Московский государственный технический университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»
Отчет по проекту
«Веб-приложение для расчета орбиты кометы»

Выполнили:
Баженов Никита ИУ5-31Б
Бусыгин Артем ИУ5-36Б
Паронько Даниил ИУ5-31Б
Чоботов Лука ИУ5-31Б

Проверил:
Нардид А. Н.

2025 г.

Задание

Цель

Создать приложение для определения орбиты кометы по наблюдениям и расчета её сближения с Землей.

Входные данные

Минимум 5 наблюдений, каждое содержит:

- Прямое восхождение (RA)
- Склонение (DEC)
- Время наблюдения
- Фотография

Этап 1: Расчет орбиты

Определить 6 орбитальных параметров:

- Большая полуось (a) в а.е.
- Эксцентриситет (e)
- Наклонение (i)
- Долгота восходящего узла (Ω)
- Аргумент перицентра (ω)
- Время прохождения перигелия (T)

Этап 2: Расчет сближения

Вычислить:

- Дату минимального сближения с Землей
- Расстояние минимального сближения

Технические требования

Разрешено использовать

- DeepSeek, поисковые системы
- Библиотеки: Astropy, PyEphem, PyAstronomy, poliastro

Обязательно

- Ввод данных наблюдений
- Загрузка фотографий
- Тесты на известных объектах (например, Марс)
- Валидация через JPL Horizons: https://ssd.jpl.nasa.gov/horizons/app.html#/

Опциональные улучшения

- Горизонтальная система координат

- Графический редактор для выбора точки
- ИИ-детекция кометы
- Архитектура с микросервисами
- Хранение расчетов и история пересчетов
- Добавление новых наблюдений к существующим

Описание проекта

Проект состоит из 4-х микросервисов: **фронтенд**, **бекенд на Go**, **бекенд на Python**, **база данных PostgreSQL**. Каждый микросервис запускается с помощью Docker, а весь проект через Docker Compose.

[Весь код проекта в репозитории]

Запуск проекта

Для запуска веб-приложения использовался Docker.

Каждый отдельный глобальный компонент веб-приложения (фронтент, бекенд, база данных) имеет свой Dockerfile. Запуск всего проекта происходит с помощью docker compose

yaml

```yaml
services:
  frontend:
    build:
      context: ./frontend/app
      dockerfile: Dockerfile
    container_name: frontend
    ports:
      - "80:80"
    depends_on:
      - backend
      - astra-backend
    restart: unless-stopped

  astra-backend:
    build:
      context: ./astro_service
      dockerfile: Dockerfile
    container_name: backend-astra
    restart: unless-stopped
    ports:
      - "8000:8000"

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: backend-app
    restart: unless-stopped
    ports:
      - "9090:9090"
    environment:
      - DB_HOST=postgres
      - DB_PORT=5432
      - DB_USER=postgres
      - DB_PASSWORD=postgres
      - DB_NAME=postgres
    depends_on:
      - astra-backend
      - postgres

  postgres:
    container_name: postgres_db
    image: postgres:15-alpine
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
      - POSTGRES_DB=postgres
    volumes:
      - postgres_data:/var/lib/postgresql/data
```

```
    restart: unless-stopped

volumes:
  postgres_data:
```

Перед всем приложением стоит reverse-proxy Nginx, который был настроен на корректную работу с SPA приложением

nginx

```
upstream backend_api {
    server backend:9090;
}

server {
    listen 80;
    server_name localhost;

    location /api/ {
        resolver 127.0.0.11 valid=200s;

        proxy_pass http://backend:9090;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;

        try_files $uri $uri/ /index.html;
    }

    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }
}
```

Фронтенд

Язык: JavaScript
Фреймворк: React

Для создания проекта был выбран фреймворк React по нескольким причинам:

- Реактивная архитектура – автоматическое обновление интерфейса при изменении данных благодаря Virtual DOM, что обеспечивает высокую производительность и отзывчивость приложения.
- Компонентный подход – переиспользуемые изолированные компоненты упрощают разработку, тестирование и поддержку кода.
- Богатая экосистема – огромное количество готовых библиотек, инструментов и решений для типичных задач (роутинг, управление состоянием, UI-компоненты).
- Большое сообщество – активная поддержка, обширная документация и множество готовых решений для любых задач.
- Гибкость – React не навязывает строгую структуру проекта, позволяя выбирать подходящие инструменты под конкретные требования.
- Производительность – эффективный механизм обновления DOM минимизирует операции перерисовки, что критично для сложных интерфейсов.

