### **Verantwoording**

De applicatie *Lazy Chef* is gemaakt voor mensen die niet weten wat ze willen koken. Het doel was om op een speelse manier recepten voor te stellen, op basis van willekeur, beschikbare ingrediënten of beschikbare tijd. Bij het *I feel lucky*-onderdeel wordt de hele ‘beleving’ verzorgd: recept, cocktail en muzieksuggestie.

### **Bouwen**

Eerst heb ik de mappen en bestanden aangemaakt waarvan ik dacht dat ik ze nodig zou hebben, om vervolgens waar nodig bij de API's een account aan te maken en een key op te vragen. Daarna heb ik een aantal functies aangemaakt: welkomstboodschap, main menu en exit. Vanuit de keuzes van het main menu ben ik gaan bouwen. Eerst *Lucky* en daarna door met *Pantry* en *Time keeper*.

Bij het opvragen van recepten kwam ik er snel achter dat *Edamam* te beperkt was, en ben ik overgestapt naar *Spoonacular*. Hetzelfde gold voor *The AudioDB*. In plaats daarvan ben ik met *Deezer* gaan werken, die ook een chart heeft. De dad-jokes waren erg flauw, dus heb ik uiteindelijk gekozen voor een luchtiger tegeltjeswijsheid van *Advice Slip.*

### **Hordes**

Het uitbreiden van een functie die je in meerdere andere functies wilt gebruiken, vond ik lastig. Met name de *show\_recipe*-functie heeft een hoop energie gekost. Van tevoren had ik besloten dat deze voor alle weergaves van recepten gebruikt zou worden. Vervolgens heb ik deze meerdere keren moeten aanpassen nadat ik hem had geschreven voor gebruik bij *lucky*: de endpoint in *pantry* gaf een lijst met minder en andere data dan bij *lucky*. Daar moest uiteindelijk een functie (*get\_recipe\_info\_id*) tussen om ook een dictionary te krijgen. Maar ook de *show\_recipe*-functie moest weer aangepast worden. Het heeft aardig wat proberen, frustratie en zoeken naar overeenkomsten tussen de verschillende endpoints gekost om erachter te komen hoe het opgelost kon worden.

Ik denk dat ik dit deels had kunnen voorkomen door eerst in de documentatie of met een pprint() te vergelijken wat de verschillende endpoints terug zouden geven.

Een tweede horde was het overzicht houden met alle functies en bijbehorende argumenten. Doordat ik voor mijn gevoel zo heen en weer geslingerd werd en de namen van variabelen veel op elkaar leken, plus functies met verschillende argumenten werden aangeroepen, schreef ik vaak verkeerde code/verwijzingen.   
Gevolg was dat ik een naam van een argument of variabele aanpaste, regelmatig op de verkeerde plek, waardoor er nog meer foutmeldingen ontstonden. Bij Lazy Chef had ik dat te vaak met het woord *data*.

Hier heb ik een aantal keer pen en papier voor gebruikt: tekenen waar een functie wordt aangeroepen, wat er mee wordt gestuurd, waar dat vandaan komt en waar dit vervolgens weer gebruikt wordt.

Hier moet ik meer mee oefenen en in het vervolg andere namen kiezen, zodat deze niet of minder overeenkomen met variabelen in bijvoorbeeld andere functies of met variabelen die je na het aanroepen weer gebruikt. Een andere oplossing zou kunnen zijn om functies onder te verdelen in groepen en elke groep in een apart bestand te zetten.

### **Doorontwikkeling**

Bij het invullen van ingrediënten is nu nog te veel interpretatie mogelijk. Dit wordt denk ik niet goed genoeg afgehandeld. Indien het niet goed gaat met de ingevoerde ingrediënten, wordt op een vrolijke manier wat anders getoond: de *lucky* functie. Dat kan wel beter. Door de invoer op te schonen, te valideren en/of aan te passen, of nieuwe invoer te vragen.

Het zoeken op basis van tijd die je wilt besteden gebeurt nu door de tijd te nemen en de resultaten aflopend te ordenen. Bij sommige invoer krijg je dan gerechten die allemaal precies dat aantal minuten kosten, bij andere invoer krijg je drie verschillende tijden. Deze tijden ontlopen elkaar meestal niet veel. Mooier zou zijn als je een ‘ongeveer’-tijd zou geven en de applicatie zoekt naar recepten met een tijd van 15 minuten minder tot 15 minuten langer, en vervolgens drie willekeurige recepten kiest binnen die criteria.  
Het sorteren zoals nu heb ik gedaan omdat anders alle gerechten tot de ingevoerde tijd in aanmerking komen. Hierdoor stonden er in het resultaat vaak recepten die veel afweken van de ingevoerde tijd.

De code opschonen, zoals functiegroepen een apart bestand geven en wellicht nog meer herhaalde code in functies zetten, zou ook een verbetering kunnen zijn.

### **Best practices**

De code is aardig DRY geworden. Dat had wat voeten in de aarde, maar ik denk dat het goed is gelukt om functies te schrijven die vaker gebruikt worden, bijvoorbeeld *get\_valid\_choice*, *get\_recipes* en *show\_recipe*.

Ook heeft het schrijven van de code aan de hand van de menustructuur gescheiden functies per onderdeel opgeleverd: *lucky*, *pantry*, *time\_keeper* — naar de beoogde functies die de applicatie moest hebben.

Om het aantrekkelijk en goed leesbaar te maken in de terminal heb ik *PrettyTable*, *pyfiglet* en *termcolor* gebruikt. Dat maakt het prettiger om de applicatie te gebruiken en lezen.

Om fouten eruit te halen heb ik gebruikgemaakt van try/except. Ook wordt in de code gecheckt of er wel invoer van een gebruiker komt en of de API wel data teruggeeft.