Muttaki I. Bismoy
G02518887
CIS612

Half-Semester Project CIS612

Abstract

I developed a website for my half-semester project that allows users to enter their birthdate or any other date and receive a variety of numerical values and a horoscope in return. This project (GitHub Link) presents a comprehensive web application designed to offer users insightful information based on their birth date. It calculates the user's age in years, months, weeks, and days, identifies their generation, and provides interesting numerological insights, such as life path numbers. Additionally, it delves into astrology, offering zodiac signs, horoscopes, and compatibility reports with other zodiac signs. The application seamlessly integrates various disciplines, such as astrology, numerology, and demographic studies, to provide a personalized and engaging user experience.

Introduction

In the age of personalized content, individuals seek more than just generic information; they look for insights that resonate on a personal level. This project taps into this desire by combining numerology and astrology to provide users with unique insights based on their birth date. It bridges the gap between complex astrological and numerological calculations and the end-user by presenting this information in an accessible and engaging web application format.

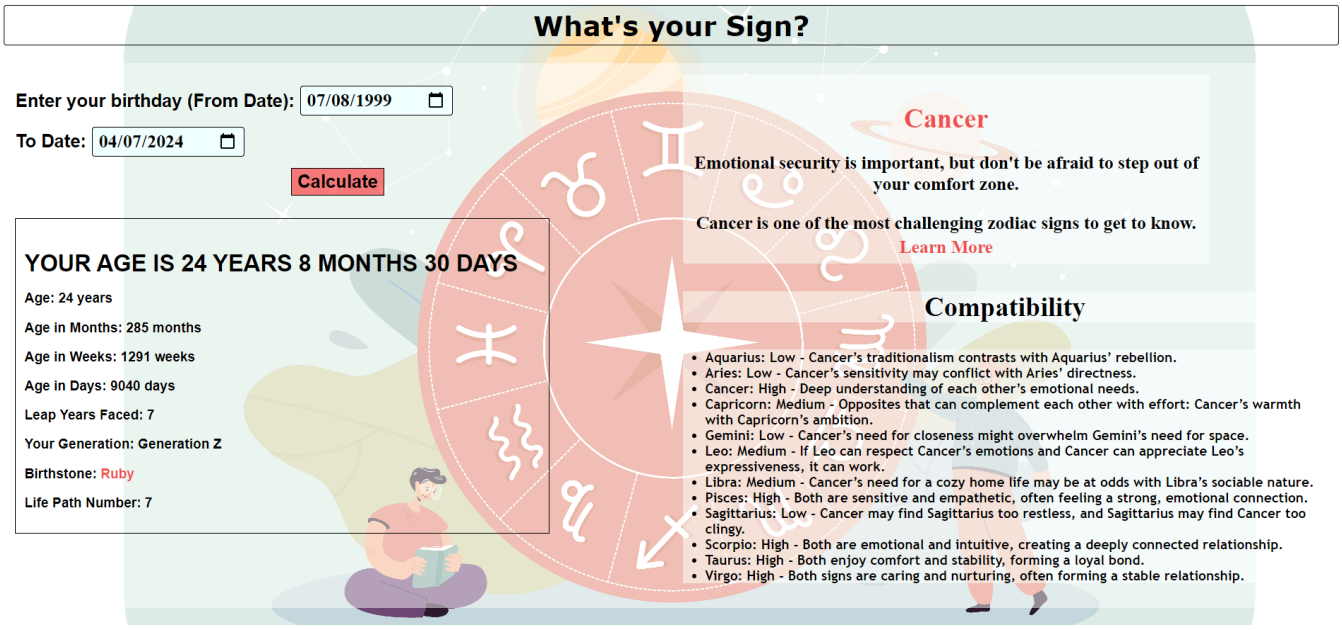


Fig. Demo of the Project

Muttaki I. Bismoy
G02518887
CIS612

Software Requirements

Despite its extensive scope, the project must remain in accordance with the Requirement Specification (CIS612) course for which it was designed.

