zenitech

STEVENS&BOLTON

# New Beings

## 1. Introduction to New Beings

**Christopher Lacy-Hulbert**
*Founder and CTO at Zenitech*

## 2. Legal issues and AI

**Tom Lingard**
*Partner at Stevens & Bolton*

## 3. Demystification: a glossary of AI terms

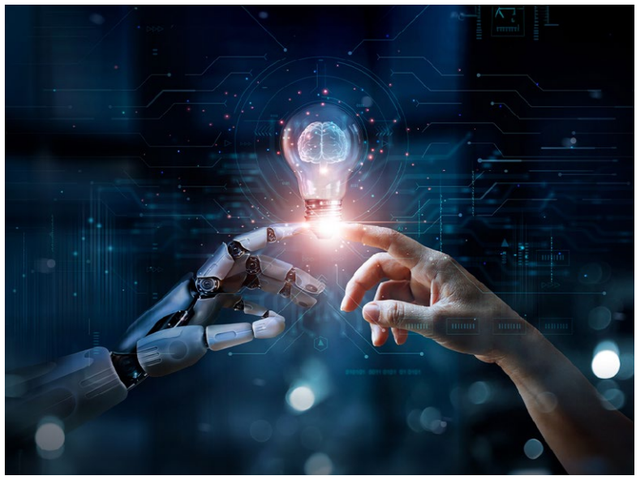
**Christopher Lacy-Hulbert**
*Founder and CTO at Zenitech*

## 4. Accelerating iOS development with AI tools

**Péter Sebestyén Krassay**
*Expert iOS Developer at Zenitech*

# A series of articles on AI

# CONTENTS



# New Beings: AI in Software Development

Zenitech has partnered with law firm, Stevens & Bolton, to create **New Beings** - a series of articles exploring how AI can, and will, be used in software development.

Join us for the journey, as we examine a range of issues, including the legal issues surrounding artificial intelligence, the practicalities of using AI to aid software development, how AI tools can assist with coding, and the issues around security and testing.

# Introduction to New Beings: a series of articles on AI

1

**Christopher Lacy-Hulbert**
*Founder and CTO at Zenitech*

Welcome to our New Beings series, covering the rapidly emerging topic of artificial intelligence in the software development domain. Here at Zenitech, a substantial amount of our work consists of high-quality software development for global brands - this work is often at a significant scale and of a mission-critical nature.

Led by an intense curiosity to understand the implications of artificial intelligence in our work, we engaged in a series of detailed investigations covering a broad spectrum of topics.

Through these investigations, Zenitech, and other companies active in software development, can benefit from a greater understanding of the capabilities of these tools, and how we might use them to advance our work and increase the competitiveness of ourselves and our clients.

We'll be sharing this series of articles over the next few weeks, so if this is an area of interest to you, please follow us and join in the discussion.

The series will cover the technical aspects of artificial intelligence in software development and consider the legal implications of working with AI, such as data, privacy, copyright infringement, intellectual property, and information security. Our goal is for this series to be used as a guide for software developers in general.

To be clear, this article series is not about the broader topic of business enablement through AI. However, we will be publishing tangible use-cases in the near future to illustrate how any business can or should use LLMs (Large Language Models) and other AI technology to enhance their business offering and remain competitive. These considerations are outside the immediate scope of software development. If this is an area of interest to you, please get in touch and we can help you shape your thoughts.

Our very best technical talent worked on this series, in collaboration with our legal partners at Stevens & Bolton.

# Legal issues and AI

**Tom Lingard**
*Partner at Stevens & Bolton*

2

Disclaimer: Artificial intelligence is a complex and rapidly evolving technology and law - in this article, I summarise some key legal issues that arise from using AI.

It is not intended to constitute legal advice and is for discussion purposes only.

This article is part of the ‘New Beings’ series of articles from Zenitech and Stevens & Bolton, examining the practicalities, issues and possibilities of using AI in development.
Today, we examine some of the legal issues surrounding AI and its use in development.

## AI and intellectual property rights

There are two main points when considering intellectual property (IP) in the context of AI - protection and ownership.

The question of ownership of IP rights in AI generated content is not straightforward and, at the moment, varies between jurisdictions. In some countries material generated by AI is not eligible for copyright protection due to the lack of human authorship.

In the UK, 'computer-generated works' are eligible for copyright protection and the author/owner of such computer-generated works is held to be the person by whom the arrangements necessary for the creation of the work are undertaken. However, the legislation dates from 1988 and does not necessarily translate well to modern, complex AI platforms.

### Infringement and use of third-party IP

All AI systems need to be trained and learn from data. In some cases, that data is specialised/unique and provided by the parties developing the AI (e.g. medical data sets). However, in others, it is obtained from publicly available (but not necessarily freely usable) data – including unsophisticated scraping of the web.

Again, there is a divergence of how this is addressed in different jurisdictions. The UK Government recently put its plans to introduce a data mining exemption to copyright infringement on hold, but on 29 June 2023 the UK Intellectual Property Office announced that work had started to develop a voluntary code of practice for copyright and AI; and that If the code of practice is not adopted, or agreement is not reached, legislation could be considered.

Rights holders across various jurisdictions are already bringing claims for infringement, including the claim brought by Getty Images against Stable Diffusion.

Of particular relevance to software developers is the extent to which an AI may have been trained on third-party or open-source code. Is it putting it into the code it is generating for you? Is the correct licence included?

### Thaler v Comptroller General

Stephen Thaler created the artificial intelligence known as DABUS (Device for the Autonomous Bootstrapping of Unified Sentience). The AI went on to create two new products - a food container that it constructed using fractal geometry (which would facilitate rapid reheating of food) and a flashing beacon that could be used for emergencies.


Thaler has been attempting to patent the products made by DABUS, but various courts around the world have ruled against him - saying that Thaler cannot patent something that wasn't created by a human. The final appeal against the UK ruling was heard by the UK Supreme Court in March.

Whilst the case focuses on ownership and inventorship in the field of patents, the implications of the judgement could be huge. For example, what does this mean for the things you create for companies that use AI as part of the development process?

## Liability and accountability

Then there's the question of liability. Some questions that need to be considered include:

- If an AI is generating faulty/infringing code, is that because it was trained on bad code in the first place, or due to how it is generating the results?
- Is the data scientist or developer liable, or the company executives who signed off on its deployment?
- Can you even tell where the problem originated?
- If AI-generated code that is licensed out to another entity causes harm to an end user, would the licensor and/or licensee be liable?
- Who will compensate the injured parties? What position might the parties' respective insurers adopt?
- What would be the applicable standard of care that would apply if AI is making the decisions that cause harm?

## Privacy and data protection

AI requires access to large amounts of data to function, which raises concerns about the privacy and security of personal information. AI should comply with privacy and data protection laws to ensure that personal data is processed lawfully.

A number of the typical AI characteristics may appear to be at odds with the underlying principles of data protection law, including the principles of transparency, data minimisation and accountability:

- Transparency – it can be difficult for organisations to explain AI systems to individuals because of their inherent complexity.
- Data minimisation – are you striking the balance between data minimisation and statistical accuracy?
- Accountability – can you demonstrate compliance with GDPR and other data protection principles?
- Cross-border considerations – where is the data being processed? Can this be identified? Is it all UK-based?

## Discrimination and bias

AI systems can perpetuate and amplify existing biases and prejudices, which could result in discriminatory practices. Addressing these issues requires a legal framework that addresses issues of fairness and transparency.

The Equality and Human Rights Commission (EHRC) has drawn attention to the fact that there is a real risk posed by the use of AI, with biases within the systems often stemming from its use of training data.

### Imbalanced training data:

- Imbalanced training data can lead to discriminatory results. For example, if men are over-represented in the training data, women are statistically 'less important' - this may impact the results of AI (e.g. by suggesting men are more likely to repay loans, if more men are represented).
- These issues will apply to any population under-represented in the training data. For example, if a facial recognition model is trained on a disproportionate number of faces belonging to a particular ethnicity and gender, it will perform better when recognising individuals in that group and worse on others.
- It may be possible to balance it out by adding or removing data about under/ overrepresented subsets of the population.

### Training data could reflect past discrimination:

- For example, if, in the past, loan applications from women were rejected more frequently than those from men due to prejudice, then any model based on such training data is likely to reproduce the same pattern of discrimination.
- You could either modify the data, change the learning process, or modify the model after training.

## Governance and regulation

The rapid development of AI requires an adequate legal framework to regulate its use, development, and deployment. The regulatory framework should balance innovation and safety while also promoting ethical practices. It's a situation that's evolving rapidly, with the UK government playing a particularly high-profile role.

AI is ever-changing and quickly adapting, so regulation is often trying to catch up. This means regulation can often fall behind the AI, and it can become unclear which regulations apply, whether new regulations will come into force etc.

### UK - Current legislation

Currently, in the UK, AI is governed by numerous different bits of legislation (data protection law, Equality Act 2010, product safety laws, consumer rights law, tort law, financial services regulation etc.), so it can be difficult to ascertain what is applicable and whether you are abiding by it.

#### White paper on regulating AI

The government recently (29 March 2023) released a white paper discussing the regulation of AI and proposing a change in regulatory approach.

It acknowledged industry concerns "that conflicting or uncoordinated requirements from regulators create unnecessary burdens and that regulatory gaps may leave risks unmitigated, harming public trust and slowing AI adoption.".

It proposes a principles-based framework for regulators to interpret and apply to AI within their remits. The proposed principles are pro-innovation, proportionate, trustworthy, adaptable, clear and collaborative.