Клиент представляет из себя SPA приложение, взаимодействие с бекендом происходит с помощью AJAX запросов.

Были созданы компоненты:
- Header – заголовок сайта
- History – отображает результаты ранних вычислений
- HistoryCard – единичная запись в истории
- ObservationsForm – форма, в которую пользователь вносит данные для расчетов
- ObservationInput – единичная строка ввода данных в форме
- ImageUploader – загрузка картинки кометы
- Orbit – отображение результата расчетов
- OrbitVisualization - визуализация орбиты

Бекенд на Go

Фреймворк: echo
Базы данных: PostgresSQL

Для создания проекта был выбран фреймворк Echo по нескольким причинам:
- Echo использует высокооптимизированный HTTP-маршрутизатор на основе Radix tree. Это обеспечивает мгновенный поиск маршрутов даже в API с тысячами эндпоинтов.
- Встроенный Data Binding и Валидация. В отличие от стандартного net/http где нужно вручную парсить JSON или query-параметры, Echo делает это одной строкой.
- Echo работает поверх стандартного интерфейса net/http. Это означает, что вы можете использовать любые стандартные Go-библиотеки и middleware, не беспокоясь о совместимости
- Echo предоставляет богатый интерфейс контекста, который упрощает работу с ответами.

Для взаимодействия с базой данных PostgresSQL использовали GORM (Go Object-Relational Mapping). Данная библиотека позволяет работать с данными через обычные Go-структуры, а не писать сырые SQL-запросы вручную.Структура Application-сервера, написанного на языке Go, была создана на основе паттерна «Луковица». В самом центре — ядро Domain (бизнес-сущности и интерфейсы), полностью независимое от внешнего мира. Вокруг ядра находится слой Service, который реализует бизнес-логику, оперируя моделями из ядра и вызывая интерфейсы репозиториев, не зная об их реализации. Взаимодействие с внешним миром (базой данных) происходит в слое Infrastructure (в файле repository), который реализует интерфейсы из ядра: здесь создаются SQL-запросы, данные из БД преобразуются в доменные модели и возвращаются сервису. Точкой входа для

внешних запросов служит слой Presentation (папка handlers), который принимает HTTP-запросы, валидирует входные данные и передает их на обработку в service, получая результат для ответа клиенту.

Были созданы 4 модели:

- Модель со всеми характеристиками по планете

```go
type CometAllCharestic struct {
  ID           string           `gorm:"primaryKey" json:"id"`
  CharesticID  string           `gorm:"uniqueIndex" json:"charesticId"`
  NameComet    string           `json:"nameComet"`
  Charestic    OrbitalCharestic `gorm:"foreignKey:CharesticID;references:ID;constraint:OnUpdate:CASCADE,OnDelete:SET NULL;" json:"orbitalCharestic"`
  Observations []Observation    `gorm:"foreignKey:CometID;constraint:OnUpdate:CASCADE,OnDelete:CASCADE;" json:"observations"`
}
```

- Модель с расчётными характеристиками, которые мы получили от сервиса, написанного на Python.

```go
type OrbitalCharestic struct {
  ID                     string  `json:"id"`
  LargeSemiAxis          float64 `json:"largeSemiAxis"`
  Eccentricity           float64 `json:"eccentricity"`
  Inclination            float64 `json:"inclination"`
  LongitudeAscendingNode float64 `json:"longitude"`
  Pericenter             float64 `json:"pericenter"`
  TrueAnomaly            float64 `json:"trueAnomaly"`
  MinDistance            float64 `json:"minDistance"`
  MinApproximationDate   string  `json:"minApproximationDate"`
}
```

- Модель, описывающая одно наблюдение за космическим объектом.

```go
type Observation struct {
  ID                  string `gorm:"primaryKey" json:"id,omitempty"`
  CometID             string `gorm:"index" json:"cometId,omitempty"`
  DirectAscension     string `json:"directAscension"`
  CelestialDeclination string `json:"celestialDeclination"`
  Date                string `json:"date"`
}
```

- Модель, которая хранит информацию, которая пришла с клиента

```go
type CometObservationsRequest struct {
  NameComet    string        `json:"nameComet"`
  Observations []Observation `json:"observations"`
}
```

В центре архитектуры находятся чистые бизнес-модели и логика, не зависящие от внешнего мира. Вокруг них выстроен прикладной слой, который реализует сценарии использования и оркестрирует потоки данных. На внешнем контуре располагается инфраструктура: handlers принимают HTTP-запросы от клиентов, Repository отвечает за персистентность данных (БД), а отдельный Integration-модуль инкапсулирует всё взаимодействие с внешним сервисом на Python. Зависимости направлены строго внутрь: внешний слой (API и БД) зависит от логики, но сама логика ничего не знает о деталях реализации HTTP, SQL или Python-клиента.