System Boundary

Prior to beginning construction on the project, I initially identified the system and context boundaries. In reality, the system and context boundaries comprise three crucial elements that can be illustrated with an egg diagram. The system is contained within the center. The "System" components refer to the internal elements of the application that contribute in a direct manner to its operation. A system boundary encloses the entire system. System context comprises the outermost layer. The "System Context" encompasses user inputs and external data sources, among other interactions with the immediate environment surrounding the system. The context boundary delineates the system environment. The outermost layer comprises the context's irrelevant environment, which is in no way associated with the system. The term "Irrelevant Environment" refers to elements and conditions that have no bearing on the functionality of the system or its intended function. Below is a table consisting of the system, system context and irrelevant environment for my half-semester project -

System	System Context	Irrelevant Environment
Date input fields	User's birthdate input	User's clothing preference
Calculate button	Current date or a second input date	Political climate
Zodiac sign determination logic	User interaction with the UI	Stock market performance
Age calculation logic	Web server processing the calculations	User's physical location
Horoscope retrieval logic	Database of zodiac sign dates and attributes	Time of day
Life path number calculation	User's device and browser	News events of the day
Zodiac compatibility logic		Weather conditions

Software Process Model

For my half semester project, I chose the Waterfall Software Process Model for its simplicity and linear approach, which aligns well with the well-defined requirements and outcomes of this project. The model's sequential phases facilitated thorough requirement analysis, system design, implementation, and successive testing. As each phase required completion before the next began, it allowed for a more focused and disciplined development process. The model's clear milestones and deliverables were suitable for this project where requirements were well understood from the outset, and significant changes were not anticipated during development, making the Waterfall model an appropriate and straightforward methodology for execution.

The step-by-step application of the model is as follows:

1. Requirements Analysis:
In this phase, I defined what the application is supposed to do and its requirements. For example, accurately calculating age, determining zodiac signs, and providing horoscopes.

2. System Design:
Here, the architecture of the application has been outlined, detailing how the application will fulfill the requirements with a database design for storing zodiac signs and horoscopes.

3. Program Design:
During this stage, I have broken down the system design into modules or units that can be developed separately, such as a user interface module, a calculation module for age and life path number, and an API module for horoscopes.

4. Coding:
This is the actual development of the application. I wrote code following the program design to implement all the functionalities described in the requirements analysis.

5. Unit & Integration Testing:
After the application is coded, each unit or module is tested individually to ensure it works correctly (unit testing). Then, the units are combined and tested to work together (integration testing- selenium).

6. System Testing:
Here, the complete, integrated system is tested to validate that it meets the specified requirements. It involves testing for performance, security, and usability.

7. Acceptance Testing:
In this phase, the system is tested for acceptability. The client or end-users test the system to verify that it can be used in a production setting to meet their needs.

8. Operation & Maintenance:
After the application is written and tested, it enters the maintenance phase, where it will be updated and patched as necessary based on user feedback and new requirements. This step has not been applied yet to my project, but it can be applied once I deploy it online.

Use-Case Diagram

There are a lot of use cases for this project, but I have demonstrated one key feature and all the main features that take input from the user and calculate the user's age. The use case diagram and its description are as follows:

Use Case 1: Enter Birthdate
- Actors: User
- Description: The user inputs their birthdate into the designated field on the web application interface.

Use Case 2: Enter To-Date
- Actors: User
- Description: The user inputs the "to date" or the current date into the web application to calculate the age or get the horoscope.