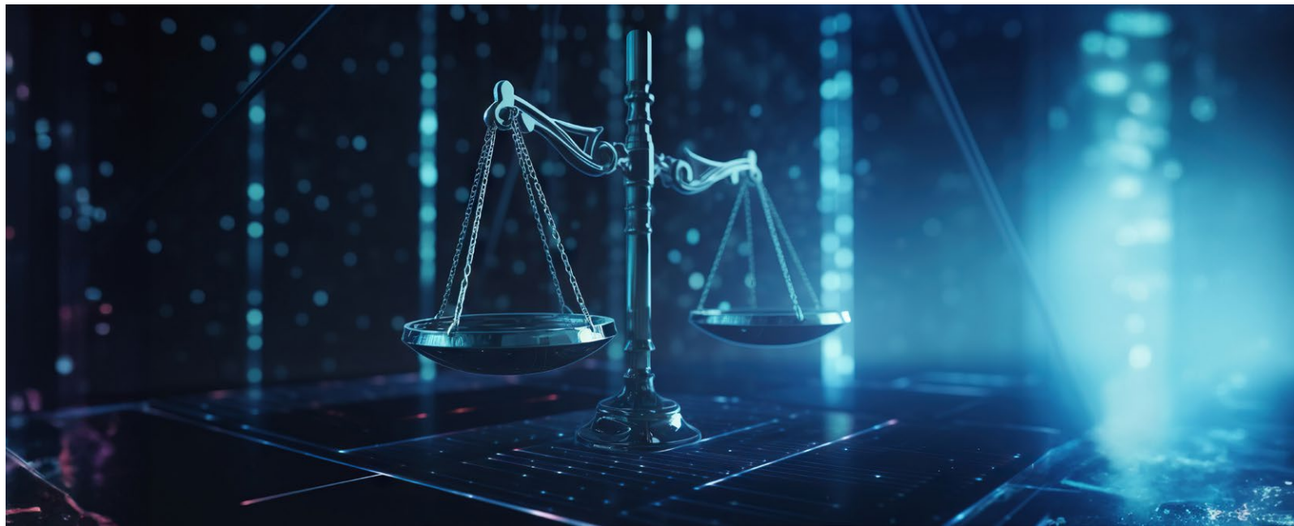
A strength of this approach is that regulators would still be able to exercise discretion and expert judgement regarding the relevance of each principle to their individual domains. Initially, the principles will be issued by the government on a non-statutory basis and applied by regulators within their remits.

Following a period of non-statutory implementation, and when parliamentary time allows, the government anticipates that it will want to strengthen and clarify regulators' mandates by introducing a new duty requiring them to have due regard to the principles.

The Government has also just published the Terms of Reference for a working group the role of which will include **"identifying, developing and codifying good practice on the use of copyright, performance and database material in relation to AI, including data mining"**.

G7 leaders are calling for some international standards around artificial intelligence (generative AI in particular).

In summary: the law is playing catch up with the development of AI. There are hefty issues to consider in terms of ownership, bias, copyright and IP protection, and regulation, and clarity will come from legal test cases in the coming months and years.



## Application to Development and Testing

The wider issue of Governance and Regulation – while doubtless important – is less applicable to the specific uses of AI for software development that we have been looking at as part of this project. Similarly, discrimination, bias and the processing of personal data within AI systems has the potential to have hugely negative effects on society as the use of such tools become more widespread — but again is beyond the scope of this work.

For the purposes of this project, we have therefore focussed on the issues of IP ownership/infringement and legal liability by reference to the terms of use of the main tools explored by the team: Chat GPT, AutoGPT, Copilot and Tabnine. We have also looked at the terms of use for some of the QA and testing tools that the team have been trialling.

### Chat GPT, AutoGPT, CoPilot and Tabnine

The first thing to note is that there are variations in approach, even across just these three tools. This is not a problem per se, but a developer that uses multiple tools in a single project is potentially introducing a high degree of legal complexity should any issue or claims arise.

For example even as regards jurisdiction:

- AutoGPT makes use of OpenAI's GPT-4 language model and both it and Chat GPT are covered by OpenAI's terms. Those terms are governed by Californian law and require mandatory arbitration to resolve disputes arising from them;
- Copilot's terms are governed by the law of British Columbia, Canada and again require arbitration of disputes under Canadian rules; and
- Tabnine's terms are governed by Israeli law and users are quired to submit to the jurisdiction of the courts of Tel Aviv.

**Liability**

The overarching theme is that, as one would expect, the tools all go to huge lengths to exclude any liability whatsoever arising from their use and/or cap that liability at a low level.

In the case of OpenAI and Copilot, the terms state that the services are provided "AS IS", which is language recognised under US law as excluding all implied warranties through which the law might otherwise protect a buyer. All three sets of terms also include broad exclusions of liability and indemnities in favour of the provider for loss and damage arising from customers' use.

As an experimental open-source application that makes use of (and requires a subscription to) GPT-4, AutoGPT's GitHub repository includes an overlaid disclaimer, which states that it:

> *is an experimental application and is provided "as-is" without any warranty, express or implied. By using this software, you agree to assume all risks associated with its use, including but not limited to data loss, system failure, or any other issues that may arise.*

In addition:

- OpenAI's terms limit its liability to the greater of: the amount the user paid for the service that gave rise to the claim during the 12 months before the liability arose; or one hundred dollars ($100); and
- Copilot's total aggregate liability from any and all claims is limited to the total amount of fees paid by the customer in the 12 months immediately preceding the date the cause of action first arose.
- In all three cases the terms seek to impose a 12-month limitation on bringing claims arising from their use; and OpenAI and CoPilot also seek to exclude users from joining class actions relating to the services.

Tabnine is the odd one out insofar as it does give specific reassurance to users concerning the open source software that is uses to deliver code, saying:

> *We note that as part of the development of the Services provided by Codotoa, Tabnine uses certain "Free and Open Source Software '' or "FOSS". In that respect Tabnine represents that its use of such FOSS is in compliance with the licence terms thereof (however Tabnine makes no other representations and/or warranties in connection with such FOSS.*

**Intellectual Property**

All the platforms are protective of their own technologies, expressly prohibiting the use of scraping/spidering technology to "steal" their underlying data. However, they take slightly differing approaches to their outputs and to what they will do with data uploaded by users.

OpenAI assigns all its right title and interest in its outputs to the user. However, it also makes clear that an output may not be unique and, therefore, that a user does not acquire exclusive rights to particular outputs or solutions;

Copilot retains all rights but instead grants the user a licence to outputs. It also states that customer grants Copilot:

- an irrevocable, perpetual licence to use or incorporate into the service any suggestions, enhancement requests, recommendations or other feedback provided by customer; and
- a worldwide, royalty-free, non-exclusive, irrevocable licence to reproduce, process and display the customer's data in an aggregated and anonymized format for Copilot's internal business purposes, including without limitation to develop and improve the service, the system and Copilot's other products and services.

Tabnine also grants users a licence to outputs, rather than assigning rights. However, it does provide reassurance that if users give Tabnine permission to access their code for analysis and "Tailor Made Services":

> *such code shall be used by Tabnine solely in order to adjust and upgrade the standard Services to provide you, and you only, with the Tailor Made Services. No other users shall be granted with any access to the Tailor Made Services provided to you, …[and]…, any code provided by you to Tabnine shall not be stored and/or used by Tabnine, and (d) for the avoidance of doubt, except with respect to creating the Tailor Made Services, Tabnine shall not be granted any intellectual property rights in the code shared by you which was provided solely for the limited use by Tabnine for creating the Tailor Made Services*

**QA Testing Tools**

In addition to these development tools we have also looked at some of the QA and testing tools used by the team, namely:

- Mabl
- Perfecto
- Functionize
- Virtuoso
- Applitools
- Appvance

These tools are, to an extent, different to Chat GPT, AutoGPT, Copilot and Tabnine as their focus is more on testing code than creating it. However, from a legal perspective their user terms or similar. They seek to prevent users from reproducing the tools or using them for unauthorised purposes, and seek as far as possible to limit liability arising from their use.

They do, though, generally recognise the distinction between: ( | ) users' own data that they upload into the tool; and ( || ) the results/reports that the tools produce.

## Conclusions

In summary, therefore, the legal relationship between AI platforms and users is generally one-sided and (even from the traditionally risk-averse point of view of a lawyer) the advice is best summed up as "proceed with caution".

Of the development tools we have looked at Tabnine seems to have the clearest and best-drafted terms for software development. It also appears to have the lowest risk-profile, making a selling point of the fact it uses properly licensed open-source software as its sole training source.

There have already been some very high-profile cases in this area, most notably the ongoing US claim involving Copilot in the US, in which Microsoft and GitHub are defending allegations that Copilot infringes the copyright in a large number of publicly-available source-code repositories on which it has been trained. That claim may, from a US perspective at least, provide some helpful clarity as to how the courts might treat some of these issues, but in the meantime the complexity, cost and potential jurisdictional issues involved in such claims will make it difficult for any single developer to seek redress should liability arise from their use of AI platforms.

That being the case developers should take what steps they can to control their "downstream" risk. In particular, it is usual for clients to require warranties around the code that is delivered to them – including as to authorship and whether it includes open-source code.

Using AI tools can almost immediately cut across those warranties, and it is interesting to note that OpenAI's terms specifically state that it is a breach to represent that output from the service was human-generated when it is not. It would be advisable, therefore, for developers to look at the terms of their customer contracts to be clear about the warranties they are offering and, if they have not done so already, make specific allowance for the use of AI platforms.

As regards testing tools, the risks seem on the whole slightly lower, but it is still really important to be clear about which tools you are using and to check that their terms are compatible with your use.

Clearly these are incredible tools, but they do pose some very specific risks for developers. And just as the technical members of Zenitech's team have highlighted the need to review carefully the outputs from AI platforms before putting them into use, the same is doubtless true from a legal perspective.


# Demystification: a glossary of AI terms

**Christopher Lacy-Hulbert**
*Founder and CTO at Zenitech*



In our ‘New Beings’ series, we explore the vast and complex area of AI as it relates to Software Development. There are so many new words and terms used in relation to AI, so we’ve put together a handy glossary of the most common-used terms.

