



Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τηλεπικοινωνιών

Τίτλος διπλωματικής

Διπλωματική Εργασία
του
Θεοφάνη Θαρρόπουλου

Επιβλέπων: Ανδρέας Συμεωνίδης
Καθηγητής Α.Π.Θ.

27 Ιουλίου 2024

Περίληψη

Αντικείμενο της παρούσας διπλωματικής εργασίας αποτελεί η έρευνα για την αξιολόγηση της ποιότητας του κώδικα που παράγεται από Μεγάλα Γλωσσικά Μοντέλα (LLMs), και πιο συγκεκριμένα από το GitHub Copilot[2]. Η μελέτη εστιάζει στην αξιολόγηση της ποιότητας του κώδικα που παράγεται από το όπιλοτ και στην βελτιστοποίηση των προτροπών (prompts) για την επίτευξη των επιθυμητών αποτελεσμάτων μέσω τεχνικών μηχανικής προτροπής (prompt engineering) και της μηχανικής μάθησης. Τα αποτελέσματα αποδεικνύουν τις δυνατότητες και τους περιορισμούς του όπιλοτ στην παραγωγή ποιοτικού κώδικα και προσφέρουν νέες προσεγγίσεις για την βελτίωση της αλληλεπίδρασης μεταξύ του χρήστη και του εργαλείου μέσω στοχευμένων τεχνικών προτροπής.

Abstract

Empty

Ευχαριστίες

Άδειο

Τίτλος διπλωματικής

Θεοφάνης Θαρρόπουλος
theofact@ece.auth.gr

27 Ιουλίου 2024

Περιεχόμενα

1 Εισαγωγή	2
1.1 Δημιουργία Κώδικα (Code Generation) - Ολοκλήρωση Κώδικα (Code Completion)	3
2 Μεθοδολογία	8
2.1 Επιλογή Εφαρμογής	8
2.1.1 Τεχνολογική Στοίβα Technology Stack	11
2.2 Συλλογή Δεδομένων	11
2.2.1 Διαδικασία Συλλογής Δεδομένων	12
2.2.2 Ανάλυση Δεδομένων	17
2.2.3 Αξιολόγηση Απαντήσεων	32
2.3 Συμπεράσματα	32
Α' Ακρωνύμια και συντομογραφίες	33

Κεφάλαιο 1

Εισαγωγή

Η ανάπτυξη των Μεγάλων Γλωσσικών Μοντέλων (LLM) έχει επιφέρει ριζικές αλλαγές στον τομέα της Τεχνητής Νοημοσύνης και της Επεξεργασίας Φυσικής Γλώσσας (NLP) [14, 82, 83, 36]. Η εισαγωγή αυτών των μοντέλων στην καθημερινή ζωή μέσω του Chat-GPT το 2022 [52] έχει πυροδοτήσει μια επανάσταση στην τεχνολογική αγορά, σηματοδοτώντας την απαρχή του αγώνα για την κυριαρχία στην αγορά της Τεχνητής Νοημοσύνης [68, 45, 62, 41, 6]. Τα μοντέλα αυτά έχουν αποδειχθεί εξαιρετικά αποτελεσματικά στην αντιμετώπιση προβλημάτων επεξεργασίας φυσικής γλώσσας, όπως η αναγνώριση φυσικής γλώσσας, προάγοντας την ανάπτυξη της Γενικής Τεχνητής Νοημοσύνης (AGI) [8, 29].

Μέσα σε αυτή την επανάσταση, η χρήση μοντέλων επεξεργασίας φυσικής γλώσσας στον τομέα του προγραμματισμού και της ανάπτυξης λογισμικού, ονόματι βοηθοί κώδικα (Code Assistants), έχει αναδειχθεί ως ένας από τους πιο υποσχόμενους τομείς της τεχνολογίας. Ένα από τα πιο διαδεδομένα εργαλεία με αυτόν το σκοπό είναι το GitHub Copilot [28, 2]. Αναπτυγμένο σε συνεργασία με την OpenAI, το GitHub Copilot ξεκίνησε χρησιμοποιώντας το μοντέλο ονόματι Codex της OpenAI [18], σχεδιασμένο εξ αρχής αποκλειστικά για τη παραγωγή κώδικα, για να προτείνει κώδικα στον προγραμματιστή κατά την γραφή κώδικα.

Η παροδική απόσυρση του μοντέλου Codex τον Μάρτιο του 2023 και η οριστική του απόσυρση το 2023 [38], οδήγησε στην ανάπτυξη ενός νέου μοντέλου, σε συνεργασία μεταξύ της OpenAI, της Microsoft Azure AI, και της GitHub AI. Το νέο μοντέλο αρχικά βασίστηκε στο GPT-3.5 Turbo [81], με την επόμενή του έκδοση να βασίζεται στο GPT-4 [81], με το κωδικό όνομα GitHub Copilot X [22], δίνοντας την δυνατότητα για μια νέα λειτουργία, του GitHub Copilot Chat, ενός chatbot μοντέλου, παρόμοιο με αυτό του Chat-GPT. Μέσω αυτής της λειτουργίας, ο προγραμματιστής μπορεί μέσα από το περιβάλλον ανάπτυξής του (IDE), να κάνει ερωτήσεις στο μοντέλο, χρησιμοποιώντας φυσική γλώσσα, ενώ το μοντέλο μπορεί να χρησιμοποιήσει τον ήδη υπάρχοντα κώδικα, την τεκμηρίωση, και τις οδηγίες του προγραμματιστή για να απαντήσει στην ερώτηση του προγραμματιστή.

Σχήμα 1.1: Παράδειγμα χρήσης του GitHub Copilot Chat εντός του IDE Visual Studio Code [72, 7]

Το GitHub Copilot Chat αρχικά ήταν διαθέσιμο μόνο μέσω λίστας αναμονής, με την πρόσθαση να δίνεται σε περιορισμένο αριθμό προγραμματιστών, κατόπιν αίτησης, με την ενσωμάτωση του GPT-4 να γίνεται επίσημα τον Νοέμβριο του 2023 [66]. Η δημόσια κυκλοφορία του GitHub Copilot Chat έγινε τον Δεκέμβριο του 2023 [80].

Πέρα από το GitHub Copilot, υπάρχουν πολλοί άλλοι code assistants, με αυτούς που λήφθηκαν υπ' όψιν να είναι οι:

- Tabnine [74, 76]
- Codeium [17]
- Amazon Codewhisperer [13]

Η απόφαση για την χρήση του GitHub Copilot ως το εργαλείο, του οποίου η λειτουργία θα εξεταστεί στην παρούσα διπλωματική, έγινε με βάση την ευρεία χρήση του, την ενσωμάτωση του μοντέλου GPT-4, και την δυνατότητα χρήσης του GitHub Copilot Chat, καθώς και την δωρεάν παροχή του σε ενεργούς φοιτητές του Αριστοτελείου Πανεπιστήμιου Θεσσαλονίκης μέσω του προγράμματος GitHub Student Developer Pack [3].

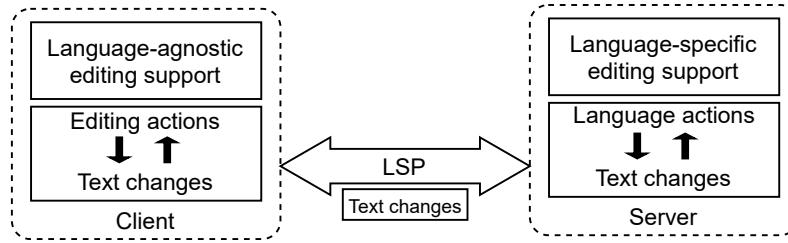
1.1 Δημιουργία Κώδικα (Code Generation) - Ολοκλήρωση Κώδικα (Code Completion)

Δύο όροι που συχνά συναντώνται στην βιβλιογραφία είναι η **Δημιουργία Κώδικα** (Code Generation) και η **Ολοκλήρωση Κώδικα** (Code Completion). Και οι δύο όροι έχουν άμεση σχέση με την παραγωγικότητα του προγραμματιστή και αποτελούν σημαντικά

εργαλεία για την ανάπτυξη λογισμικού [43, 40, 10], και ενώ και τα δύο μπορούν να χρησιμοποιήσουν μοντέλα Τεχνητής Νοημοσύνης [70, 60] με την διαφορά να βρίσκεται στον τρόπο λειτουργίας τους.