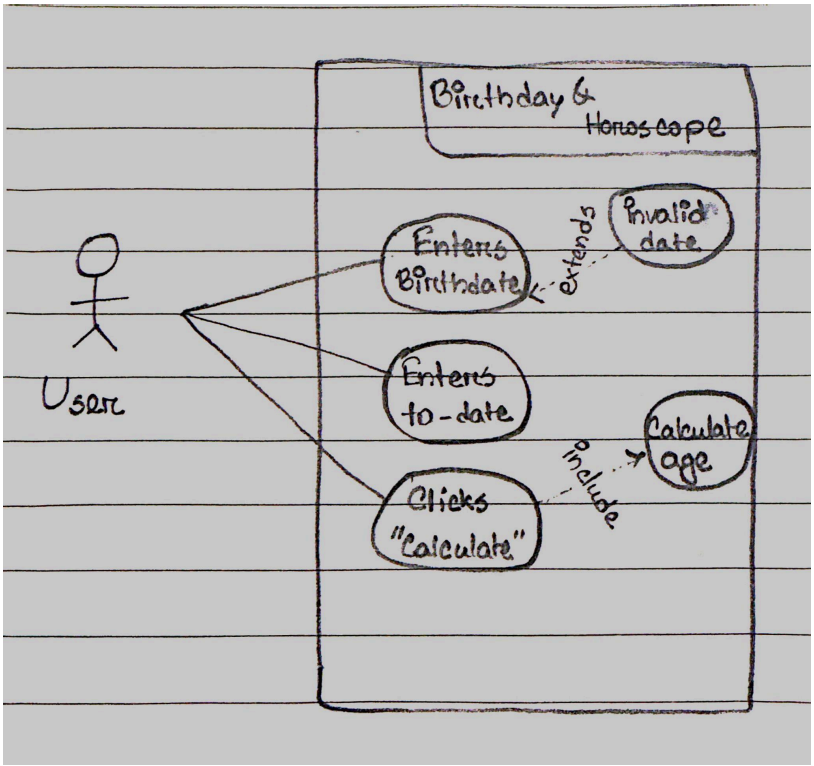


Fig. Use Case Diagram

Use Case 3: Click "Calculate"
- Actors: User
- Description: The user clicks the "Calculate" button to submit the input data and receive information such as age, zodiac sign, horoscope, etc.

Use Case 4: Calculate Age
- Actors: There’s no actor. The system conducts this use case.
- Description: The web application calculates the user's age based on the birthdate and to-date provided by the user.
- Pre-condition: The user has to enter his/her birthday and to-date first and then click on the calculate button

Use Case 5: Invalid Date
- Actors: There’s no actor. The system conducts this use case.
- Description: If the user enters an invalid date, the web application displays an error message.

Functional and Non-Functional Requirements

After drawing the table of system boundaries and the use case diagram and its description, the next thing that goes with the requirement specification course is functional and non-functional requirements.
Functional requirements describe what a system should do, detailing behaviors and functions that the system will support. They are specific to the tasks and processes that a system must perform or accommodate. Non-functional requirements, on the other hand, define how a system should behave. They specify criteria that can be used to judge the operation of a system, rather than specific behaviors. These include performance, security, usability, reliability, and compliance standards.

Functional Requirements:

1. User Input Handling:
- The application shall allow users to input their date of birth through a date picker interface.
- The application shall allow users to input a 'To Date' to calculate age from birthdate until the given 'To Date'.

2. Age Calculation:
- The application shall calculate the user's age in years, months, weeks, and days from the provided birthdate to the current date, or 'To Date'.
- The application shall correctly account for leap years in age calculations.

3. Zodiac Information:
- The application shall determine the user's zodiac sign based on their birthdate.
- The application shall provide horoscope information for the user's zodiac sign.

4. Life Path Number:
- The application shall calculate the user's life path number based on numerology from their birthdate.

5. Zodiac Compatibility:
- The application shall provide compatibility scores or insights based on the user's zodiac sign with other signs.

6. Additional Information:
- The application shall provide the birthstone associated with the user's birth month.
- The application shall display the generation category (e.g., Millennial, Generation X) based on the user's birth year.

Non-Functional Requirements:

1. Usability:
- The application shall provide an intuitive and user-friendly interface accessible to users of all ages.
- The application will be accessible on both mobile and desktop platforms via a web browser if deployed.

2. Performance:
- The application shall display results within 2 seconds of receiving user input.
- The application shall handle at least 1000 concurrent users without performance degradation if deployed.

3. Reliability:
- The application shall have an uptime of 99.9% outside of scheduled maintenance.

4. Scalability:
- The application will be able to scale to accommodate an increasing number of users.

5. Maintainability:
- The application shall be easy to maintain with well-documented code and a clear structure.

6. Compatibility:
- The application shall be compatible with major browsers, including Chrome, Firefox, Safari, and Edge.