**I hope it’s helpful.**

| | |
|---|---|
| **Generative AI** | A subfield of artificial intelligence focused on creating new content or predictions. It uses algorithms and models, often based on machine learning techniques, to generate outputs such as text, images, music, and more. |
| **Large Language Models (LLMs** | These are AI models trained on vast amounts of text data. They predict or generate new text based on input by learning patterns in the data they’ve been trained on. Examples include GPT-3, GPT-4, BERT, etc. |
| **GPT (Generative Pretrained Transformer)** | A type of LLM developed by OpenAI. It’s trained to predict the next word in a sentence and can generate human-like text based on a given prompt. |
| **BERT (Bidirectional Encoder Representations from Transformers)** | A model developed by Google that’s designed to understand the context of words in a sentence by looking at the words before and after it. |
| **Transfer Learning** | A machine learning method where a pre-trained model is adapted for a second, related task. This approach saves resources as it requires less data and computational time. |
| **Fine-tuning** | The process of tweaking a pre-trained model for a specific task. It involves adjusting the model’s parameters to optimise its performance on the new task. |
| **Transformer Models** | A type of deep learning model introduced in the paper “Attention is All You Need”. It utilises self-attention mechanisms and has been highly influential in NLP tasks. |
| **Generative Adversarial Networks (GANs)** | A type of generative model consisting of two neural networks – a generator and a discriminator. The generator creates new data samples, such as images, by learning to mimic the distribution of the training data, while the discriminator learns to differentiate between real and fake samples. The two networks are trained together in a competitive process until the generator can create realistic samples that can fool the discriminator. |
| **Autoencoders** | Autoencoders are a type of neural network that are used for unsupervised learning. They are designed to encode input data into a lower-dimensional representation and then decode it back to its original form. Autoencoders can be used for tasks such as image compression, denoising, and image generation. |
| **Variational Autoencoders (VAEs)** | VAEs are a type of autoencoder that uses a probabilistic approach to generate new data samples. They learn a probabilistic distribution over the latent space and use it to generate new samples that are similar to the training data. |
| **Recurrent Neural Networks (RNNs)** | RNNs are a type of neural network that are designed to process sequential data, such as text or speech. They use feedback loops to process the input data one step at a time and maintain a memory of the previous steps. RNNs can be used for tasks such as language translation, text generation, and speech recognition. |
| **Transformers** | Transformers are a type of neural network architecture that are designed to process sequences of data, such as text or speech. They use attention mechanisms to focus on the relevant parts of the input sequence and process them in parallel, making them more efficient than RNNs for long sequences. |

# Accelerating iOS development with AI tools: a practical exploration with ChatGPT and Copilot X



**Péter Sebestyén Krassay**
*Expert iOS Developer at Zenitech*

This article is part of the 'New Beings' series of articles from Zenitech and Stevens & Bolton, examining the practicalities, issues and possibilities of using AI in development.

Today, we explore the practicalities of iOS development using AI.

## Introduction

In this article, we delve into the coding phase, focusing on how AI tools, specifically ChatGPT and Copilot X, can accelerate iOS development. Having previously looked at the legal issues relating to AI, we can now explore how to harness the capabilities of these AI tools in building iOS applications.



## Tools

We'll focus on two AI tools in this article.

1. **ChatGPT (GPT-4)**: A state-of-the-art Large Language Model (LLM) by OpenAI, ChatGPT is built on the GPT-4 architecture, specialising in generating human-like text and synthesising code.

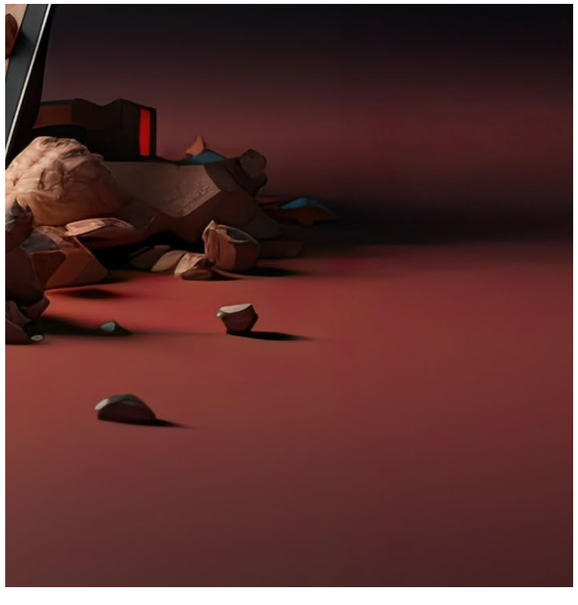
2. **Copilot X**: An AI code assistant based on a custom LLM, Copilot X provides context-aware code suggestions, optimising coding workflows. (Currently in private **beta**.)

The legal issues around the use of these tools are the subject of a dedicated article in this series, New Beings, Legal Issues and AI.

## Overview of the task

For this test of AI tools, our iOS developers at Zenitech worked on an application for a fictional airport. The task was to create a feature-rich iOS app that integrates multiple screens, network communication to fetch real-time data, and local persistence for user-specific information.

Through the course of development, we employed ChatGPT and Copilot X to expedite the coding process by creating Data Transfer Objects (DTOs), fetching flight data, and generating relevant UI code.



## The fictional 'client': SZIA

The South-Zubony International Airport (SZIA), an imaginary airport located in Hungary, needs an application to improve the travel experience of passengers. The core purpose is to provide users with an intuitive and efficient way to access real-time flight information, airport news, maps, and a platform for submitting complaints. The scope of the application includes network communication for real-time data retrieval, local data persistence, multiple views for different features, and user interactions.

### Functionalities: Arrivals, Departures, Complaints, Favourites

The application requires the following primary functionalities:

- **Arrivals:** This screen showcases a list of arriving flights with the option for users to refresh the data.
- **Departures:** Similar to arrivals, this screen displays a list of departing flights.
- **Complaints:** Users can submit complaints through a form, which can include a text description and an optional image attachment.
- **Favourites:** Users have the option to mark specific flights as favourites, which are then saved locally on the device for easy access.

These functionalities aim to provide a comprehensive travel tool to passengers, helping them stay informed and engaged throughout their journey.

### Backend Integration

The backend system for the SZIA application is a REST API that provides endpoints for retrieving real-time flight information, news, and submitting complaints. The backend is accessible at https://szia-backend.autsoft.hu/api/, and is crucial in supplying the data needed for the application's features. The endpoints include

- GET /Airlines,
- POST /Complaints,
- **GET /Flights**, and
- GET /News.

#### Introduction to the OpenAPI descriptor

An OpenAPI descriptor is provided for the backend, which is a specification document written in YAML or JSON (interchangeable). This descriptor provides a standardised description of the API's endpoints, requests, responses, and other components.

By using this OpenAPI descriptor, developers can understand the API's capabilities and generate client libraries, server stubs, API documentation, and other code artefacts. This proves to be particularly beneficial when integrating ChatGPT and Copilot X, as it allows these AI tools to have a better understanding of the API structure and generate more accurate code.

## Setting Up Development Environments

### ChatGPT 4

Integrating ChatGPT into your development workflow is a relatively straightforward process. ChatGPT doesn't require any special software installation; all you need is a functional web browser.

- ChatGPT is accessible through web browsers and doesn't necessitate a specific development environment.
- Create an account on OpenAI's platform if you don't have one already.
- Subscribe to ChatGPT Plus, which costs $20 per month. ChatGPT Plus is a subscription plan that offers a host of benefits including general access to ChatGPT even during peak times, faster response times, and priority access to new features and improvements (i.e. **GPT-4**).

Once you have subscribed to ChatGPT Plus, you can start using it (through your browser) to assist in generating code snippets, brainstorming ideas, or answering queries related to (iOS) development.

### Copilot X

You need to fulfil certain prerequisites to harness the capabilities of Copilot X within Visual Studio Code - Insiders:

- Ensure you have a GitHub account. (If not, you can create one here.)
- Subscribe to GitHub Copilot, which is priced at $100 per year. You can subscribe to GitHub Copilot here.
- Request access to the Copilot X private beta by visiting this page. It's important to note that access to Copilot X is granted on an invitation basis, and you may have to wait for approval.
- Once you have access to Copilot X, download and install Visual Studio Code - Insiders edition from this link.
- After installing Visual Studio Code - Insiders, launch the application and go to the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of the window.
- Search for "GitHub Copilot Nightly" and install the extension. This extension is the nightly build of GitHub Copilot and includes the most recent features.
- Additionally, search for "GitHub Copilot Chat" (Preview) and install this extension as well. This extension allows you to chat with Copilot X and get real-time coding assistance.
- Activate these extensions by login into your GitHub account.
- After installing the extensions and successfully activating them, you will notice a new icon appearing in the sidebar. This is where you can engage in a chat with Copilot X, ask questions, and receive code suggestions.

With Copilot X integrated into Visual Studio Code - Insiders, you are now equipped with a powerful AI assistant that can offer intelligent code suggestions and accelerate your (iOS) development process.

### Copilot for Xcode - A Worthy Mention

While our experimentation primarily revolves around the integration of ChatGPT and Copilot X in Visual Studio Code - Insiders, there is another tool worth mentioning for the Xcode environment. Copilot for Xcode is an innovative extension that integrates the functionalities of ChatGPT and GitHub Copilot directly into Xcode.

This open-source extension operates by making API calls to offer real-time, AI-driven code suggestions. Kudos to the developers for actively contributing to this project!

However, it's important to note that Copilot for Xcode uses API keys for its functionality. In the context of this article, integrating it wasn't feasible as I didn't have access to GPT-4 via API, and Copilot X also employs a different language model compared to Copilot, which would have made the comparison less equitable.

But Copilot for Xcode is definitely worth exploring. Even if just for experimentation, integrating it into Xcode could offer valuable insights into how AI-driven code suggestions can augment the development process.

While Copilot X offers excellent integration within Visual Studio Code, it is being developed by Microsoft and lacks the capability to compile Xcode projects.

Copilot for Xcode presents an exciting opportunity for those eager to see AI in action within the Xcode environment.


## Generating Networking Layer with AI Tools

First, let's dive into creating the networking layer for our app. The network layer is what allows us to get and update data for our app. We'll be using ChatGPT and Copilot X to speed up this task.

Please note that I've omitted some of the entities for the sake of brevity where I haven't explicitly stated otherwise.

```
"Airline": {
  "properties": {
    "imageUrl": {
      "type": "string"
    },
    "name": {
      "type": "string"
    },
  "airlineCode": {
      "type": "string"
    },
    "id": {
      "type": "number",
      "format": "double"
    }
},
  "additionalProperties": false
},
"Flight": {
  "properties": {
    "flightNumber": {
      "type": "string"
},
"departure": {
  "type": "string"
},
"arrival": {
  "type": "string"
},
"departureCity": {
  "type": "string"
},
"departureCode": {
  "type": "string"
},
"arrivalCity": {
  "type": "string"
},
"arrivalCode": {
  "type": "string"
},
"departureTime": {
  "type": "string",
  "format": "date-time"
},
"arrivalTime": {
  "type": "string",
  "format": "date-time"
},
"status": {
  "type": "string"
},
"checkinDeskNumber": {
  "type": "number",
  "format": "double"
},
"gateNumber": {
  "type": "number",
  "format": "double"
},
"delay": {
  "type": "number",
  "format": "double"
},
"comment": {
  "type": "string"
},
"id": {
  "type": "number",
  "format": "double"
},
"airlineId": {
  "type": "number",
  "format": "double"
}
},
"required": [
  "flightNumber",
  "departure",
  "arrival",
  "departureTime",
  "arrivalTime",
  "status",
  "checkinDeskNumber",
  "gateNumber",
  "delay"
],
"additionalProperties": false
}
```

## Feeding the OpenAPI JSON to ChatGPT - Models

I gave a snippet of the OpenAPI definition to ChatGPT which describes the following models: Airline, Flight and asked it to generate Swift network models for my iOS application. Using the information from the OpenAPI definition, here is the first version of the generated Swift models for these entities.