Ολοκλήρωση Κώδικα

Σύμφωνα με Omar et al. [51], η Ολοκλήρωση Κώδικα αφορά εργαλεία που τα περισσότερα περιβάλλοντα ανάπτυξης παρέχουν μέσω της μορφής πλωτού μενού που περιέχει συμφραζόμενες-σχετικές μεταβλητές, πεδία, μεθόδους, τύπους και άλλα αποσπάσματα κώδικα. Επιλέγοντας από το μενού, οι προγραμματιστές μπορούν να αποφύγουν πολλά συνηθισμένα ορθογραφικά και λογικά λάθη και να εξαλείψουν τις περιπτές πληκτρολογήσεις. Το βασικό εργαλείο που χρησιμοποιείται για την επίτευξη του σκοπού αυτού είναι το Language Server Protocol (LSP) της Microsoft, δίνοντας την δυνατότητα σε κάθε περιβάλλον ανάπτυξης να επικοινωνήσει με την εξωτερική διεργασία του Language Server και να λάβει πληροφορίες για τον κώδικα που γράφεται, όπως τις προτάσεις ολοκλήρωσης κώδικα, τα λάθη, και τις προτάσεις διόρθωσης, σε μια διάταξη χρήστη - διακομιστή [59, 15].



Σχήμα 1.2: Η αρχιτεκτονική του Language Server Protocol, Δανεισμένο από [61]

Η Ολοκλήρωση Κώδικα δεν παράγει κώδικα από το μηδέν, αλλά προτείνει συμπληρωματικές λύσεις στον υπάρχοντα κώδικα, βοηθώντας τον προγραμματιστή να ολοκληρώσει τον κώδικα του ταχύτερα.

The screenshot shows a code editor window in Visual Studio Code. The code being edited is:

```
return; },  
    OkImpl  
)  
Old_Standard_... node_modules/next/dist/compiled/@ne...  
Oldenburg     node_modules/next/dist/compiled/@next/...  
getById: pu Ole          node_modules/next/dist/compiled/@next/...  
.input(z. Oleo_Script   node_modules/next/dist/compiled/@next/...  
.query(as Oleo_Script_Swash_C... node_modules/next/dist/compiled/@next/...  
const r Ole nest       node_modules/next/dist/compiled/@next/...  
.find Oleh_Baby      node_modules/next/dist/compiled/@next/...  
whe [?] OpacityIcon  node_modules/@radix-ui/react-icons/dis...  
i [?] OpenInNewWindowIc... node_modules/@radix-ui/react-ic...  
, Ole Open_Sans      node_modules/next/dist/compiled/@next/...  
) [?] Option  
then((res) -> d
```

A code completion dropdown menu is open over the variable 'r'. The suggestions include:

- OkImpl
- Old_Standard_... node_modules/next/dist/compiled/@ne...
- Oldenburg node_modules/next/dist/compiled/@next/...
- Ole node_modules/next/dist/compiled/@next/...
- Oleo_Script node_modules/next/dist/compiled/@next/...
- Oleo_Script_Swash_C... node_modules/next/dist/compiled/@next/...
- Olest node_modules/next/dist/compiled/@next/...
- Oleh_Baby node_modules/next/dist/compiled/@next/...
- OpacityIcon node_modules/@radix-ui/react-icons/dis...
- OpenInNewWindowIc... node_modules/@radix-ui/react-ic...
- Open_Sans node_modules/next/dist/compiled/@next/...
- Option

Σχήμα 1.3: Παράδειγμα Ολοκλήρωσης Κώδικα εντός του IDE Visual Studio Code

Δημιουργία Κώδικα

Η Δημιουργία Κώδικα αφορά την άμεση παραγωγή κώδικα από ένα μοντέλο στοχασχτικά, με βάση το συγκείμενο κώδικα του προγραμματιστή. Το μοντέλο που χρησιμοποιείται για την παραγωγή κώδικα είναι εκπαιδευμένο σε μεγάλα σύνολα δεδομένων και μπορεί να παράγει κώδικα από το μηδέν. Το μοντέλο αυτό μπορεί να παράγει κώδικα σε πολλές γλώσσες προγραμματισμού, όπως Python, JavaScript, C++ κ.α. Η παραγωγή κώδικα μπορεί να γίνει μέσω μιας διεπιφάνεις (API) που παρέχεται από το μοντέλο, ή μέσω μιας επέκτασης ενός ειδικού περιβάλλοντος ανάπτυξης, όπως στην περίπτωση του GitHub Copilot.

```

src > server > api > routers > TS game.ts > [x] gameRouter > ⚡ get
16 export interface GameDetails extends Omit<Game, "franchiseId" | "publisherId"> {
11   platforms: Array<{
8     >;
7     franchise: Omit<Franchise, "image">;
6     publisher: Omit<Publisher, "image">;
5     reviews?: Array<ReviewContext>;
4   }
3
2   export const gameRouter = createTRPCRouter({
1     getAll: publicProcedure.query(async () => {
53       const result: Result<Array<Game>, TRPCError> = await ctx.prisma.game
         .findMany()
         .then((res) => Ok(res), handlePrismaError);

       return result;
     })
1
2
3     getById: publicProcedure
4       .input(z.object({ id: z.string().cuid2() }))
5       .query(async ({ ctx, input }) => {
6         const result: Result<Game, TRPCError> = await ctx.prisma.game
7           .findOneUnique({
8             where: {
9               id: input.id,

```

Σχήμα 1.4: Παράδειγμα Δημιουργίας Κώδικα σε πραγματικό χρόνο (real time code generation) εντός του IDE Visual Studio Code από το GitHub Copilot

Η παραγωγή κώδικα γίνεται με διάφορους βαθμούς λεπτομέρειας. Αρχικά, το μοντέλο υπολογίζει το επόμενο σύμβολο του κώδικα χρησιμοποιώντας **next token prediction** [32, 39, 77, 24, 20, 19]. Στο στάδιο της ολοκλήρωσης ολόκληρης σειράς κώδικα, το μοντέλο χρησιμοποιεί ένα κομμάτι συγκείμενου κώδικα. [32, 30, 69, 42]. Στο μέγιστο δυνατό βαθμό, το μοντέλο μπορεί να παράγει ολόκληρες συναρτήσεις ή κλάσεις. [26, 30, 2].