Natural Language Requirements and Formalization

Requirement 1: The web application shall allow the user to input their birth date and a target date to calculate the age.
Pre-condition: UserAccessesApp = True
Post-condition: BirthDateAndTargetDateInputReceived = True

Requirement 2: The web application shall calculate the age in years, months, weeks, and days from the birth date to the target date.
Pre-condition: BirthDateAndTargetDateInputReceived = True
Post-condition: AgeCalculatedAndDisplayed = True

Requirement 3: The web application shall determine and display the user's zodiac sign based on their birth date.
Pre-condition: BirthDateInputReceived = True
Post-condition: ZodiacSignDeterminedAndDisplayed = True

Requirement 4: The web application shall provide horoscope information based on the user's zodiac sign.
Pre-condition: ZodiacSignDetermined = True
Post-condition: HoroscopeInfoDisplayed = True

Requirement 5: The web application shall calculate and display the user's life path number based on numerology.
Pre-condition: BirthDateInputReceived = True
Post-condition: LifePathNumberCalculatedAndDisplayed = True

Requirement 6: The web application shall provide compatibility information based on the user's zodiac sign.
Pre-condition: ZodiacSignDetermined = True
Post-condition: CompatibilityInfoDisplayed = True

Requirement 7: The web application shall display a list of significant leap years within the user's age.
Pre-condition: AgeCalculated = True
Post-condition: LeapYearsListed = True

Requirement 8: The web application shall provide a link to more information about the user's zodiac sign.
Pre-condition: ZodiacSignDetermined = True
Post-condition: MoreInfoLinkDisplayed = True

Traceability Matrix

Requirement	Age Calculation	Zodiac Sign	Horoscope	Life Path Number	Compatability	Leap Year	Learn More
1	X						
2	X					X	
3		X					
4		X	X				
5				X			
6		X			X		
7	X					X	
8		X					X

Decision Table

Let's consider the feature of calculating and displaying the user's zodiac sign and life path number. Here's how we might structure the Decision Table:

Pre-Conditions:
- User enters a valid date of birth (DOBValid = True).
- User clicks on the 'Calculate' button (CalculateClicked = True).

Post-Conditions:
- The correct zodiac sign is displayed (ZodiacDisplayed = True).
- The correct life path number is calculated and displayed (LifePathDisplayed = True).

Rules:
- If the date of birth is not valid, neither the zodiac sign nor the life path number should be calculated.
- If the date of birth is valid and the 'Calculate' button is clicked, then the zodiac sign and life path number should be calculated and displayed.

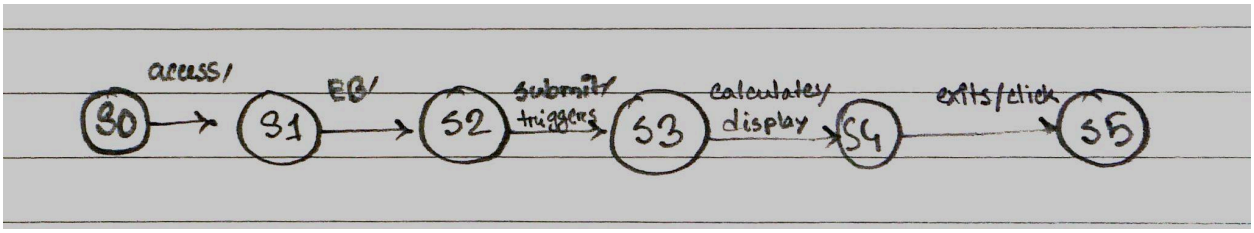
Decision Table:

Rules	R1	R2
DOBValid	F	T
CalculateClicked	-	T
ZodiacDisplayed	-	X
LifePathDisplayed	-	X

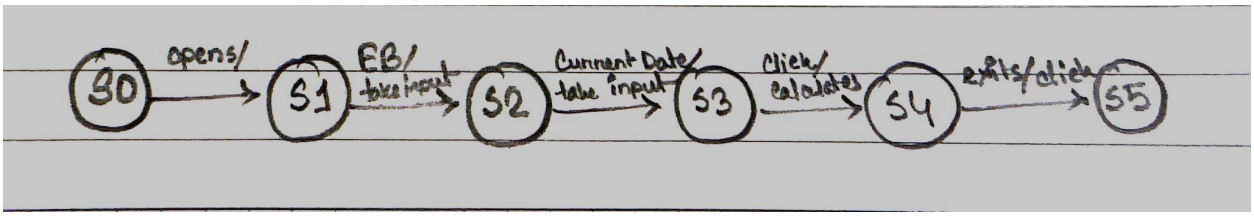
The first rule (R1) accounts for the situation where the date of birth is not valid, which means that the system should not attempt to display the zodiac or life path number. The second rule (R2) applies when the date of birth is valid and the 'Calculate' button is pressed, leading to the system displaying both the zodiac sign and life path number. This Decision Table helps to clearly define the actions the system should take based on the pre-conditions of the user's input, aligning with the formalized natural language requirements of the application. It supports the development process by ensuring a thorough understanding and clear documentation of how different input conditions should affect the system's behavior.