**Airline.swift**

```
struct Airline: Codable {
  let imageUrl: String?
  let name: String?
  let airlineCode: String?
  let id: Double?
}
```

**Flight.swift**

```
struct Flight: Codable {
  let flightNumber: String
  let departure: String
  let arrival: String
  let departureCity: String?
  let departureCode: String?
  let arrivalCity: String?
  let arrivalCode: String?
  let departureTime: String
  let arrivalTime: String
  let status: String
  let checkinDeskNumber: Double
  let gateNumber: Double
  let delay: Double
  let comment: String?
  let id: Double?
  let airlineId: Double?
}
```

- **Codable**: Each struct is marked with the Codable protocol, which permits both encoding to and decoding from a serialised format like JSON.
- **Optional vs. Non-Optional Properties**: Properties that are not explicitly required are declared as optional in Swift (appending a ? to the type).
- **Type Consistency**: The consistency with types are maintained (except for dates) as described in the OpenAPI document. For instance, properties defined as "type": "number", "format": "double" were declared as Double in Swift.
- **Date Handling**: Properties such as departureTime and arrivalTime are currently strings, which is not ideal for date manipulation. This should be improved by using a more specialised type: Date.
- **Equatable**: The structs are not marked as Equatable, which is a protocol that allows us to compare two instances of the same type for equality. This is a useful feature to have when writing unit tests.

Telling ChatGPT to generate the models again taking into consideration the date-time format and also adding Equatable protocol conformance, I got the following updated versions of the models

```
struct Airline: Codable, Equatable {
  let imageUrl: String?
  let name: String?
  let airlineCode: String?
  let id: Double?
}
```

```
struct Flight: Codable, Equatable {
  let flightNumber: String
  let departure: String
  let arrival: String
  let departureCity: String?
  let departureCode: String?
  let arrivalCity: String?
  let arrivalCode: String?
  let departureTime: Date
  let arrivalTime: Date
  let status: String
  let checkinDeskNumber: Double
  let gateNumber: Double
  let delay: Double
  let comment: String?
  let id: Double?
  let airlineId: Double?
}
```

I'm extremely satisfied with the results - the models are finely crafted, it's as if I had written them by hand or employed a finely tuned custom OpenAPI generator

## Feeding the OpenAPI JSON to Copilot X - Models

Having swagger.json open in Visual Studio Code - Insiders edition, I invoked Copilot to generate the models of the networking layer. Copilot X generated the following code.

```
struct SziaUserRequest: Codable {
  let username: String
  let email: String
  let id: Double
  let sziaUserId: Double
}

struct SziaUser: Codable {
  let username: String
  let email: String
  let id: Double
}
```

- **Codable:** Each struct is marked with the Codable protocol, which permits both encoding to and decoding from a serialised format like JSON.
- **Hallucinogenic entity:** The SziaUserRequest struct is not even present in the OpenAPI definition.
- **Optional vs. Non-Optional Properties:** Properties that are not explicitly required are NOT declared as optional.
- **Missing entities:** Copilot X did not generate models for the other entities such as Airline, Flight, etc.

Having swagger.json open in Visual Studio Code - Insiders edition, I explicitly selected the definition for the entities I wanted to generate, I invoked Copilot X to generate the models of the networking layer with specific instructions:

> *Can you please generate Codable, Equatable Swift struct from the selected OpenAPI model definitions from swagger.json? Take into consideration that some properties are optional and that format: date-time should be a Date property.*

```
struct Airline: Codable, Equatable {
  let id: Double?
  let name: String?
  let airlineCode: String?
  let imageUrl: String?
}

struct Flight: Codable, Equatable {
  let flightNumber: String
  let departure: String
```

```
    let arrival: String
    let departureCity: String?
    let departureCode: String?
    let arrivalCity: String?
    let arrivalCode: String?
    let departureTime: Date
    let arrivalTime: Date
    let status: String
    let checkinDeskNumber: Double
    let gateNumber: Double
    let delay: Double
    let comment: String?
    let id: Double?
    let airlineId: Double?
}
```

This time I've got similar output to ChatGPT and Copilot X has generated the models for all the entities.

## Feeding the OpenAPI JSON to ChatGPT - Request/Response

I've told ChatGPT to generate the networking code for the GET `Flights`` endpoint without any other specific instructions, I've just given it the OpenAPI specification of that endpoint. Here is the result.

```
import Foundation

struct Flight: Decodable {
    // Assuming there are properties defined under '#/definitions/Flight'
}

class FlightAPI {
    let baseURL = "https://api.imaginaryairport.com"

    func getFlights(filter: String?, completion: @escaping (Result<[Flight], Error>) -> Void) {
        var urlString = "\(baseURL)/Flights"

        if let filter = filter {
            urlString.append("?filter=\(filter.addingPercentEncoding(withAllowedCharacters: .urlHostAllowed) ?? "")")
        }

        guard let url = URL(string: urlString) else {
            completion(.failure(NSError(domain: "", code: -1, userInfo: [NSLocalizedDescriptionKey: "Invalid URL"])))
            return
        }

        URLSession.shared.dataTask(with: url) { data, response, error in
            if let error = error {
                completion(.failure(error))
                return
            }

            guard let data = data else {
                completion(.failure(NSError(domain: "", code: -2, userInfo: [NSLocalizedDescriptionKey: "No data received"])))
                return
            }

            do {
                let flights = try JSONDecoder().decode([Flight].self, from: data)
                completion(.success(flights))
            } catch {
                completion(.failure(error))
            }
        }.resume()
    }
}
```

- **Base URL**: The generated code has a base URL property, which is a good practice, I just had to add it by hand pointing to the SZIA server.
- **Error Handling:** The generated code has a decent error handling. It checks for invalid URL, absence of data, and decoding errors, which are essential for stability. It doesn't account for specific HTTP status codes or domain-specific errors, but these aren't included in the OpenAPI specification either. We don't usually use NSErrors, because we have Swift's Error protocol to conform to and create meaningful errors.
- **Multithreading:** The network request is made asynchronously using `URLSession.shared.dataTask(with:completion:)``. However, the completion handler is not explicitly dispatched to any queue which means it will run in the background thread. In most cases, you would want the completion handler to be executed on the main thread if it involves UI updates.
- **Working code?**: The generated code is not working code. It results in a DecodingError for departureTime. (This is because this assignment doesn't respect ISO8601 standard for date-time format on purpose as a gotcha for candidates.)

Giving the Flight object and the error message to ChatGPT, it generated a working fix and also dispatched the result to the main thread (Except in case of failure).

```
...
    do {
        let decoder = JSONDecoder()
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSZ"
        decoder.dateDecodingStrategy = .formatted(dateFormatter)
        let flights = try decoder.decode([Flight].self, from: data)
        DispatchQueue.main.async {
            completion(.success(flights))
        }
    } catch {
        completion(.failure(error))
    }
...
```



I'm quite pleased with how the AI-generated code performed in handling the network request. But, I'm curious to explore if we can further refine and modernise this implementation by leveraging newer APIs such as Swift's Combine framework or async/await, which can provide more elegant and streamlined solutions compared to the traditional completion handlers.

```
import Foundation

enum FlightAPIError: Error {
    case invalidURL
    case failedRequest
    case decodingError
}

struct FlightAPI {

    let baseURL: String

    func fetchFlights(filter: String? = nil) async throws -> [Flight] {
        var urlComponents = URLComponents(string: baseURL + "/Flights")
        if let filter = filter {
            urlComponents?.queryItems = [URLQueryItem(name: "filter", value: filter)]
        }

        guard let url = urlComponents?.url else {
            throw FlightAPIError.invalidURL
        }

        let (data, response) = try await URLSession.shared.data(from: url)

        guard let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 200 else {
            throw FlightAPIError.failedRequest
        }

        do {
            let flights = try JSONDecoder().decode([Flight].self, from: data)
            return flights
        } catch {
            throw FlightAPIError.decodingError
        }
    }
}
```

- **Modern Swift Concurrency**: The generated code is using the new async/await APIs and it's also handling errors as I've requested.
- **Working code?**: The generated code is not working code. It results in the same DecodingError for departureTime. ChatGPT didn't remember the fix it generated for the previous code.

However, by giving the underlying decoding error to ChatGPT again, it generated a working fix.

```
...
    do {
      let decoder = JSONDecoder()
      let dateFormatter = DateFormatter()
      dateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSZ"
      dateFormatter.timeZone = TimeZone(secondsFromGMT: 0)
      decoder.dateDecodingStrategy = .formatted(dateFormatter)
      let flights = try decoder.decode([Flight].self, from: data)
      return flights
    } catch {
      print("Decoding error: \(error)")
      throw FlightAPIError.decodingError
    }
...
```

ChatGPT handled the generation of network requests commendably, demonstrating an impressive capacity to adapt to requirements such as custom date formatting.

## Feeding the OpenAPI JSON to Copilot X - Request/Response

I've also tried to feed the OpenAPI JSON to Copilot X to see how it would perform. I've used the same OpenAPI JSON file as before without giving specific instructions about how I want the networking code generated. Copilot X gave me the following code

```
guard let url = URL(string: "https://example.com/api/Flight") else {
    fatalError("Invalid URL")
}

var request = URLRequest(url: url)
request.addValue("application/json", forHTTPHeaderField: "Content-Type")

let task = URLSession.shared.dataTask(with: request) { data, response, error in
    guard let data = data, error == nil else {
        print("Error: \(error?.localizedDescription ?? "Unknown error")")
        return
    }

    do {
        let flights = try JSONDecoder().decode([Flight].self, from: data)
        print("Flights: \(flights)")
    } catch {
        print("Error decoding response: \(error.localizedDescription)")
    }
}

task.resume()
```

- **Base URL**: The generated code doesn't use a base URL property. It's not a big deal here as the URL is hardcoded. However, it's a good practice to have a base URL property in case of dynamic URLs.

