full **mocked unit tests** with dependency injection (using a protocol for the API manager)
- Implement **loading/error states** in the SwiftUI UI
- Add **offline support or caching** if the app needed to scale

Additional Information: Is there anything else we should know? Feel free to share any insights or constraints you encountered.

One thing worth mentioning is that I built this project with scalability and clarity in mind. I focused on clean data flow using `@StateObject`, `@Binding`, and a dedicated `ViewModel`, so the architecture can support future features like filtering, favorites, or pagination.