Finite State Machine & Statecharts

FSM & Statechart for zodiac sign and horoscope calculation:
1. Start (S0): Initial state when the user accesses the web application.
2. Enter Birthday (S1): The user inputs their birth date.
3. Submit Date (S2): The user submits the date, triggering the calculation process.
4. Calculate Sign (S3): The system calculates the zodiac sign based on the submitted date.
5. Display Results (S4): The zodiac sign and horoscope are displayed to the user.
6. End (S5): The user exits or closes the web application.

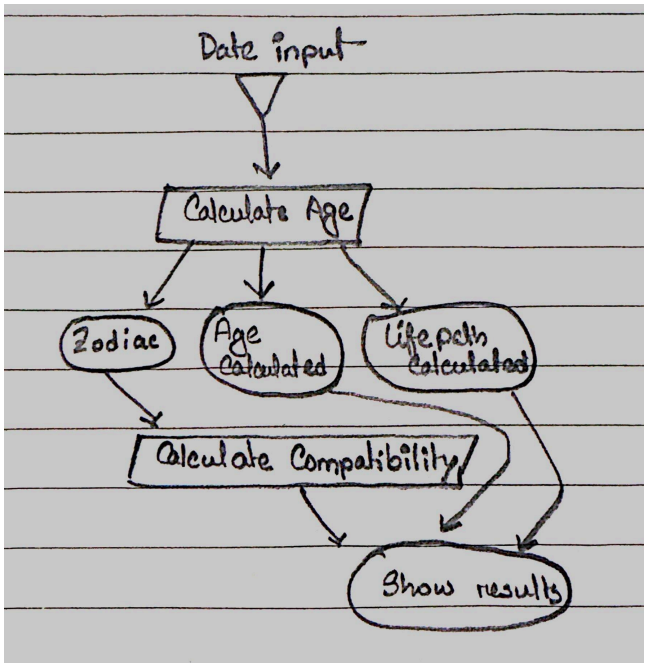


FSM & Statechart for age calculation:
1. Start (S0): The user opens the age calculation page.
2. Input Birth Date (S1): The user inputs their birth date into the system.
3. Input Current Date (S2): Either the user enters the current date or the system automatically fills it in.
4. Calculate Age (S3): The system calculates the age when the user clicks "Calculate".
5. Display Age (S4): The calculated age is displayed to the user.
6. End (S5): The process ends, either by the user closing the application or initiating a new calculation.



Petri Nets

The flow would begin with the user inputting a date, triggering the 'Calculate Age' transition. Once the age is calculated, it transitions to concurrent states where the Zodiac sign is determined, and the life path number is calculated. These two places could potentially feed into another transition: 'Calculate Compatibility', which might take the calculated age, Zodiac sign, and life path number to determine compatibility results. Finally, the transition to 'Show results' displays the outcome to the user.



Goal Modeling was skipped due to its straightforward objectives and limited scope. The application's features, such as age calculation and horoscope provision, presented clear and well-understood goals without the need for elaborate analysis. Given its user-centric design, the project benefited more from direct implementation and feedback than extensive upfront planning. Adopting the approach allowed the system to prioritize rapid delivery of working software and adapt to user feedback, making the extensive goal modeling process redundant. Resource constraints also played a role in this decision, as goal modeling can be time-consuming and resource-intensive, which might not align with the project's scope and scale.

Why did I choose to do these things?