- **Error Handling**: The generated code does an ok job for error handling. In case of an invalid URL it crashes the app. This is fine here as this would be a programming error. Other times with dynamic urls, crashing the app is too extreme. 🟡
- **Multithreading**: The network request is made asynchronously using URLSession.shared.dataTask(with:completion:). However, the completion handler is not explicitly dispatched to any queue which means it will run in the background thread. In most cases, you would want the completion handler to be executed on the main thread if it involves UI updates. 🟡
- **Filtering**: The generated code doesn't support filtering. 🔴
- **Working code?**: The generated code is not working code. It results in a DecodingError for departureTime. 🔴

Giving the Flight object and the error message to Copilot X, it also generated a working fix. It is worth noting that I had to explicitly say that the date format is not ISO8601 to get the fix. 🟢

```
let decoder = JSONDecoder()
let dateFormatter = DateFormatter()
dateFormatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
dateFormatter.timeZone = TimeZone(secondsFromGMT: 0)
decoder.dateDecodingStrategy = .formatted(dateFormatter)

do {
    let flights = try decoder.decode([Flight].self, from: data)
    print("Flights: \(flights)")
} catch {
    print("Error decoding response: \(error.localizedDescription)")
}
```

I've also tried to feed Copilot X more specific instructions about how I want the networking code generated. I've asked it to use the new async/await APIs and handle errors. Copilot X gave me the following code:

```
enum FlightError: Error {
    case invalidURL
    case decodingError
}

func fetchFlights() async throws -> [Flight] {
    guard let url = URL(string: "https://example.com/api/Flight") else {
        throw FlightError.invalidURL
    }

    var request = URLRequest(url: url)
    request.addValue("application/json", forHTTPHeaderField: "Content-Type")

    let (data, response) = try await URLSession.shared.data(for: request)

    guard let httpResponse = response as? HTTPURLResponse, httpResponse.statusCode == 200 else {
        throw FlightError.invalidURL
    }

    do {
        let flights = try JSONDecoder().decode([Flight].self, from: data)
        return flights
    } catch {
        throw FlightError.decodingError
    }
}
```

- **Modern Swift Concurrency:** Pretty good! The generated code is using the new async/await APIs and it's also handling errors like I've requested. 🟢
- **Working code?**: The generated code is not working code. It results in a DecodingError for departureTime. It didn't remember the date formatter from the previous example. 🔴

Giving the underlying decoding error to Copilot X, it generated a working fix. 🟢

I'm still overall impressed with Copilot X; when provided with specific instructions, it delivers targeted and quality results.

## Developing User Interface with AI Tools

Next, we'll walk through the process of using AI tools, to develop the user interface of our application. Our focus will be on creating a SwiftUI List that displays flights, using the previously created Flight struct and the related network fetching code.

For our SwiftUI List, the requirements are:

- A simple and visually appealing list.
- Each list item should display data about the specific flight.
- The list should be scrollable and populate dynamically based on the data fetched from the network call.
- Appropriate formatting for the date and time.

### ChatGPT

I've told ChatGPT about my Flight struct and the (`async throws``) network fetching code. I've also told it that I want to use SwiftUI to display a list of these Flights. Here's what it generated:

```
import SwiftUI

struct FlightsListView: View {
    @State private var flights: [Flight] = []
    @State private var errorMessage: String? = nil

    private static let dateFormatter: DateFormatter = {
        let formatter = DateFormatter()
        formatter.dateStyle = .short
        formatter.timeStyle = .short
        return formatter
    }()

    var body: some View {
        NavigationView {
            VStack {
                if let errorMessage = errorMessage {
                    Text(errorMessage)
                        .foregroundColor(.red)
                }

                List(flights) { flight in
                    VStack(alignment: .leading) {
                        Text(flight.flightNumber)
                            .font(.headline)
                        Text("\(flight.departure) - \(flight.arrival)")
                            .font(.subheadline)
                        HStack {
                            Text("Departure:")
                            Text(Self.dateFormatter.string(from: flight.departureTime))
                        }
                        HStack {
                            Text("Arrival:")
                            Text(Self.dateFormatter.string(from: flight.arrivalTime))
                        }
                        Text("Status: \(flight.status)")
                        if let delay = flight.delay, delay > 0 {
                            Text("Delayed by \(delay) minutes")
                                .foregroundColor(.red)
                        }
                        if let comment = flight.comment {
                            Text(comment)
                                .font(.caption)
                        }
                    }
                }
                .navigationTitle("Flights")
            }
            .task {
                do {
                    flights = try await fetchFlights()
                } catch {
                    errorMessage = "Failed to fetch flights: \(error.localizedDescription)"
                }
            }
        }
    }
```

The generated code is pretty good, but it's not perfect and doesn't even compile. Let's go through the issues one by one.

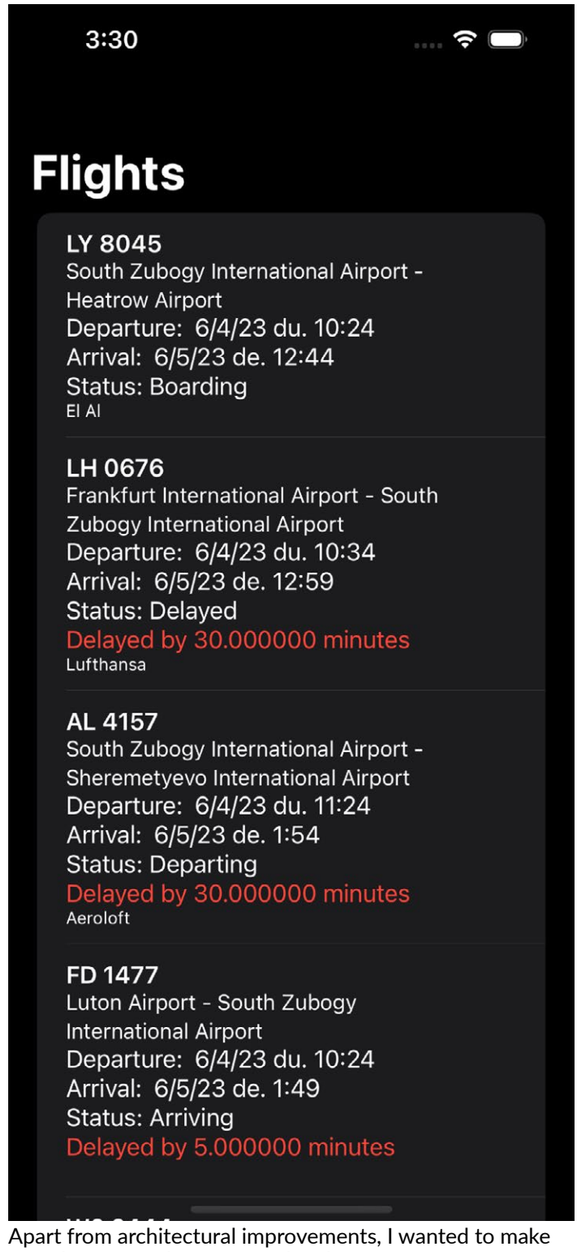
- Flight doesn't conform to Identifiable protocol, so the list doesn't know how to identify each item. 🟡 The fix is pretty easy, either make Flight Identifiable or make it Hashable and pass the \.self`` key path to the list's init for the id parameter. I've chosen the latter not wanting to mess with the Flight` struct.

```
List(flights, id: \.self) { flight in
    ...
}
```

- flight.delay is not optional, so the if let check is unnecessary. 🟡 Similarly it's easy to remedy, just remove the unwrapping.

```
if flight.delay > 0 {
    Text("Delayed by \(flight.delay) minutes")
    .foregroundColor(.red)
}
```

- fetchFlights is defined in FlightAPI. I consider it my fault as I haven't told ChatGPT about it. 🟢 The fix is easy, just instantiate and call FlightAPI.fetchFlights().



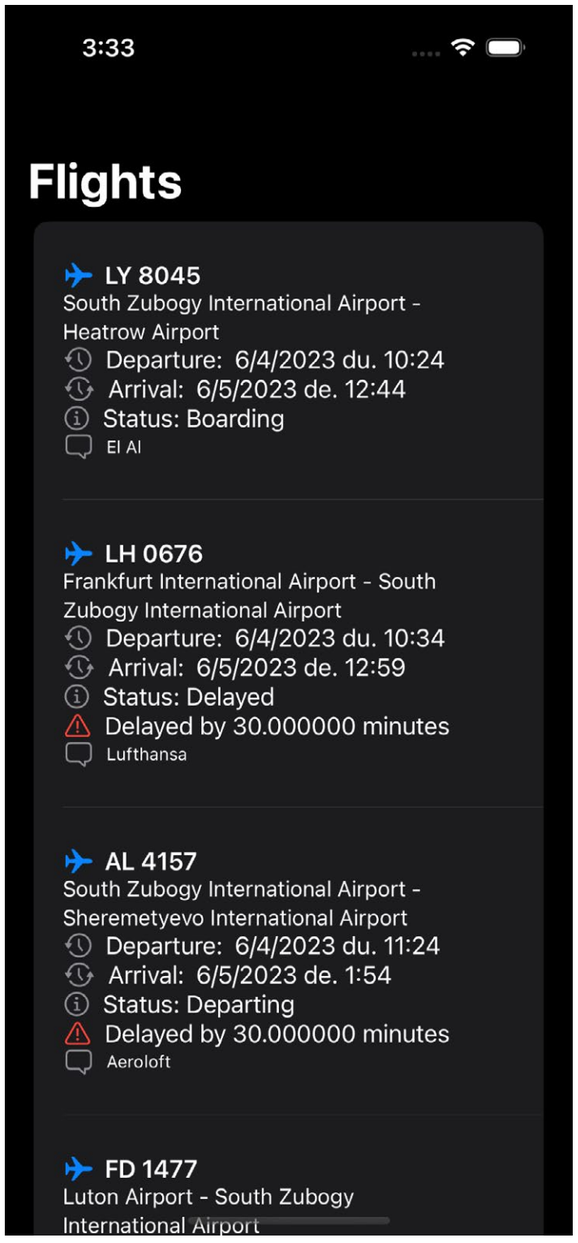

Apart from architectural improvements, I wanted to make two changes to the generated code:

- Don't create a DateFormatter, just use Text's date formatter initialiser.
- Add some images from SF Symbols to make it more visually appealing.

```
struct FlightsListView: View {
  @State private var flights: [Flight] = []
  @State private var errorMessage: String? = nil

  var body: some View {
    NavigationView {
      VStack {
        if let errorMessage = errorMessage {
          Text(errorMessage)
            .foregroundColor(.red)
            .padding()
        }

        List(flights, id: \.self) { flight in
          HStack(alignment: .top) {
            VStack(alignment: .leading) {
              HStack {
                Image(systemName: "airplane")
                  .foregroundColor(.blue)
                Text(flight.flightNumber)
                  .font(.headline)
              }
              Text("\(flight.departure) - \(flight.arrival)")
                .font(.subheadline)
              HStack {
                Image(systemName: "clock.arrow.circlepath")
                  .foregroundColor(.gray)
                Text("Departure:")
                Text(flight.departureTime, format: .dateTime)
              }
              HStack {
                Image(systemName: "clock.arrow.2.circle-
path")
                  .foregroundColor(.gray)
                Text("Arrival:")
                Text(flight.arrivalTime, format: .dateTime)
              }
              HStack {
                Image(systemName: "info.circle")
                  .foregroundColor(.gray)
                Text("Status: \(flight.status)")
              }
              if flight.delay > 0 {
                HStack {
                  Image(systemName: "exclamationmark.
triangle")
                    .foregroundColor(.red)
                  Text("Delayed by \(flight.delay) minutes")
                }
              }
              if let comment = flight.comment {
                HStack {
                  Image(systemName: "bubble.right")
                    .foregroundColor(.gray)
                  Text(comment)
                    .font(.caption)
                }
              }
            }
          }
          .padding(.vertical)
        }
      }
```

```
      .navigationTitle("Flights")
    }
    .task {
      do {
        flights = try await FlightAPI().fetchFlights()
      } catch {
        errorMessage = "Failed to fetch flights: \(error.
localizedDescription)"
      }
    }
  }
}
```

The only minor issue is the incorrect usage of the `Text` initialiser, but it was only a syntax error.