Также была настроена CORS-политика. Эта команда настраивает CORS (Cross-Origin Resource Sharing) для нашего сервера. Без неё браузер запретит веб-страницам с других адресов обращаться к вашему API.

```go
astroEcho.Use(middleware.CORSWithConfig(middleware.CORSConfig{
  AllowOrigins: []string{"https://localhost:9090"},
  AllowMethods: []string{
    http.MethodGet,
    http.MethodPost,
    http.MethodDelete},
  AllowCredentials: true, MaxAge: 300}))
```

Бекенд на Python

1. Обзор бекенда

Разработан вычислительный модуль по определению орбит комет и расчета их сближения с Землей.
Входные данные: минимум 5 наблюдений (RA, Dec, время)
Выходные данные: 6 орбитальных параметров + дата и расстояние сближения
Язык разработки: Python

2. Технологии

- NumPy - векторные вычисления
- Astropy - координаты, эфемериды планет
- Poliastro - орбитальная механика
- SciPy - оптимизация (least_squares, minimize_scalar)
- FastAPI - сервер для взаимодействия с бекендом на Go

3. Архитектура

Модуль 1: astra.py - определение орбиты Модуль 2: CometApproach.py - расчет сближения Первый модуль принимает на вход наблюдения кометы (прямое восхождение, склонение, время) и определяет 6 орбитальных параметров методом наименьших квадратов. Второй модуль использует полученную орбиту и рассчитывает дату и расстояние минимального сближения с Землей. Такое разделение позволяет независимо тестировать и модифицировать каждый компонент. Поток данных организован следующим образом: входные наблюдения (RA, Dec, время) → преобразование координат → начальное приближение → оптимизация методом наименьших квадратов → орбитальные элементы в эклиптической системе → пропагация орбиты во времени → поиск минимума расстояния → дата и расстояние сближения.

4. Реализованные модули

4.1 Определение орбиты (astra.py)
В центре модуля находится функция DetermineOrbit, которая реализует алгоритм

метода наименьших квадратов. Процесс начинается с сортировки наблюдений по времени и выбора эпохи (среднее наблюдение). Далее позиции Земли для всех моментов наблюдений предвычисляются и кэшируются в словаре. Целевая функция оптимизации CalculateResiduals для каждого набора орбитальных параметров создает орбиту, пропагирует её к моментам наблюдений и вычисляет невязки между предсказанными и наблюдаемыми координатами:

```python
residuals[2*idx] = predictedRa - obs.directAscension
residuals[2*idx + 1] = predictedDec - obs.celestialDeclination
```

Начальное приближение вычисляется эвристически на основе углового перемещения кометы:

```python
if angularSeparation > 10 and deltaTime < 30:
  estimatedA = 2.5; estimated = 0.7
elif angularSeparation < 1 and deltaTime > 100:
  estimatedA = 20.0; estimatedE = 0.85
```

Быстро движущиеся объекты получают меньшую большую полуось, медленные - большую. Функция ConvertOrbitToEcliptic преобразует полученную орбиту из международной небесной системы отсчета (ICRS) в барицентрическую эклиптическую систему координат через создание объектов SkyCoord. Это необходимо, так как классические орбитальные элементы комет традиционно определяются относительно плоскости эклиптики.

4.2 Расчет сближения (CometApproach.py)
Функция FindClosestApproach реализует поиск глобального минимума расстояния методом одномерной оптимизации. Ключевая особенность - кэширование позиций Земли в словаре earthCache, что предотвращает повторные вычисления эфемерид:

```python
if timeJD in earthCache:
  earthHeliocentric = earthCache[timeJD]
else:
  earthHeliocentric = GetEarthPosition(currentTime)
  earthCache[timeJD] = earthHeliocentric
```

Метод оптимизации - Brent с ограничениями и точностью 1e-8 суток (≈0.86 секунды). При ошибках система автоматически переключается на более робастный метод золотого сечения. Функция CalculateCometApproachData является точкой входа модуля и по умолчанию выполняет поиск сближения на горизонте 300 лет от текущего момента.