Initially, I set out to understand the limits within which my application would function and interact. My intention in establishing these boundaries was to guarantee precision in the process of creating the application. This involved differentiating between the fundamental elements of the application (the system), the interaction with users and external data (the context), and elements that are not affected by or do not affect the system (the irrelevant environment). The design of my project was purposefully tailored to meet the exact requirements of my CIS612 course. I incorporated the system boundary egg diagram into my work to align with academic standards and objectives. The Waterfall Software Process Model was selected due to its sequential nature, enabling me to address each phase of the project individually, starting from requirements analysis and concluding with operation and maintenance. This decision was pivotal in maintaining concentration and ensuring a methodical development process. I have provided a comprehensive description of both the functional requirements, which outline the specific actions that your system must be able to perform, and the non-functional requirements, which establish the criteria for how the system should operate. This differentiation is crucial, as it not only determines the functionality of the application but also the level of excellence it must adhere to. The use case diagram provides a visual representation of various user interactions that the system needs to manage, such as inputting dates and performing calculations. It aids in comprehending user demands and developing the system interface and functionalities accordingly. These aid in converting user expectations into technical specifications that direct the development process. Their role is to verify that the software's functionality is in accordance with the requirements of the users and the objectives of the system. This formalization entails articulating requirements in a manner that is practical and implementable for developers, specifically referring to myself. It was essential to ensure that each specified requirement could be directly traced and verified through testing or demonstration. The formalized NLR played a crucial role in not only clearly defining the requirements but also in effectively managing and tracking the software development lifecycle using the traceability matrix. The implementation of this method greatly improved the project's capacity to provide a software solution that was in line with the requirements and expectations of the users. The decision table method provides a precise approach for representing intricate decision logic that involves multiple conditions and their corresponding outcomes. It guarantees that all possible situations are taken into account and managed accurately in the software, thereby minimizing mistakes and enhancing the resilience of the system. Finite State Machines (FSM) and statecharts offer a lucid representation of the various states that a system can assume and the manner in which it transitions between these states, contingent upon user actions or other events. This is especially valuable for comprehending the dynamic behaviors within the system, such as the transition from input acquisition to computation. Petri nets are utilized for representing concurrent processes and system states, providing a visual representation that aids in the analysis and design of intricate systems where multiple processes may occur simultaneously or interact with one another.

Methodology

How the Project Works

The project consists of a frontend web application and a backend server. The frontend, developed with Vue.js, provides a user-friendly interface where users can input their birth date and a comparison date (usually the current date) to calculate their age. Once the calculation is requested, the frontend sends this data to the backend. The backend, built with Flask, processes the request by calculating the user's age, determining their zodiac sign, generating a life path number, and fetching their horoscope and zodiac compatibility. It then sends this information back to the frontend, where it is displayed to the user.

Key functionalities include:
- Age Calculation: Computes how old the user is in years, months, weeks, and days.
- Zodiac Sign Identification: Determines the user's zodiac sign based on their birth date.
- Life Path Number Calculation: Calculates the user's life path number, offering insights into their personality and life's journey.
- Horoscope and Zodiac Compatibility: Provides daily horoscopes and compatibility scores with other zodiac signs.