Η παραγωγή κώδικα επίσης μπορεί να λάβει υπ' όψιν αποκλειστικά το συγκείμενο κώδικα πριν την θέση του κέρσορα στο αρχείο ή και τον κώδικα που ακολουθεί την θέση του κέρσορα. [33] Το μοντέλο του GitHub Copilot χρησιμοποιεί την δεύτερη προσέγγιση, παράγοντας κώδικα που συμπληρώνει τον υπάρχοντα κώδικα του προγραμματιστή. [2, 26, 78]. Κατά την έρευνα λήφθηκε η απόφαση να χρησιμοποιηθεί, ως επί το πλείστον, η λειτουργία GitHub Copilot Chat για την έρευνα και τα πειράματα, κυρίως γιατί ο σύγκειμενος κώδικας που χρησιμοποιεί το μοντέλο κατά την λειτουργία αύτη, μπορεί να επιλεχθεί είτε από το μοντέλο, είτε συγκεκριμένα από τον προγραμματιστή. Ένας ακόμα λόγος που επιλέχθηκε η χηση του GitHub Copilot Chat, είναι ότι η καταγραφή του κώδικα που παράχθηκε κατά την έρευνα από το μοντέλο ήταν πολύ δυσκολότερη με την χρήση του GitHub Copilot, καθώς η δημιουργία κώδικα σε πραγματικό χρόνο (**real time code generation**) ήταν πολύ δύσκολη αυτής του GitHub Copilot Chat.

```

1  TS game.ts 1, M ×
src > server > api > routers > TS game.ts > [e]gameRouter
23 export interface GameDetails extends Omit<Game, "franchiseId" | "publisherId"> {
24   _count: { reviews: number };
25   developers: Array<Omit<Developer, "image">>;
26   features: Array<Omit<Feature, "image">>;
27   genres: Array<Genre>;
28   platforms: Array<{
29     storeLink: string;
30     platform: Omit<Platform, "image">;
31   }>;
32   franchise: Omit<Franchise, "image">;
33   publisher: Omit<Publisher, "image">;
34   reviews: Array<ReviewContext>;
35 }
36
37 export const gameRouter = createTRPCRouter({
38   getAll: publicProcedure.query(
39     async ({ ctx }: { ctx: any }): Promise<Result<Array<Game>, TRPCError>> => {
40       const result: Result<Array<Game>, TRPCError> = await ctx.prisma.game
41         .findMany()
42         .then((res) => Ok(res), handlePrismaError);
43       return result;
44     }
45   ),
46   getById: publicProcedure
47     .input(z.object({ id: z.string().cuid2() }))
48     .query(async ({ ctx, input }: { ctx: any; input: { id: string } }): Promise<Result<Game, TRPCError>> => {
49       const result: Result<Game, TRPCError> = await ctx.prisma.game
50         .findUnique({
51           where: {
52             id: input.id,
53           },
54         })
55     }
56   ),
57   You: 6 months ago + feat(trpc): game router
58 }
59
60

```

What does this code reference do? ...

GitHub Copilot

Used 1 reference

TS game.ts:52-60 src/server/api/routers

This TypeScript code is defining a method `getAll` for a `publicProcedure` object. This method is a query that asynchronously retrieves all instances of `Game` from a database using Prisma, a database toolkit.

Here's a step-by-step breakdown:

- `publicProcedure.query` is a method that defines a query operation. It takes an `async` function as an argument.
- The `async` function takes an object with a `ctx` property as an argument. `ctx` is typically short for "context" and is often used to pass around important shared resources.
- Inside the function, it uses `ctx.prisma.game.findMany()` to query the database for all instances of `Game`. Prisma's `findMany` method returns a `Promise` that resolves to an array of `Game` instances.
- The `.then` method is used to handle the `Promise` returned by `findMany`. If the `Promise` resolves successfully, the `Ok` function is called with the result (`res`). If the `Promise` is rejected, the `handlePrismaError` function is called with the error.

What is the purpose of using Prisma in this code?

Ask Copilot or type / for commands

Spaces: 2 UTF-8 LF {} TypeScript Go Live Spell Prettier

Σχήμα 1.5: Παράδειγμα Δημιουργίας Κώδικα σε πραγματικό χρόνο (real time code generation) εντός του IDE Visual Studio Code από το GitHub Copilot

Κεφάλαιο 2

Μεθοδολογία

Στο κεφαλαίο αυτό παρουσιάζεται η μεθοδολογία που ακολουθήθηκε κατά την διάρκεια της σύλλογης των δεδομένων των πειραμάτων με το GitHub Copilot και με το GitHub Copilot Chat. Αρχικά, η επιλογή της εφαρμογής πάνω στην οποία το GitHub Copilot αξιολογήθηκε, περιγράφεται παρακάτω.

2.1 Επιλογή Εφαρμογής

Για την επιλογή της κατάλληλης εφαρμογής, λήφθηκαν οι παρακάτω παράμετροι υπ' όψιν:

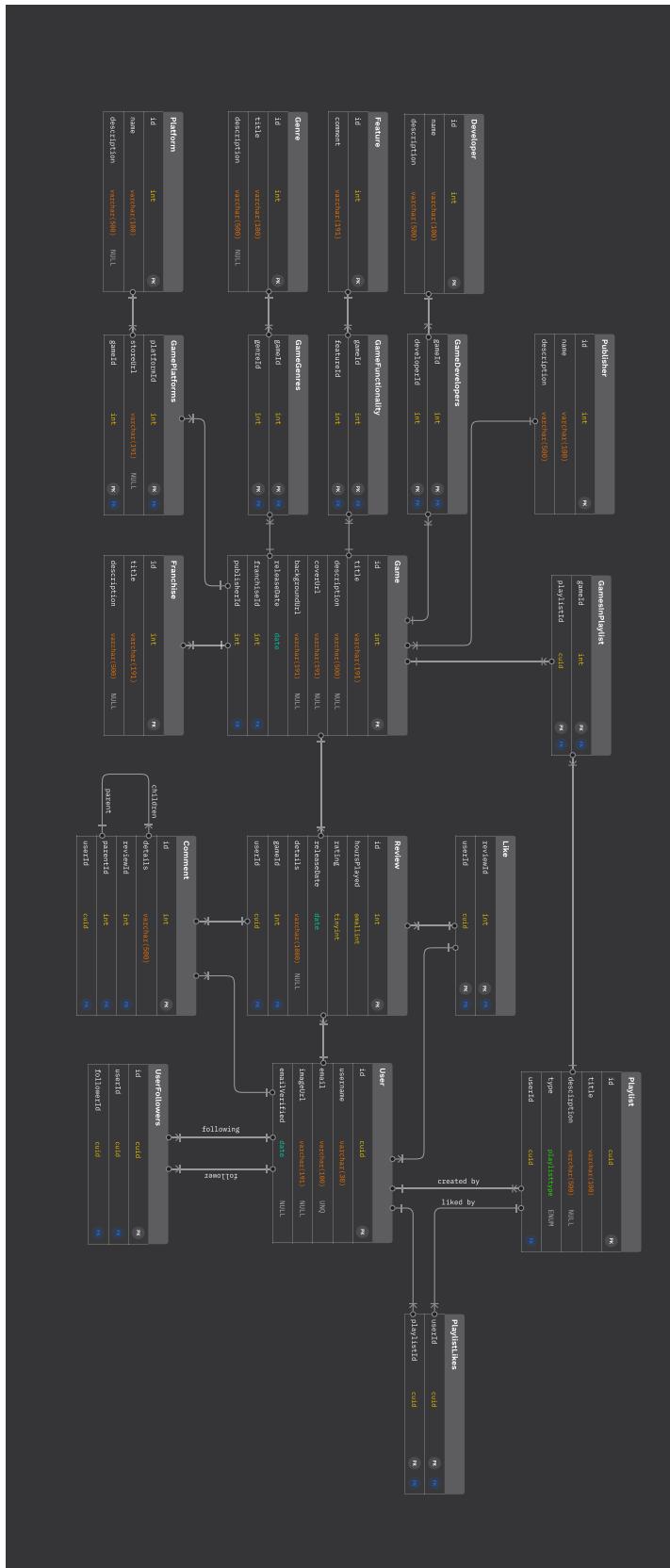
- **Πολυπλοκότητα της εφαρμογής:** Η εφαρμογή πρέπει να είναι αρκετά πολύπλοκη ώστε να απαιτεί την χρήση του GitHub Copilot για την ανάπτυξη του κώδικα.
- **Τύπος της εφαρμογής:** Η εφαρμογή πρέπει να είναι μια ιδέα αντίστοιχη με τις υπόλοιπες υλοποιήσεις του κλάδου της Μηχανικής Λογισμικού και συγκεκριμένα της Ανάπτυξης Ιστότοπων **Web Development**.
- **Γλώσσα της εφαρμογής και Τεχνολογική Στοίβα Technology Stack:** Η επιλογή της Τεχνολογικής Στοίβας πρέπει να γίνει με βάση μοντέρνες τεχνολογίες και με ευρεία χρήση, προκειμένου το μοντέλο να έχει την περισσότερη εμπειρία και τις καλύτερες αποδόσεις, αλλά και να γίνεται μια αξιολόγηση με πραγματικά και εξελισσόμενα εργαλεία.