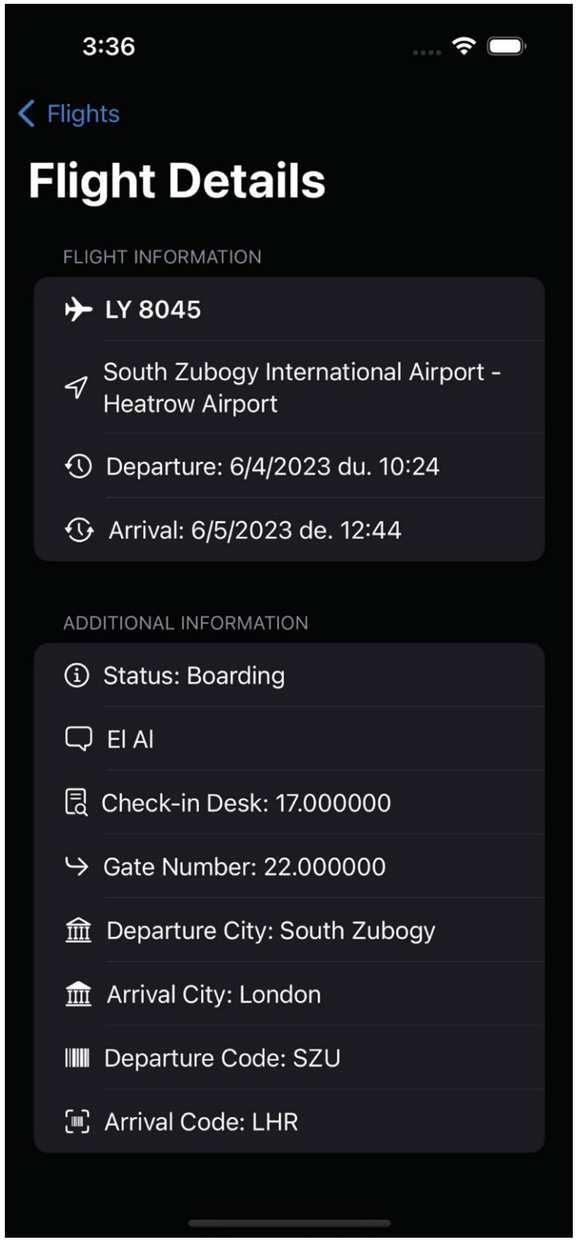

Finally to add a cherry on top, I asked ChatGPT to generate a `FlightDetailView` with all the parameters it can display. It provided me with a nice view with the minor issues of optional handling and date formatting that we've already seen.

```
import SwiftUI

struct FlightDetailView: View {
  let flight: Flight

  private static let dateFormatter: DateFormatter = {
    let formatter = DateFormatter()
    formatter.dateStyle = .short
    formatter.timeStyle = .short
    return formatter
  }()

  var body: some View {
    List {
      Section(header: Text("Flight Information")) {
        HStack {
          Image(systemName: "airplane")
          Text(flight.flightNumber)
            .font(.headline)
        }
        HStack {
          Image(systemName: "location")
          Text("\(flight.departure) - \(flight.arrival)")
        }
        HStack {
          Image(systemName: "clock.arrow.circlepath")
          Text("Departure: \(flight.departureTime, format:
.dateTime)")
        }
        HStack {
          Image(systemName: "clock.arrow.2.circlepath")
          Text("Arrival: \(flight.arrivalTime, format: .date-
Time)")
        }
      }

      Section(header: Text("Additional Information")) {
        HStack {
          Image(systemName: "info.circle")
          Text("Status: \(flight.status)")
        }
        if flight.delay > 0 {
          HStack {
            Image(systemName: "exclamationmark.trian-
gle")
            Text("Delayed by \(flight.delay) minutes")
          }
        }
        if let comment = flight.comment {
          HStack {
            Image(systemName: "bubble.right")
            Text(comment)
          }
        }
        HStack {
          Image(systemName: "doc.text.magnifyingglass")
          Text("Check-in Desk: \(flight.checkinDeskNum-
ber)")
        }
```

```
        HStack {
          Image(systemName: "arrow.turn.down.right")
          Text("Gate Number: \(flight.gateNumber)")
        }

        if let departureCity = flight.departureCity {
          HStack {
            Image(systemName: "building.columns")
            Text("Departure City: \(departureCity)")
          }
        }
        if let arrivalCity = flight.arrivalCity {
          HStack {
            Image(systemName: "building.columns.fill")
            Text("Arrival City: \(arrivalCity)")
          }
        }
        if let departureCode = flight.departureCode {
          HStack {
            Image(systemName: "barcode")
            Text("Departure Code: \(departureCode)")
          }
        }
        if let arrivalCode = flight.arrivalCode {
          HStack {
            Image(systemName: "barcode.viewfinder")
            Text("Arrival Code: \(arrivalCode)")
          }
        }
      }
    }
    .listStyle(.insetGrouped)
    .navigationTitle("Flight Details")
  }
}
```

After hooking it up to the navigation link, we have a simple, but functional app!

```
NavigationLink(destination: FlightDetailView(flight:
flight)) {
...
}
```




## Copilot X

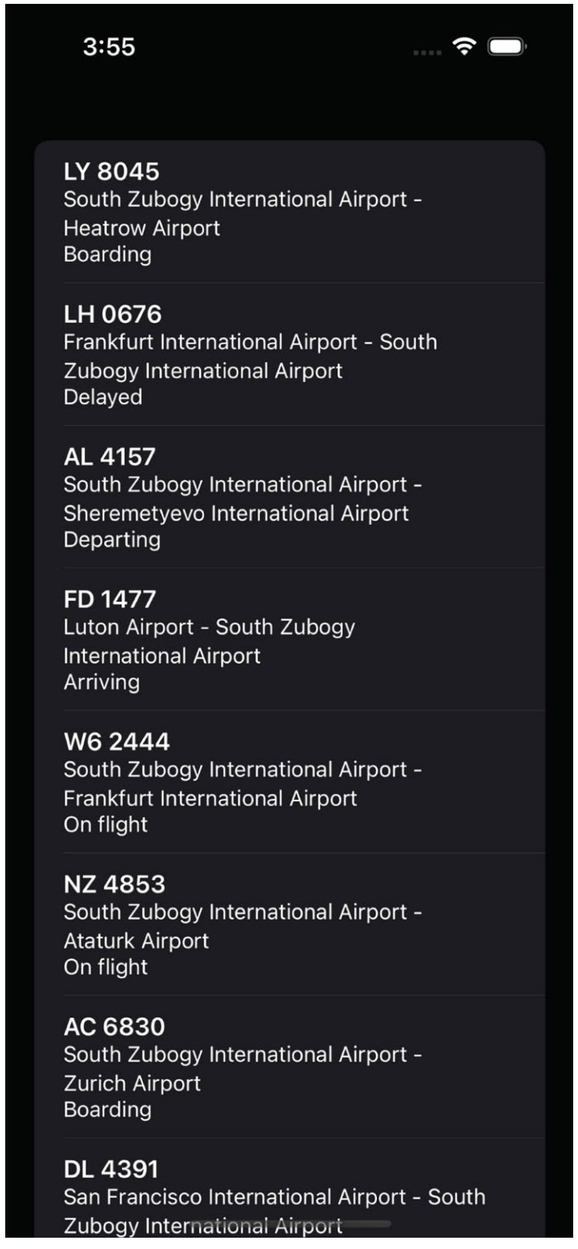
Learning from my mistake from the model generation when trying out Copilot X, I've decided to be specific immediately:

> *I have an array of Flight objects: "struct Flight: Codable, Equatable { let flightNumber: String let departure: String let arrival: String let departureCity: String? let departureCode: String? let arrivalCity: String? let arrivalCode: String? let departureTime: Date let arrivalTime: Date let status: String let checkinDeskNumber: Double let gateNumber: Double let delay: Double let comment: String? let id: Double? let airlineId: Double? }" I would like you to create a visually pleasing SwiftUI List. The Flight object doesn't conform to Identifiable yet and I also want you to use the modern .task API as my fetchFlights API is async.*

```
import SwiftUI

struct FlightsListView: View {
    @State private var flights: [Flight] = []

    var body: some View {
        List(flights) { flight in
            VStack(alignment: .leading) {
                Text(flight.flightNumber)
                    .font(.headline)
                Text("\(flight.departure) - \(flight.arrival)")
                    .font(.subheadline)
                Text(flight.status)
                    .font(.subheadline)
            }
        }
        .task {
            do {
                let fetchedFlights = try await fetchFlights()
                self.flights = fetchedFlights
            } catch {
                print("Error fetching flights: \(error.localized-
Description)")
            }
        }
    }

    func fetchFlights() async throws -> [Flight] {
        // Your async fetchFlights implementation here
    }
}

struct FlightListView_Previews: PreviewProvider {
    static var previews: some View {
        FlightsListView()
    }
}
```