What the Project Does

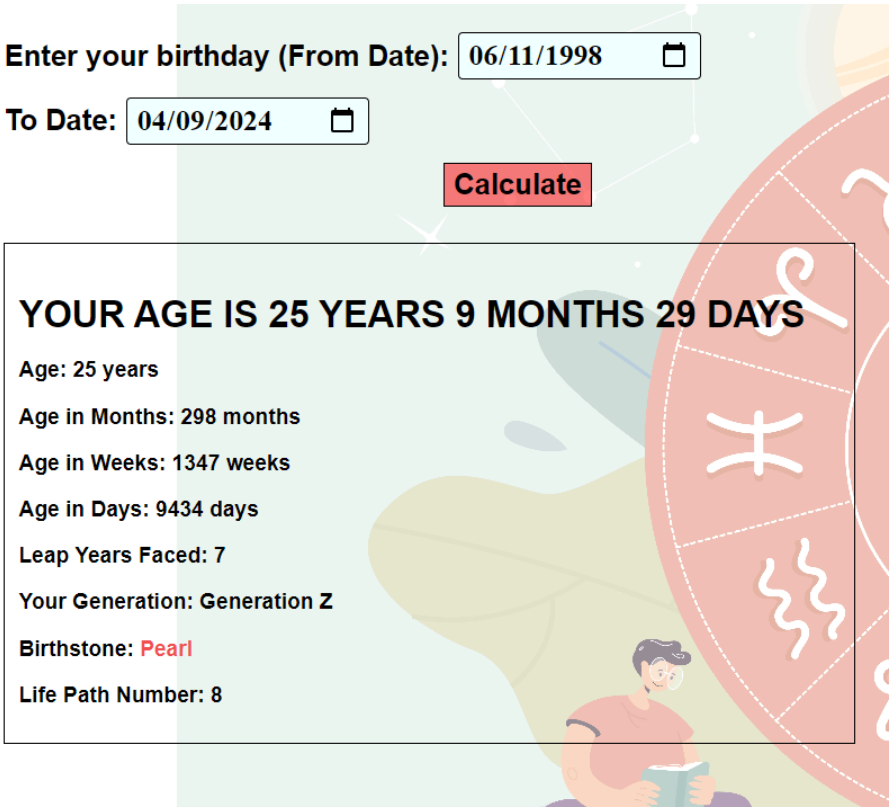
The project offers a multifaceted approach to understanding oneself through various lenses:
- Astrological Insights: By providing the user's zodiac sign and daily horoscope, the application offers a glimpse into the astrological influences on their day-to-day life and personality.
- Numerological Analysis: The life path number calculation reveals key personality traits and potential life paths, adding another layer of personal insight.
- Demographic Context: The generation identification offers a broader demographic context, connecting the user with broader societal cohorts.
- Compatibility Reports: The zodiac compatibility feature allows users to explore their relationships with others through the astrological compatibility scores, adding a fun and interactive element to the experience.

What are some useful functions, and what do they do?

1. **`calculate_age()` :** The snippet of code is part of a Flask application's backend route that handles the calculation of a user's age based on input dates. When the '/calculate' route receives a POST request with JSON data, it retrieves the 'fromDate' and 'toDate' values, which represent the user's birthdate and the date to calculate the age until, respectively. The `datetime.strptime` function converts these date strings into `datetime` objects. The age in years is calculated by subtracting the birth year from the current year and adjusting for whether the current date has surpassed the birth date in the current year. Age in months multiplies the years by 12 and adds the difference between the current and birth months. Age In weeks, divide the total days by 7. The `count_leap_years` function calculates the number of leap years experienced by the user, which could be used to further refine the age calculation. The `get_zodiac_info` function presumably determines the zodiac sign based on the birth date and could provide additional astrological information such as a horoscope and birthstone. Additional code calculates 'extra_months' and 'extra_days' for a more precise age and determines the user's generation based on the year of birth. Finally, the function returns a JSON object with all these calculated values, including the user's age in years, months, weeks, and days, the number of leap years faced, generation, birthstone name, and URL for more information about the birthstone. This data is likely to be used in the frontend to display it to the user.

```python
@app.route('/calculate', methods=['POST'])
@cross_origin(origin="localhost:8080")
def calculate_age():
    if not request.is_json:
        print("Request is not JSON!")
        return jsonify({"error": "Missing JSON in request"}), 400

    data = request.get_json()
    print("Received data:", data)
    from_date = datetime.strptime(data['fromDate'], '%Y-%m-%d')
    to_date = datetime.strptime(data['toDate'], '%Y-%m-%d')
    delta = to_date - from_date

    age_years = to_date.year - from_date.year - ((to_date.month, to_date.day) < (from_date.month, from
    age_months = age_years * 12 + to_date.month - from_date.month
    age_weeks = delta.days // 7
    age_days = delta.days
    leap_years = count_leap_years(from_date.year, to_date.year)

    zodiac_info = get_zodiac_info(from_date)

    extra_months = to_date.month - from_date.month - (to_date.day < from_date.day)
    extra_days = to_date.day - from_date.day

    if extra_days < 0:
        days_in_prev_month = (to_date - timedelta(days=to_date.day)).day
        extra_days += days_in_prev_month

    if extra_months < 0:
        extra_months += 12

    generation = "Unknown"
    if from_date.year < 1946:
        generation = "The Silent Generation"
    elif from_date.year < 1965:
        generation = "Baby Boomer"
    elif from_date.year < 1981:
        generation = "Generation X"
    elif from_date.year < 1997:
        generation = "Millennials"
    elif from_date.year < 2013:
        generation = "Generation Z"


    birth_month = from_date.month
    birthstone_name, birthstone_url = birthstones[birth_month]

    return jsonify({
        'age': age_years,
        'months': age_months,
        'weeks': age_weeks,
        'days': age_days,
        'leapYears': leap_years,
        'generation': generation,
        'extra_months': extra_months,
        'extra_days': extra_days,
        'birthstone': {
            'name': birthstone_name
```