Με βάση τις παραπάνω παραμέτρους, επιλέχθηκε η ανάπτυξη μιας εφαρμογής η οποία θα αποτελείται από έναν ιστότοπο που θα παρέχει ένα μέσο δικτύωσης που αφορά τα βιντεοπαιχνίδια. Η εφαρμογή αυτή είχε ήδη αναπτυχθεί στα πλαίσια του μαθήματος ‘Μηχανική Λογισμικού Ι’, επομένως οι βασικές ανάγκες και λειτουργίες της εφαρμογής είχαν καταγραφθεί. Η επιλογή αυτή έγινε προκειμένου η αξιολόγηση του κώδικα και των προτάσεων του GitHub Copilot να μην έχει αλλαγές με βάση τις αλλαγές που προέκυψαν κατά τον σχεδιασμό της εφαρμογής. Η εφαρμογή αποτελείται από τρεις (3) βασικές λειτουργίες:

- Κριτική παιχνιδιών και κοινοποίηση των κριτικών των χρηστών με τους υπόλοιπους χρήστες της εφαρμογής.

- Διαχείριση και κοινοποίηση λιστών (playlists) από βιντεοπαιχνίδια με τους υπόλοιπους χρήστες της εφαρμογής.
- Δυνατότητα στον χρήστη να σχολιάσει τις κριτικές των άλλων χρηστών, καθώς και να ‘ακολουθήσει’ άλλους χρήστες, λαμβάνοντας ειδοποιήσεις για τις ενέργειές τους.

Παρατίθεται το Σχεσιακό Διάγραμμα Οντοτήτων της εφαρμογής:



Σχήμα 2.1: Σχεσιακό Διάγραμμα Οντοτήτων της εφαρμογής

2.1.1 Τεχνολογική Στοίβα Technology Stack

Η επιλογή της Τεχνολογικής Στοίβας έγινε με βάση την ευρεία χρήση των τεχνολογιών αυτών, την ευκολία στην ανάπτυξη και την ευκολία στην ενσωμάτωση του GitHub Copilot. Η εφαρμογή αναπτύχθηκε με την χρήση των παρακάτω τεχνολογιών:

- **Γλώσσα Προγραμματισμού:** Η εφαρμογή αναπτύχθηκε με την χρήση της γλώσσας προγραμματισμού TypeScript [46], ενός συντακτικού υπερσυνόλου της γλώσσας JavaScript [49], μια από τις πιο διαδομένες γλώσσες προγραμματισμού στον κόσμο. [16, 67]
- **Εμπρόσθια Ανάπτυξη (Frontend Development):** Η εμπρόσθια ανάπτυξη της εφαρμογής έγινε με την χρήση της React [44], μιας βιβλιοθήκης της JavaScript για την ανάπτυξη διεπαφών χρήστη. Συγκεκριμένα, χρησιμοποιήθηκε το πλαίσιο εργασίας Next.js [75], το οποίο παρέχει δυνατότητες όπως την προ-φόρτωση των σελίδων, την δυνατότητα δημιουργίας στατικών ιστοσελίδων, και την δυνατότητα δημιουργίας δυναμικών ιστοσελίδων, αποτελώντας ένα από τα πιο ευρέως χρησιμοποιημένα πλαίσια εργασίας.
- **Πίσω Ανάπτυξη (Backend Development):** Η πίσω ανάπτυξη της εφαρμογής έγινε με την χρήση της βιβλιοθήκης tRPC [73], μιας βιβλιοθήκης που παρέχει την δυνατότητα δημιουργίας API με την χρήση της γλώσσας TypeScript. Η βάση δεδομένων της εφαρμογής αποθηκεύτηκε σε μια βάση δεδομένων MySQL [54], μιας από τις πιο διαδεδομένες σχεσιακές βάσεις δεδομένων. Το εργαλείο της Αντικειμενο-σχεσιακής Απεικόνισης (ORM) που χρησιμοποιήθηκε ήταν το Prisma [57], το οποίο παρέχει την δυνατότητα δημιουργίας απλών και ασφαλών ερωτημάτων (queries) στην βάση δεδομένων. Το σύστημα διαχείρισης της ταυτότητας των χρηστών (authentication) και των δικαιωμάτων τους (authorization) υλοποιήθηκε με την χρήση της βιβλιοθήκης NextAuth.js [50].
- **Έλεγχος Λογισμικού (Software Testing):** Για τον έλεγχο της λειτουργίας του παραγόμενου κώδικα, χρησιμοποιήθηκε το πλαίσιο εργασίας Jest [53], το οποίο παρέχει την δυνατότητα δημιουργίας και εκτέλεσης δοκιμαστικών συνόλων κώδικα, με σκοπό την εξασφάλιση της άρτιας λειτουργίας του κώδικα. [34, 31, 5, 47, 63]

Η παραπάνω στοίβα ονομάστηκε T3 stack, έχοντας πλέον δημιουργήσει μια μεγάλη κοινότητα στον κλάδο της ανάπτυξης ιστότοπων και της ανάπτυξης λογισμικού. [71]

2.2 Συλλογή Δεδομένων

Για την συλλογή των δεδομένων, αρχικά δημιουργήθηκε ο σκελετός της βάσης κώδικα (codebase), καθώς και μια λειτουργικότητα της εφαρμογής, μαζί με τα αντίστοιχα αρχεία ελέγχου της λειτουργίας του κώδικα. Η λειτουργικότητα που επιλέχθηκε να αναπτυχθεί χωρίς την χρήση του μοντέλου ήταν αυτής της σειράς του βιντεοπαιχνιδιού (Franchise). Η επιλογή έγινε γιατί η λειτουργικότητα αυτή ήταν αρκετά απλή, καθώς η σειρά έχει μόνο σχέσεις 1:M με οντότητα του βιντεοπαιχνιδιού (Game) ::.

Με αυτό τον τρόπο, δημιουργήθηκαν μέθοδοι για την δημιουργία, την ανάγνωση, την ενημέρωση και την διαγραφή των σειρών του βιντεοπαιχνιδιού (CRUD), καθώς και έλεγχοι για την σωστή εκτέλεση αυτών, με βάση την ταυτοποίηση και των δικαιωμάτων των

χρηστών. Σκοπός της προεργασίας αυτής ήταν η αξιολόγηση του κατά πόσο το μοντέλο θα ακολουθούσε τις κατευθυντήριες γραμμές που ορίστηκαν από τον προγραμματιστή.

2.2.1 Διαδικασία Συλλογής Δεδομένων

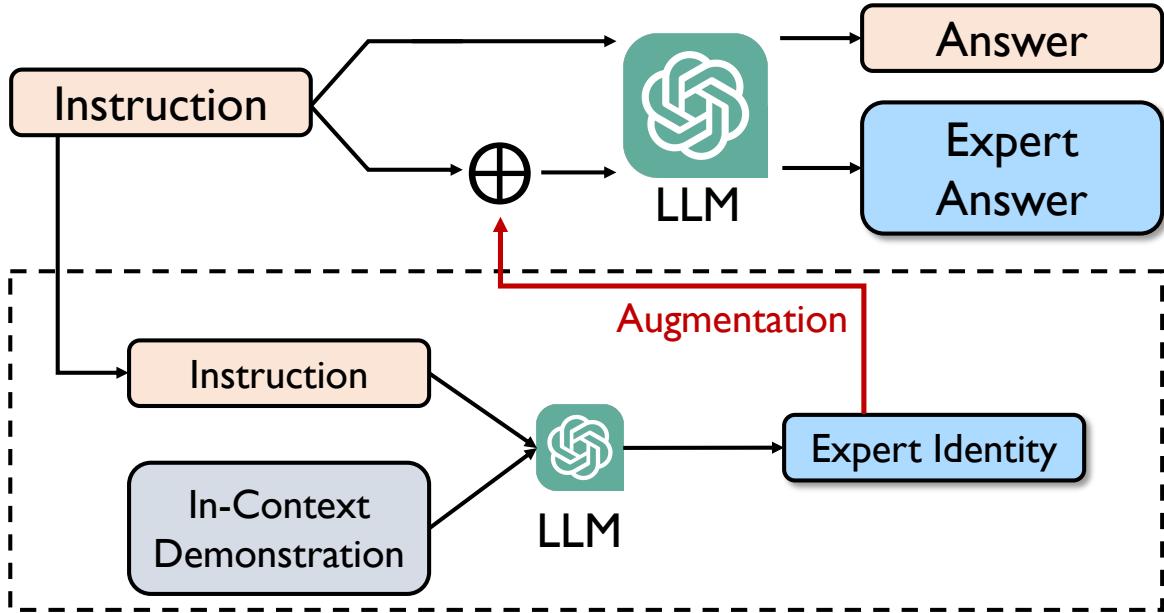
Για την καλύτερη μορφή και κατηγοριοποίηση των δεδομένων, η κάθε προτροπή χωρίστηκε σε τέσσερις (4) κατηγορίες, με βάση το είδος της:

- **Γλώσσας (Language):** Προτροπές που αφορούν ερωτήσεις για την γλώσσα προγραμματισμού (συντακτικό, δομή).
- **Πίσω Ανάπτυξης (Backend):** Προτροπές που αφορούν ερωτήσεις για την ανάπτυξη του API της εφαρμογής.
- **Ελέγχου (Testing):** Προτροπές που αφορούν ερωτήσεις για την ανάπτυξη ελέγχου για τις μεθόδους που έχουν γραφτεί.
- **Άλλες:** Προτροπές που αφορούν ερωτήσεις που δεν σχετίζονται με την συγγραφή κώδικα (για την λειτουργικότητα του GitHub Copilot, για την δομή των απαντήσεών του, τυχόν λάθη από πλευρά του προγραμματιστή κατά την σύνταξη της προτροπής).

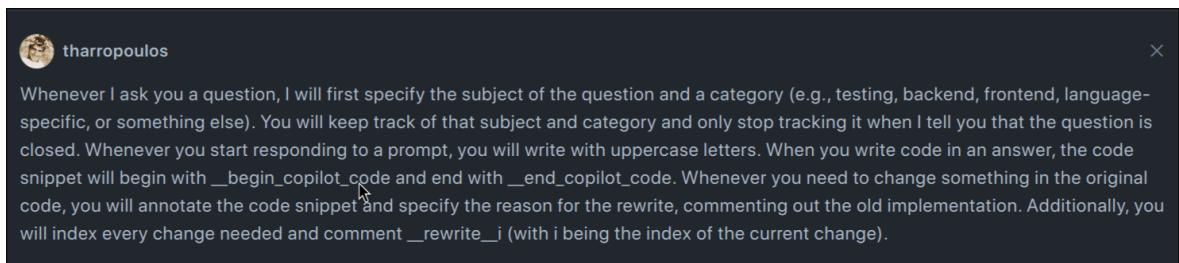
Πέρα από την αρχική κατηγοριοποίηση των προτροπών, η κάθε προτροπή επίσης κατηγοριοποίηθηκε περαιτέρω, με το κάθε θέμα της ερώτησης, όπως για παράδειγμα ανάπτυξη των μεθόδων που αφορούσαν μια οντότητα της εφαρμογής και ανάπτυξη ελέγχων αυτών των μεθόδων, εκ των οπίσιων ελέγχων χωρίστηκαν σε Integration Tests (έλεγχοι συμβατότητας) και σε Unit Tests (έλεγχοι μονάδας) [35, 37]. Οι διαφορές μεταξύ των δύο είναι οι εξής:

- **Unit Testing (Έλεγχος Μονάδας):** Το Unit Testing επικεντρώνεται στον έλεγχο μεμονωμένων κομματιών κώδικα, συνήθως μεθόδων ή συναρτήσεων, σε απομόνωση. Στοχεύει στην επαλήθευση ότι κάθε μονάδα του λογισμικού λειτουργεί σωστά.
- **Integration Testing (Έλεγχος Συμβατότητας):** Το Integration Testing εξετάζει πώς διαφορετικά μέρη του συστήματος λειτουργούν μαζί. Ελέγχει την αλληλεπίδραση μεταξύ διαφορετικών μονάδων ή συστατικών για να διασφαλίσει ότι συνεργάζονται σωστά ως σύνολο. Η ειδοποιός διαφορά είναι ότι το Unit Testing εστιάζει σε μεμονωμένα κομμάτια κώδικα, ενώ το Integration Testing εξετάζει πώς αυτά τα κομμάτια συνεργάζονται. [56]

Για την συλλογή των δεδομένων, δημιουργήθηκε μια περσόνα ενός προγραμματιστή για το μοντέλο [84, 9, 79], με τους κανόνες να ακολουθεί κάθε του απάντηση μια συγκεκριμένη δομή. Ζητήθηκε από το μοντέλο να αρχίζει κάθε μήνυμα με το θέμα της ερώτησης, τον τύπο της ερώτησης και έναν άριθμο που αναπαριστά την σειρά της ερώτησης για το συγκεκριμένο θέμα. Στη συνέχεια, ζητήθηκε από το μοντέλο να βάζει σχόλια στον κώδικα που πάραγε, υποδεικνύοντας την αρχή και το τέλος του παραγόμενου κώδικα, με τη μορφή `BEGIN_COPILOT_CODE / END_COPILOT_CODE`. Τέλος, ζητήθηκε από το μοντέλο να παράγει κώδικα που να ακολουθεί τις κατευθυντήριες γραμμές που ορίστηκαν από τον προγραμματιστή.



Σχήμα 2.2: Η μεθοδολογία δημιουργίας περσόνας για το μοντέλο, Δανεισμένο από [79]



Σχήμα 2.3: Η προτροπή που δόθηκε στο μοντέλο για την δημιουργία της περσόνας

Η επιλογή να ζητηθεί από το μοντέλο να υποδεικνύει τον κώδικα που παράγει έγινε προκειμένου να συλλεχθούν δεδομένα για την ποιότητα του κώδικα που παρήχθη, αλλά και για να καταγραφτούν οι φορές που ο προγραμματιστής επενέβη, με σκοπό να διορθώσει λάθη που το μοντέλο αδυνατούσε. Η τακτική αυτή προϋπόθετε ότι ο προγραμματιστής και χειριστής του μοντέλου θα είχε την δυνατότητα να αντιληφθεί τα λάθη που παρήχθησαν από το μοντέλο, και να τα διορθώσει, καθώς και να γνωρίζει πώς να προτρέψει το μοντέλο να απαντήσει την επιθυμητή απάντηση. Χωρίς αυτήν την απόφαση, το μοντέλο θα παρήγαγε κώδικα ο οποίος δεν θα ήταν λειτουργικός, καθώς υπήρχαν συντακτικά λάθη που δεν μπορούσε να παρατηρήσει.



```
1 // BEGIN_COPILOT_CODE
2 describe("and the user is following the creator", () => {
3   it("should return the playlist", async () => {
4     // Arrange
5     const playlist = await prisma.playlist.create({
6       data: {
7         name: faker.lorem.words(3),
8         description: faker.lorem.words(),
9         type: "CUSTOM",
10        visibility: "FOLLOWERS_ONLY",
11        user: {
12          connect: { id: mockAdmin.id },
13        },
14      },
15    });
16    ...
17 // END_COPILOT_CODE
```