5. Оптимизация производительности

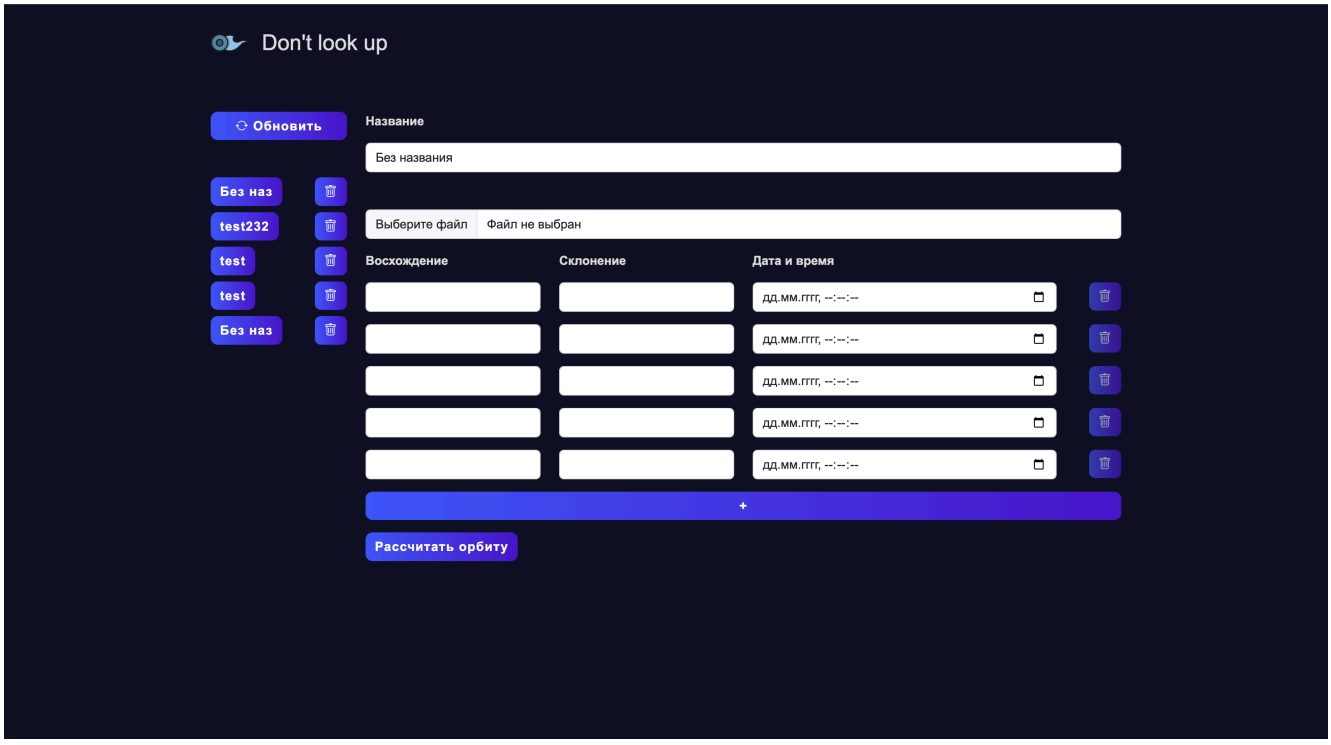
Кэширование эфемерид - позиции Земли вычисляются один раз и сохраняются в словарях. Это сокращает число обращений к функции get_body. Векторизация - все геометрические расчеты выполняются через векторизованные операции NumPy, избегая медленных циклов Python. Ограничение итераций - параметр

max_nfev=200 в методе наименьших квадратов предотвращает зависание при патологических случаях и обеспечивает гарантированное время отклика.

6. Математические методы

Определение орбиты основано на минимизации суммы квадратов невязок между наблюдаемыми и предсказанными положениями кометы. Метод Trust Region Reflective обеспечивает надежную сходимость при наличии строгих границ параметров. Пропагация орбиты во времени выполняется через решение уравнения Кеплера, реализованное в библиотеке Poliastro. Поиск минимального сближения использует метод Brent - комбинацию золотого сечения и параболической интерполяции, обеспечивающую суперлинейную сходимость без вычисления производных.

Пример работы



Don't look up

Обновить

Без наз
test232
test
test
Без наз

Название

test23235

Выберите файл | Файл не выбран

Восхождение	Склонение	Дата и время
345	1	17.10.2025, 18:08:03
345	112	09.10.2025, 18:08:05
45	367	15.10.2025, 18:08:07
67	78	16.10.2025, 18:08:09
189	25	29.10.2025, 18:08:11

+

Рассчитать орбиту

A = 81.41536766514486
E = 0.11910368483958501
I = 112.56131130358321
Ω = 200.72051406167262
ω = 19.28933635658249
ν = 79.50266331196411

Минимальное расстояние = 78.55934737566598
Дата сближения = 2025-11-01T11:11:28