2. `get_zodiac_sign(month, day)`: This function determines a user's zodiac sign based on their birth date. It checks the provided month and day against a predefined range of dates corresponding to each zodiac sign.



3. `get_zodiac_info(birth_date)`: This function acts as an aggregator, gathering various pieces of zodiac-related information. It uses the user's birth date to determine their zodiac sign, life path number, and zodiac compatibility.

- `calculate_life_path_number(month, day, year)`: Called within `get_zodiac_info`, this function calculates the life path number based on the numerology of the user's birth date.
- `get_zodiac_compatibility(zodiac_sign)`: Also called `get_zodiac_info`, it retrieves compatibility scores or insights based on the user's zodiac sign.

If a zodiac sign is determined, the function fetches the horoscope, facts about the sign, and a URL for more information from predefined dictionaries (`horoscopes` and `facts_about_signs`). It then bundles this information into a JSON-like dictionary, including the life path number and compatibility information, which is likely sent back to the frontend for display. If the zodiac sign is unknown (e.g., if the birth date does not match any predefined zodiac date ranges), it defaults to returning a dictionary with values set to "Unknown" or "No horoscope available."
The frontend portion (not shown in the snippet but implied), would display this information appropriately, likely under a section titled "Zodiac Sign" or similar. This functionality enriches the web app by providing personalized zodiac and numerology information based on the user's inputted birth date.

```
def get_zodiac_sign(month, day):
    for zodiac, (start_date, end_date) in zodiac_dates.items():
        if (month == start_date[0] and day >= start_date[1]) or \
           (month == end_date[0] and day <= end_date[1]):
            return zodiac
    return "Unknown"

def get_zodiac_info(birth_date):
    month = birth_date.month
    day = birth_date.day
    year = birth_date.year
    zodiac_sign = get_zodiac_sign(month, day)
    life_path_number = calculate_life_path_number(month, day, year)
    compatibility_info = get_zodiac_compatibility(zodiac_sign)

    if zodiac_sign:
        horoscope = horoscopes.get(zodiac_sign, "No horoscope available.")
        facts = facts_about_signs.get(zodiac_sign, "No facts available.")
        more_info_url = f"https://en.wikipedia.org/wiki/{zodiac_sign}_(astrology)"

        return {
            'sign': zodiac_sign,
            'horoscope': horoscope,
            'facts': facts,
            'more_info_url': more_info_url,
            'life_path_number': life_path_number,
            'compatibility': compatibility_info
        }
    else:
        return {
            'sign': "Unknown",
            'horoscope': "No horoscope available.",
            'facts': "No facts available.",
            'more_info_url': "https://en.wikipedia.org/wiki/Zodiac"
        }
```

Fig. Backend of determining Zodiac Sign and Zodiac Info

Conclusion

In creating the Birthday and Horoscope web application, the process was streamlined to prioritize direct implementation and functional clarity. The project's objectives were straightforward, centered on age calculation and horoscope provision. The development process followed the Waterfall model, providing a structured and sequential approach well-suited to the clearly defined requirements. By focusing on formalizing the Natural Language Requirements, the project ensured accurate translations of user needs into system functions. The decision to forego goal modeling was strategic, given the application's limited scope and specific user-centric features, making the utilization of resources more efficient. Each requirement was carefully considered and included for its direct contribution to the application's functionality, ensuring a focused development with immediate benefits to the user experience. This approach effectively fulfilled the project's aim, allowing for efficient feature delivery and adaptability to user feedback post-deployment.