Σχήμα 2.4: Παράδειγμα κώδικα που παρήχθη από το μοντέλο, με ορθογραφικό λάθος

Η διόρθωση λαθών από τον προγραμματιστή αποτέλεσε έσχατη λύση, καθώς πρώτη επιλογή ήταν η αποφυγή συγγραφής κώδικα από τον προγραμματιστή, μέσω επισύμανσης των λαθών στο μοντέλο, με την προσδοκία ότι το μοντέλο θα διόρθωνε το λάθος χωρίς την ανάγκη επέμβασης του προγραμματιστή. Τα λάθη που διορθώθηκαν από το μοντέλο μετά την προτροπή του προγραμματιστή σημειώθηκαν με σχόλια με πρόθημα *REWRITE*, για αλλαγές σε μία σειρά κώδικα, ενώ με σχόλια με πρόθημα *REVISION* για αλλαγές σε μικρά κομμάτια κώδικα.

```
● ● ●  
1  data: {  
2    // REVISION_1: Correctly connect the user and game to the review  
3    // content: input.content,  
4    // rating: input.rating,  
5    // userId: ctx.user.id,  
6    // userId: ctx.session.user.id,  
7    // gameId: input.gameId,  
8    ...rest,  
9    createdAt: new Date(),  
10   updatedAt: new Date(),  
11   user: {  
12     connect: {  
13       id: ctx.session.user.id,  
14     },  
15   },  
16   game: {  
17     connect: {  
18       id: gameId,  
19     },  
20   },  
21   ...
```

Σχήμα 2.5: Παράδειγμα διόρθωσης λάθους από το μοντέλο μεγάλης έκτασης, μετά την προτροπή του προγραμματιστή



```
1  // REWRITE_1: remove deprecated image field  
2  //   image: faker.image.imageUrl(),  
3  image: faker.image.url(),
```

Σχήμα 2.6: Παράδειγμα διόρθωσης λάθους από το μοντέλο μικρής έκτασης, μετά την προτροπή του προγραμματιστή

Κατά την διόρθωση λαθών, ο προγραμματιστής άφηνε σχόλια, εξηγώντας το λόγο που παρενέβη, με σκοπό την καλύτερη δυνατή τεκμηρίωση, αλλά και την εκμάθηση του μοντέλου, με σκοπό την αποφυγή μελλοντικών λαθών, σημειώνοντας τα κομμάτια κώδικα με τα σχόλια *BEGIN_NON_COPILOT_CODE / END_NON_COPILOT_CODE*.

```

1 // BEGIN_NON_COPILOT_CODE
2 // Explicit prisma relations are hard to work with, copilot struggles with them
3 getGames: publicProcedure
4   .input(z.object({ id: z.string().cuid2() }))
5   .query(
6     async ({
7       ctx,
8       input,
9     }): Promise<
10    Result<Array<GameToPlatform & { game: Game }>, TRPCError>
11  > => {
12    const result: Result<
13      Array<GameToPlatform & { game: Game }>,
14      TRPCError
15    > = await ctx.prisma.platform
16      .findUnique({
17        where: {
18          id: input.id,
19        },
20        select: {
21          games: {
22            include: {
23              game: true,
24            },
25          },
26        },
27      })
28    ...
29 // END_NON_COPILOT_CODE

```

Σχήμα 2.7: Παράδειγμα διόρθωσης λάθους από τον προγραμματιστή, μετά την αδυναμία του μοντέλου να διορθώσει το λάθος

Λάθη που διορθώθηκαν μέσω μιας γραμμής (όπως ονομασίες μεταβλητών ή συμβόλων), σημειώθηκαν με σχόλια με πρόθημα *MANUAL_REWRITE*. Επίσης, κατά την χειροκίνητη συγγραφή κώδικα, το μοντέλο του GitHub Copilot, έδινε στον προγραμματιστή προτάσεις διόρθωσης του κώδικα. Ορισμένες προτάσεις που έλυναν το πρόβλημα, ενσωματώθηκαν στον τελικό κώδικα, σημειώνοντας την αρχή και το τέλος του κώδικα που παρήχθη σε πραγματικό χρόνο, μέσω σχολίων *BEGIN_COPILOT_SUGGESTION* / *END_COPILOT_SUGGESTION*, σημειώνοντας την αρχή και το τέλος του προτεινόμενου κομματιού κώδικα.



```
1 // MANUAL_REWRITE: correct rating type
2 // rating: faker.datatype.number({ min: 1, max: 5 }),
3 rating: faker.number.int({ min: 1, max: 5 }),
```

Σχήμα 2.8: Παράδειγμα διόρθωσης λάθους μικρής έκτασης από τον προγραμματιστή, μετά την αδυναμία του μοντέλου να διορθώσει το λάθος

2.2.2 Ανάλυση Δεδομένων

Μετά την συλλογή των δεδομένων, ακολούθησε η ανάλυσή των ευρημάτων, μέσω ενός αναλυτή (lexer [12]), προκειμένου να μετρηθούν τα εξής:

- Ο αριθμός των περιστατικών επέμβασης του προγραμματιστή σε μεγάλες εκτάσεις κώδικα
- Ο αριθμός των περιστατικών επέμβασης του προγραμματιστή σε μικρές εκτάσεις κώδικα
- Ο αριθμός των περιστατικών που το μοντέλο διόρθωσε το λάθος σε μικρές εκτάσεις κώδικα
- Ο αριθμός των περιστατικών που το μοντέλο διόρθωσε μεγαλύτερα λάθη, έπειτα από προτροπή του προγραμματιστή

Τύπος	Μικροαλλαγές Copilot	Μικροαλλαγές προγραμματιστή
Backend	20	3
Integration Tests	26	7
Unit Tests	32	3

Πίνακας 2.1: Μικροαλλαγές ανά τύπο

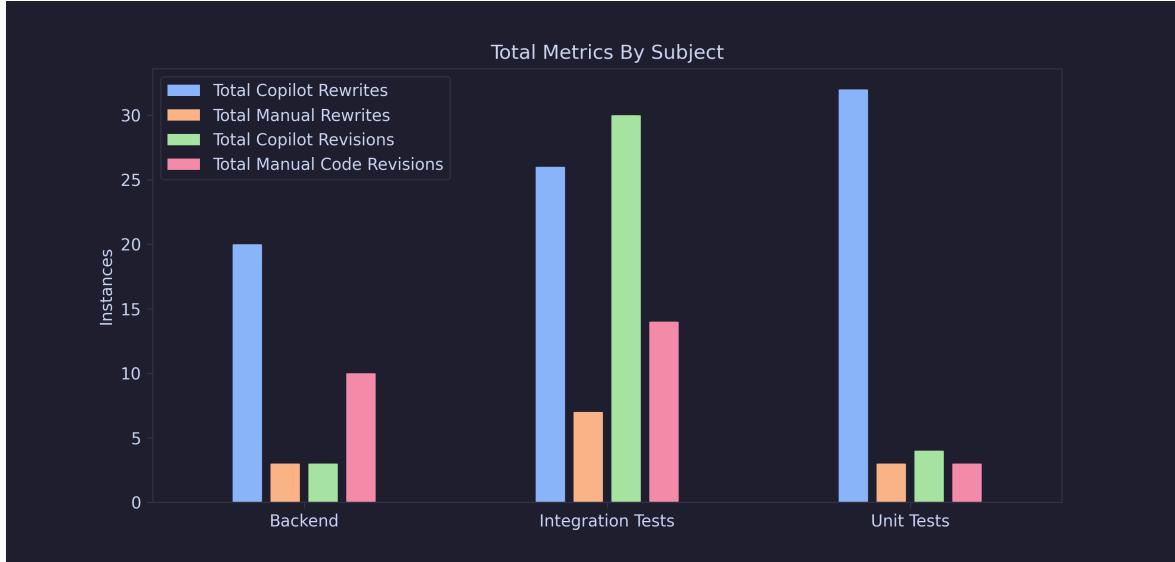
Τύπος	Αλλαγές Copilot	Αλλαγές προγραμματιστή
Backend	3	10
Integration Tests	30	14
Unit Tests	4	3

Πίνακας 2.2: Αλλαγές μεγάλης έκτασης ανά τύπο

Τα δεδομένα λήφθηκαν μέσα από εικοσιοκτώ (28) αρχεία και πεντακόσιες εικοσιπέντε (525) προτροπές και συγκεκριμένα:

- Δεκαεπτά (17) αρχεία που αφορούν τον έλεγχο του κώδικα
- Έντεκα (11) αρχεία που αφορούν την ανάπτυξη του API

Παρατίθεται ένα διάγραμμα που αναπαριστά τα αποτελέσματα της ανάλυσης:



Σχήμα 2.9: Συνολικά μετρικά ανά τύπο

Όπως φαίνεται από το διάγραμμα ;;, οι περισσότερες αλλαγές που πραγματοποιήθηκαν από την πλευρά του μοντέλου έγιναν σε μικρές εκτάσεις κώδικα, ενώ οι περισσότερες αλλαγές που πραγματοποιήθηκαν από την πλευρά του προγραμματιστή έγιναν σε μεγάλες εκτάσεις κώδικα. Σε κάθε τύπο αρχείου παρατηρείται αυτή η τάση, με τον προγραμματιστή να χρειάζεται να επεμβεί πολύ σπανιότερα σε σχέση με το μοντέλο και συγκεκριμένα, σε κάθε τύπο αρχείου, οι επεμβάσεις του προγραμματιστή για μικρές εκτάσεις κώδικα είναι οι λιγότερες.