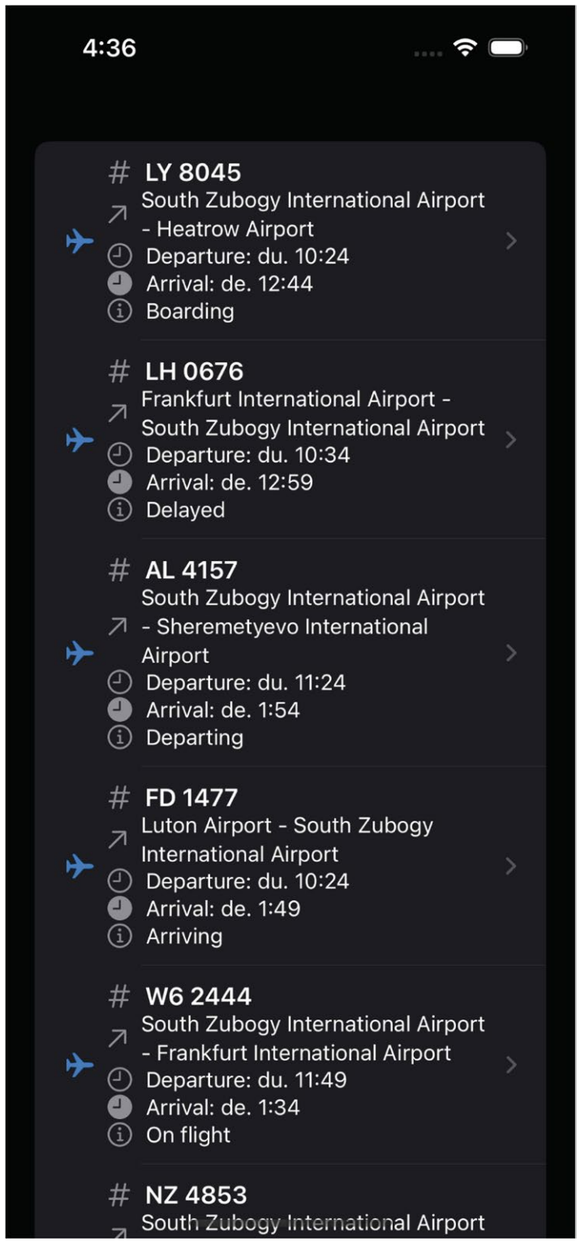
The generated code is mostly fine, but has pretty much the same issues as ChatGPT's code. I had to modify List's init to provide an id and also call the fetchFlights correctly.

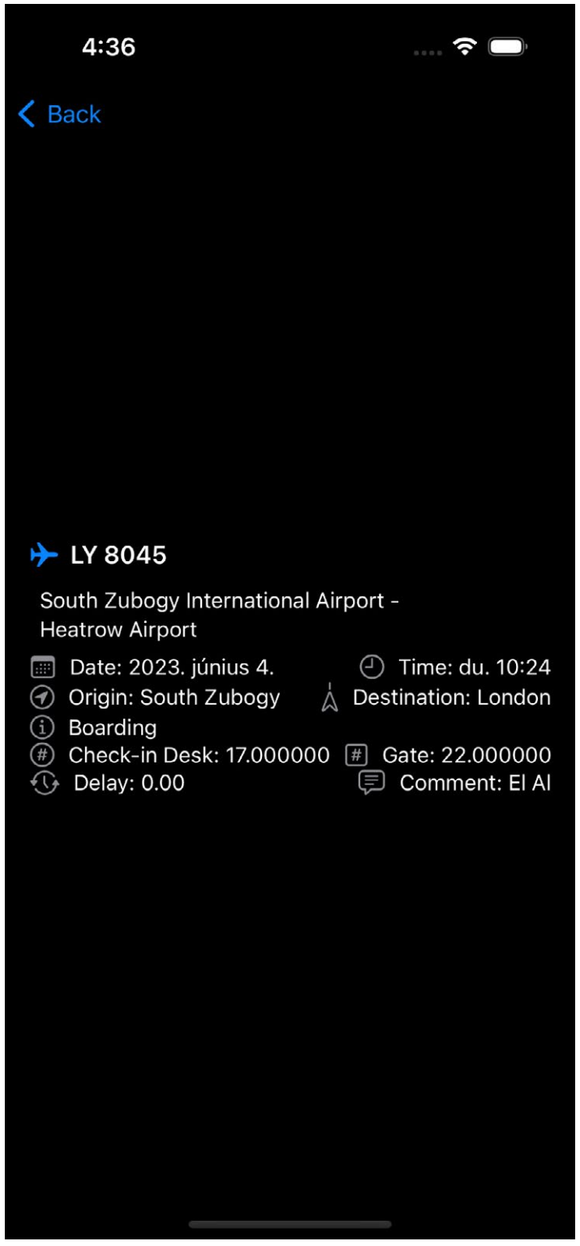





Playing around with Copilot X, I wanted to achieve a similar result as with ChatGPT and show the departure and arrival times, format them using Text's date formatter, add contextually relevant SF Symbol images to the Text's and create the detail view as well

```
import SwiftUI

struct FlightsListView: View {
    @State private var flights: [Flight] = []

    var body: some View {
      NavigationView {
        List(flights, id: \.self) { flight in
            NavigationLink {
              FlightDetailView(flight: flight)
            } label: {
              HStack {
                Image(systemName: "airplane")
                  .foregroundColor(.blue)
                VStack(alignment: .leading) {
                  HStack {
                    Image(systemName: "number")
                      .foregroundColor(.gray)
                    Text(flight.flightNumber)
                      .font(.headline)
                  }
                  HStack {
                    Image(systemName: "arrow.up.right")
                      .foregroundColor(.gray)
                    Text("\(flight.departure) - \(flight.arriv-
al)")
                      .font(.subheadline)
                  }
                  HStack {
                    Image(systemName: "clock")
                      .foregroundColor(.gray)
                    Text("Departure: \(Text(flight.depar-
tureTime, style: .time))")
                      .font(.subheadline)
                  }
                  HStack {
                    Image(systemName: "clock.fill")
                      .foregroundColor(.gray)
                    Text("Arrival: \(Text(flight.arrivalTime,
style: .time))")
                      .font(.subheadline)
                  }
                  HStack {
                    Image(systemName: "info.circle")
                      .foregroundColor(.gray)
                    Text(flight.status)
                      .font(.subheadline)
                  }
                }
              }
            }
        }
        .task {
            do {
                let fetchedFlights = try await FlightAPI().
fetchFlights()
                self.flights = fetchedFlights
            } catch {
                print("Error fetching flights: \(error.local-
izedDescription)")
            }
        }
      }
    }
}

struct FlightListView_Previews: PreviewProvider {
    static var previews: some View {
        FlightsListView()
    }
}
```

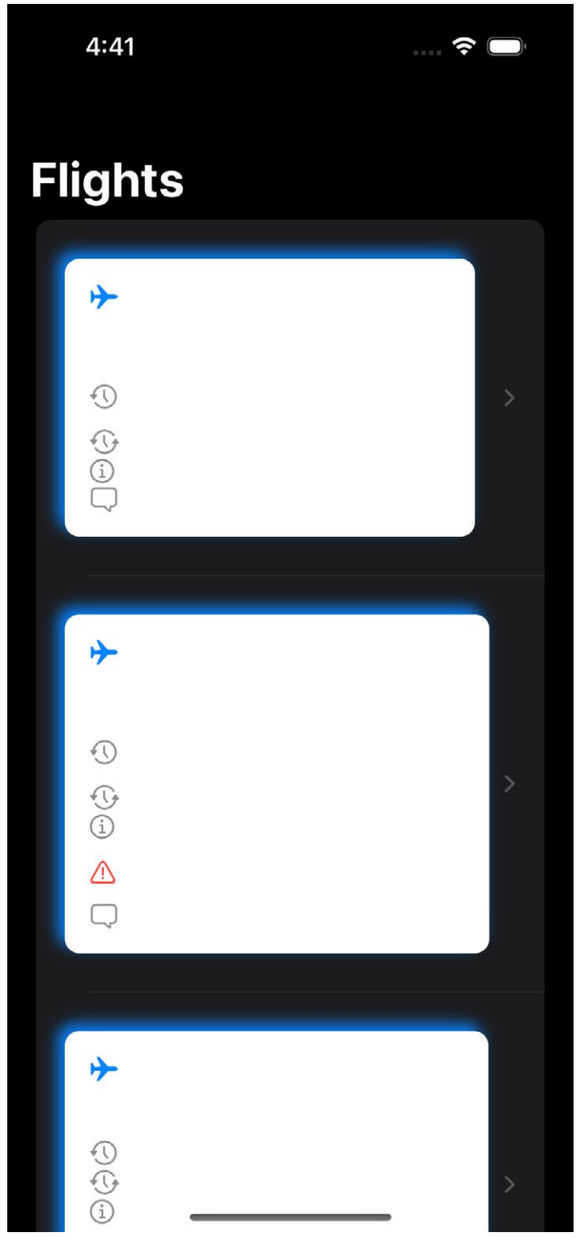


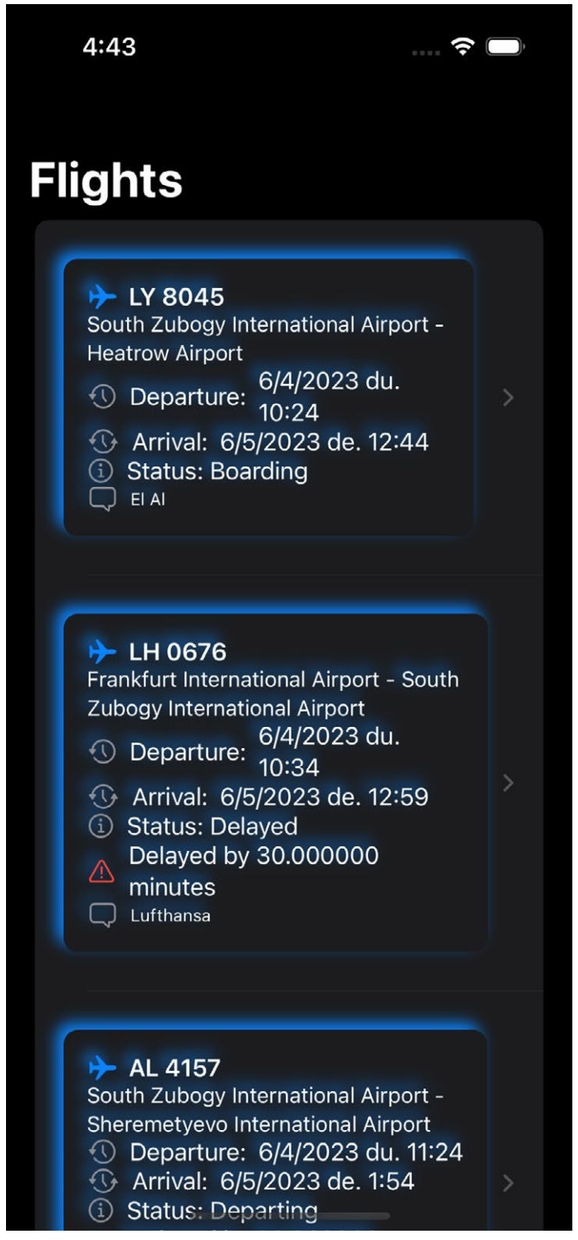

## Before Giving Too Many Points to ChatGPT

Also, before declaring ChatGPT the winner, I wanted to try out a few more things. I wanted to see how it would perform with more complex instructions.