Αξιοσημείωτη επίσης η παρατήρηση ότι στην περίπτωση των αρχειών που αφορούσαν το Unit Testing, οι αλλαγές σε μικρή έκταση κώδικα από το μοντέλο ήταν με διαφορά οι περισσότερες, ενώ παράλληλα οι υπόλοιπες αλλαγές ήταν ελάχιστες. Αυτό μπορεί να οφείλεται στη χρήση βιβλιοθηκών, με βάση τον έλεγχο του κώδικα και πιο συγκεκριμένα αυτή της faker.js[1]. Η συγκεκριμένη βιβλιοθήκη αφορά την δημιουργία ψεύτικων δεδομένων, στο ίδιο πνεύμα με το Lorem Ipsum [4], αλλά για δεδομένα που αφορούν ένα λογισμικό, όπως ημερομηνίες, ονόματα, διευθύνσεις ηλεκτρονικού ταχυδρομίου κ.α.. Η βιβλιοθήκη έχει λάβει πολλές εκδόσεις τα τελευταία χρόνια, με αλλαγές στον τρόπο που καλούνται οι μέθοδοι του API της, με αποτέλεσμα το μοντέλο να μην μπορεί να ακολουθήσει πάντοτε τις αλλαγές αυτές. Σημαντικό να σημειωθεί επίσης ότι το μοντέλο αδυνατούσε να παρατηρήσει τις αλλαγές που ζητάθηκαν από τον προγραμματιστή, με αποτέλεσμα να ζητηθεί μέσω διαφορετικής προτροπής η αλλαγή της ίδιας κλήσης μιας μεθόδου πάνω από μία φορά στο ίδιο αρχείο.

Στην περίπτωση των αρχείων που αφορούσαν το Backend, οι απαιτούμενες αλλαγές ήταν οι λίγοτερες σε πλήθος, με τη διαφορά ότι οι περισσότερες επεμβάσεις του προγραμματιστή αφορούσαν την γραφή ολόκληρων μεθόδων. Η απουσία των απαραίτητων αλλαγών, οφείλεται στο γεγονός ότι η βασική λογική πίσω από ένα μεγάλο ποσοστό των μεθόδων αφορά την γραφή βασικών CRUD μεθόδων, οι οποίες είναι προκαθορισμένες και δεν απαιτούν αλλαγές. Αυτό έχει ως αποτέλεσμα το μοντέλο να μην χρειάζεται να προτρέπεται να αλλάξει την λογική των μεθόδων, αλλά μόνο την υλοποίηση τους. Στις

περιπτώσεις μεθόδων που επηρέαζαν οντότητες με σχέσεις M:N [58], το μοντέλο του GitHub Copilot αδυνατούσε να ακολουθήσει την λογική, με αποτέλεσμα να χρειαστεί να υλοποιηθούν από τον προγραμματιστή.

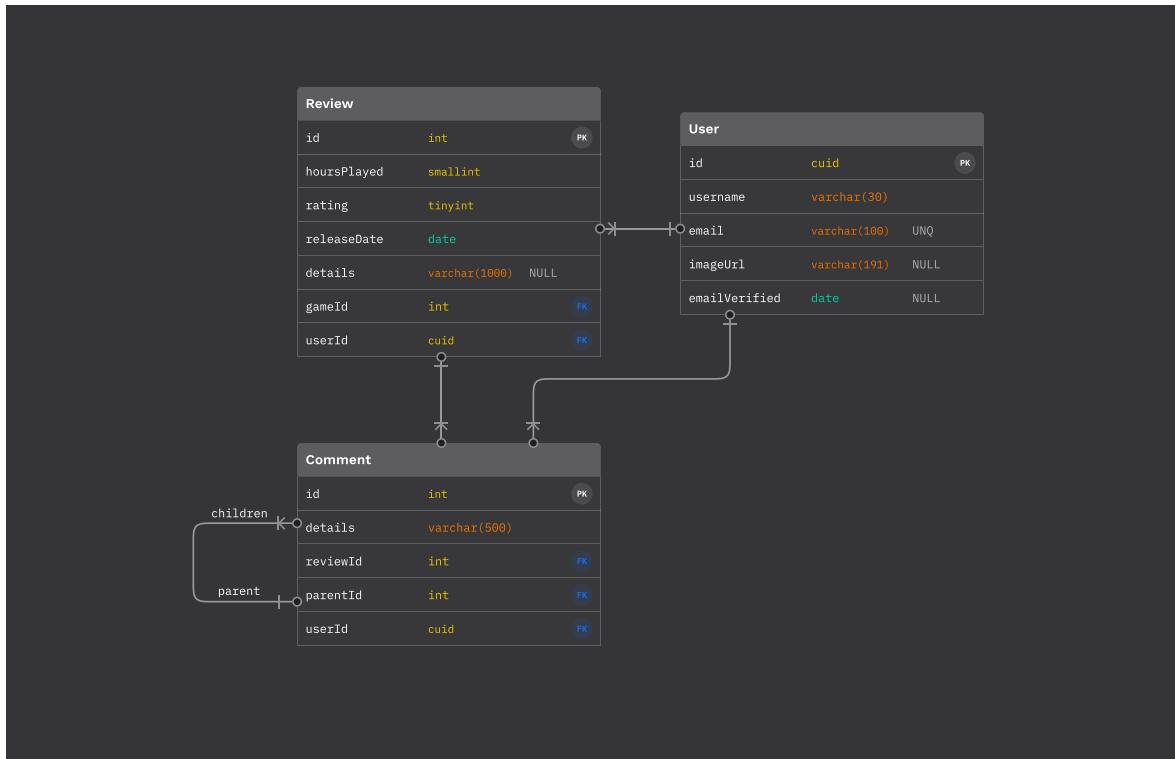
Παράδειγμα αποτελεί η περίπτωση της οντότητας του σχολίου. Τα σχόλια πραγματοποιούνται σε κριτικές των χρηστών, παρομοίως όπως σε άλλες εφαρμογές κοινωνικών δικτύων. Τα σχόλια πραγματοποιούνται σε κριτικές των χρηστών, παρομοίως όπως σε άλλες εφαρμογές κοινωνικών δικτύων. Κάθε σχόλιο έχει μια σχέση 1:M με την κριτική, μια σχέση 1:M με τον χρήστη, μια σχέση M:1 με τις αντιδράσεις τύπου Mou αρέσει (Likes), αλλά μπορεί να έχει και άλλα σχόλια που συνδέονται με αυτό, δημιουργώντας ένα νήμα (thread) σχολίων, παρόμοιο με έναδιατεταγμένο K-οστο δέντρο [27]. Η λογική πίσω από την υλοποίηση των μεθόδων που αφορούν τα σχόλια, είναι περίπλοκη, με τα ερωτήματα που απαιτούνται να σταλθούν προς στη βάση μέσω του Prisma να είναι πολλά και περίπλοκα. Το μοντέλο του GitHub Copilot αδυνατούσε να ακολουθήσει την λογική, με αποτέλεσμα να χρειαστεί να υλοποιηθούν από τον προγραμματιστή.

```

1 const baseSelectClause = Prisma.validator<Prisma.CommentSelect>()({
2   _count: {
3     select: {
4       likes: true,
5       children: false,
6     },
7   },
8   id: true,
9   content: true,
10  user: {
11    select: {
12      image: true,
13      id: true,
14      name: true,
15    },
16  },
17  reviewId: true,
18  createdAt: true,
19  updatedAt: true,
20 });
21
22 const parentSelectClause = {
23   ...baseSelectClause,
24   // Don't take the children (this is where the current comment is)
25   children: false,
26   // Don't take the parent (this is two depths above)
27   parent: false,
28 };
29
30 const childrenSelectClause = {
31   // Don't take the parent (current comment)
32   ...baseSelectClause,
33   parent: false,
34   // Take the first depth of children
35   children: {
36     // Take the second depth of children, and no more than that
37     select: parentSelectClause,
38   },
39 };