> *Can you add a little blue shadow to the top trailing corner of a list item?*

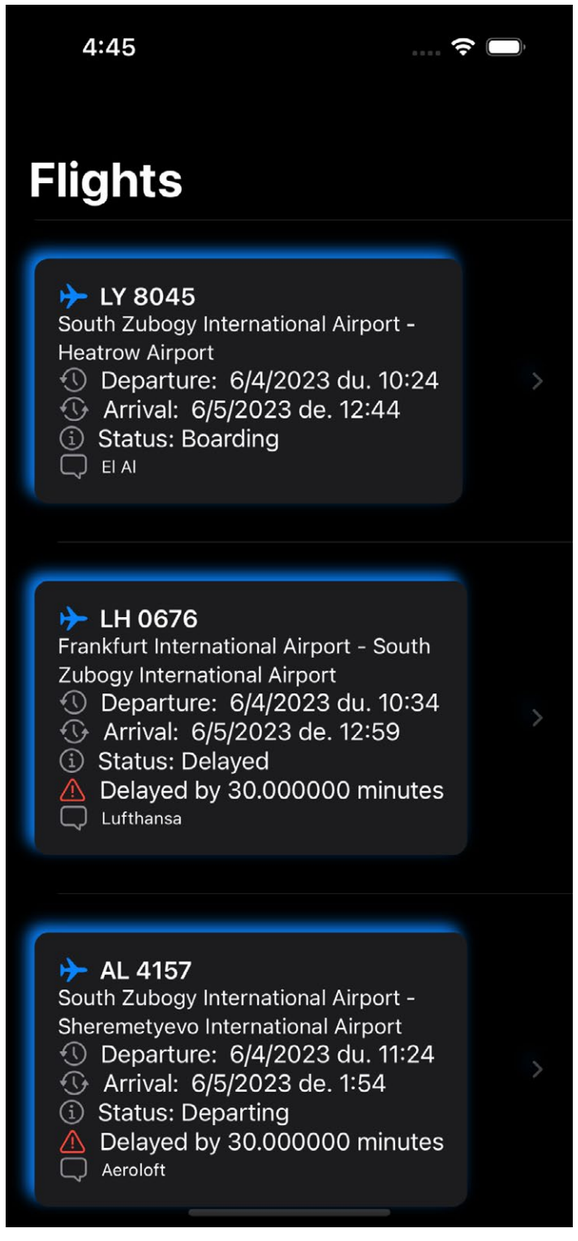



> *Nearly but you are now not supporting dark mode, it's not readable.*




Copilot X seems to be performing slightly worse than ChatGPT, but it's still pretty impressive.

> *Change the list style to plain and only add shadow to the whole cell, not to every text.*




> *Can you please add pull-to-refresh functionality?*

As **of** my **last** knowledge update **in** September 2021, SwiftUI did **not** have **a** built-**in** pull-to-refresh feature. However, you can use **a** `UIRefreshControl` **with a** `UIScrollView` **by** wrapping **it in a** `UIViewRepresentable`.

Let's create a custom SwiftUI view named `RefreshableScrollView` which will integrate the UIKit's `UIRefreshControl` **into a** SwiftUI view:
...

SwiftUI does have a **refreshable** modifier on **List** from iOS 15, but still kudos to ChatGPT by solving the problem in a different way.

> *I would like to use MVVM, can you please create a ViewModel for my FlightsListView?*

```
import Foundation

class FlightsViewModel: ObservableObject {
    @Published var flights: [Flight] = []
    @Published var errorMessage: String? = nil

    func fetchFlights() async {
        do {
            flights = try await FlightAPI().fetchFlights()
        } catch {
            errorMessage = "Failed to fetch flights: \(error.
localizedDescription)"
        }
    }
}
struct FlightsListView: View {
    @StateObject private var viewModel = FlightsView-
Model()

    var body: some View {
        NavigationView {
            VStack {
                if let errorMessage = viewModel.errorMes-
sage {
                    Text(errorMessage)
                        .foregroundColor(.red)
                        .padding()
                }

                RefreshableScrollView(onRefresh: { await
viewModel.fetchFlights() }) {
                    ForEach(viewModel.flights, id: \.self) {
flight in
                        NavigationLink(destination: FlightDe-
tailView(flight: flight)) {
                            FlightCell(flight: flight)
                        }
                    }
                }
                .padding(.horizontal)
            }
            .navigationTitle("Flights")
        }
        .task {
            await viewModel.fetchFlights()
        }
    }
}
```



> *Where is FlightCell?*

```
import SwiftUI

struct FlightCell: View {
    var flight: Flight

    var body: some View {
        VStack(alignment: .leading) {
            HStack {
                Image(systemName: "airplane")
                    .foregroundColor(.blue)
                Text(flight.flightNumber)
                    .font(.headline)
            }
            ...
        }
        .padding()
        .background(Color.secondarySystemBackground)
        .clipShape(RoundedRectangle(cornerRadius: 10))
        .shadow(color: .blue, radius: 5, x: -5, y: -5)
    }
}
```

## Conclusion

Both ChatGPT 4 and Copilot X are formidable allies for iOS developers in their quest to augment the development process.

ChatGPT 4 is the victor in generating higher-quality results across an array of tasks, albeit at a higher cost and with some usage constraints.

Copilot X, though trailing slightly in performance, compensates with a seamless IDE integration, making it more convenient for the day-to-day workflow.

Since both have their own set of advantages, it's difficult to crown an indisputable champion. The decision hinges on individual priorities - whether one values quality and sophistication or integration and cost-effectiveness.

### Context is Limited

When using ChatGPT or Copilot X, we need to recognise the limitations in their memory and input capacities. ChatGPT can handle **4096** tokens, while Copilot X is confined to **2048**. (I've asked them to provide this data.) This necessitates conciseness and precision when dealing with extensive documentation or large-scale enterprise projects.

However, the rapidly advancing AI landscape instills confidence that these token constraints will likely experience significant expansions in the near future.

### Prompt Engineering and Expertise Matters

ChatGPT and Copilot X's output quality is substantially enhanced through meticulous, prompt engineering and contextualisation.

While they can generate code incorporating modern Swift Concurrency and Swift errors, these aren't their defaults due to the abundance of completion handler solutions in their training data.

This highlights the importance of domain expertise to steer the AI tools towards generating state-of-the-art code. These tools can be instrumental in accelerating the development process when wielded by skilled hands.

### Generating Network Layer

We can clearly see the abilities of both ChatGPT and Copilot X in sculpting the network layer's models, with ChatGPT taking an early lead.

The models generated by Copilot X caught up once given more specific directives. The learning point here is not to presume Copilot X can rely solely on open files; it requires some guidance.

Though traditional OpenAPI generators remain unchallenged in speed and consistency, ChatGPT's ability to offer custom solutions and troubleshoot issues offers a versatile dynamism that is valuable. Both AI tools prove adept at refining .mustache files for OpenAPI generators.

ChatGPT 4 also demonstrated finesse in generating networking calls but needed some fine-tuning. Copilot X, though requiring more context and iterations, capitalised on its IDE integration for generating networking calls.

In summary, while ChatGPT 4 had a slight edge in code accuracy, Copilot X was more convenient and responsive. Both significantly accelerated the development process, albeit with the need for some manual refinement.

### UI is Still a Mostly Manual Process

ChatGPT 4 edges ahead of Copilot X in generating UI code, but is contingent on the provision of unambiguous directives.

Both tools exhibited occasional syntactical slips and a limited acquaintance with cutting-edge SwiftUI features.

Copilot X was noted for its imaginative tendencies (hallucination), conjuring non-existent UI elements. Though helpful in expediting the code-writing process, neither tool could replicate the finesse of UI designed by human experts.

AI systems have yet to master the nuances of interpreting and executing complex designs. Thus, ChatGPT and Copilot X are best used as supplementary resources that complement human expertise rather than supplant it.

### What Will Apple Do?

With Microsoft at the forefront with OpenAI-backed tools, Google close on its heels with Bard, and Meta also delving into the domain, my eyes are now on Apple.

So far, Apple has remained tight-lipped, but the market dynamics strongly suggest that Apple might not be far behind in unveiling its own offering.

While the recent WWDC was mostly bereft of announcements on this front, the tech community remains on tenterhooks for the next edition. Apple, known for its innovation, might just bring something groundbreaking to the table.

## Bringing it All Together

In the realm of iOS development, AI tools like ChatGPT and Copilot X are carving out powerful accelerants that can streamline various aspects of the development lifecycle.

While ChatGPT 4 emerges as the forerunner in generating sophisticated code, Copilot X's integration with IDEs makes it a practical tool for daily use (if you don't mind switching back and forth between Visual Studio Code and Xcode).

However, it is essential to recognise and navigate the limitations of these AI systems, especially in terms of token capacity and the necessity for precise prompts.

As of now, when it comes to UI design, human creativity and expertise hold sway, as AI-generated code still lacks the finesse required for intricate designs. But, for generating network layers and other backend functionalities, AI tools are proving to be valuable.

It is essential for developers to remain abreast of the evolving AI landscape. The token limitations that are pertinent today may not be significant constraints tomorrow, and AI-generated code may soon rival human-generated code in sophistication.

Finally, in this highly competitive tech ecosystem, Apple's entry into the AI tool space is eagerly anticipated. While ChatGPT and Copilot X are excellent tools in their own right, Apple's innovation could redefine the standards and expectations.

In conclusion, integrating AI tools into (iOS) development is not just a fad, but an evolution that is here to stay.

However, the use of AI does not obviate the need for human expertise. Rather, it's the synergy of human intelligence and AI capabilities that will drive the future of (iOS) development. As developers, continually adapting and learning to harness these tools effectively is the key to staying relevant and successful in this ever-evolving landscape.

zenitech

STEVENS&BOLTON

## Coming up:

**AutoGPT**
**Copilot**
**SuperAGI**
**Tabnine**

# Look out for Edition 2 of the New Beings AI eBook