```

Σχήμα 2.10: Παράδειγμα μεθόδου που αφορά τα σχόλια, με πολλαπλές σχέσεις M:N



Σχήμα 2.11: Σχεσιακό διάγραμμα της οντότητας του σχολίου

Στη περίπτωση της οντότητας της λίστας (Playlist), η λογική της λειτουργικότητας αυτής είναι να δίνει στο χρήστη τη δυνατότητα να δημιουργεί λίστες με τα αγαπημένα του βιντεοπαιχνίδια. Ο κάθε χρήστης έχει κατά την εγγραφή του στην εφαρμογή πέντε (5) λίστες:

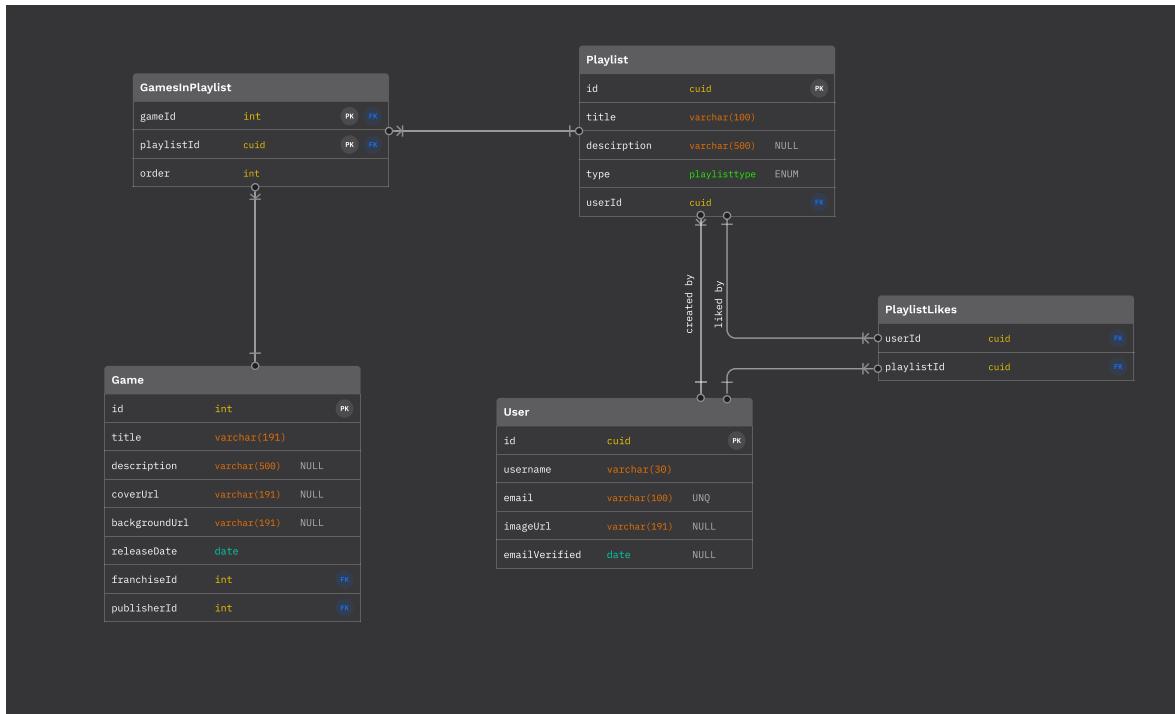
- Αγαπημένα (Liked): Τα βιντεοπαιχνίδια που άρεσαν στον χρήστη, η πιο εύχρηστη λίστα για τον χρήστη
- Λίστα αναμονής (Backlog): Τα βιντεοπαιχνίδια που θέλει να παίξει στο μέλλον
- Ολοκληρωμένα (Completed): Τα βιντεοπαιχνίδια που έχει παίξει και φτάσει στο τέλος τους στο παρελθόν παίξει στο μέλλον
- Ενεργά (Playing): Τα βιντεοπαιχνίδια που παίζει αυτή τη χρονική περίοδο
- Εγκαταλειμμένα (Dropped): Τα βιντεοπαιχνίδια που δεν του άρεσαν και τα έχει εγκαταλείψει

Ο χρήστης έχει το δικαίωμα να δημιουργήσει όσες ακόμα προσαρμοσμένες (Custom) λίστες θέλει, και να τις μοιραστεί είτε με τους ακολούθους του, είτε να τις κάνει δημόσιες, είτε να τις κρατήσει ως ιδιωτικές.

```
1 export const createPlaylistSchema = z.object({
2   name: z.string(),
3   description: z.string().optional(),
4   type: z.enum([
5     "BACKLOG",
6     "LIKED",
7     "COMPLETED",
8     "PLAYING",
9     "DROPPED",
10    "CUSTOM",
11  ]),
12  visibility: z.enum(["PUBLIC", "PRIVATE", "FOLLOWERS_ONLY"]),
13});
```

Σχήμα 2.12: Τύπος δεδομένων για την δημιουργία της οντότητας της λίστας

Κάθε λίστα έχει μια σχέση 1:M με τον χρήστη, μια σχέση 1:M με τις αντιδράσεις, και μία σχέση N:M με βιντεοπαιχνίδια. Η σχέση με τα βιντεοπαιχνίδια υλοποιείται μέσω ενός πίνακα σύνδεσης [11], ο οποίος, πέρα από τα ξένα κλειδιά του βιντεοπαιχνιδιού και της λίστας, περιέχει και το πεδίο order, το οποίο καθορίζει τη σειρά με την οποία τα βιντεοπαιχνίδια εμφανίζονται στη λίστα. Η λογική πίσω από την υλοποίηση προϋπόθεται τη χρήση μιας διπλά συνδεμένης λίστας, προκειμένου να διατηρείται η επιθυμητή σειρά, αν ο χρήστης αποφασίσει να αλλάξει τη σειρά με την οποία τα βιντεοπαιχνίδια θα εμφανίζονται στη λίστα.



Σχήμα 2.13: Σχεσιακό διάγραμμα της οντότητας της λίστας

Το μοντέλο του GitHub Copilot αδυνατούσε να ακολουθήσει την λογική, με αποτέλεσμα να χρειαστεί να υλοποιηθούν εξ ολοκλήρου από τον προγραμματιστή.

```

1 updateOrder: protectedProcedure
2   .input(
3     z.object({
4       id: z.string().cuid2(),
5       gameId: z.string().cuid2(),
6       order: z.number().int(),
7     })
8   )
9   .mutation(
10    async ({
11      ctx,
12      input,
13    }): Promise<Result<Playlist & { games: Array<GameToPlaylist & { game: Game }> }, TRPCError>> => {
14      const whereClause = Prisma.validator<Prisma.PlaylistWhereInput>()({
15        id: input.id,
16        deleted: null,
17        user: { id: ctx.session.user.id },
18      });
19
20      const oldOrder = await ctx.prisma.gameToPlaylist
21        .findUnique({
22          where: {
23            gameId_playlistId: {
24              gameId: input.gameId,
25              playlistId: input.id,
26            },
27          },
28          select: { order: true },
29        })
30        .then((res) => Ok(res), handlePrismaError);
31
32      if (!oldOrder.ok) return oldOrder;
33
34      const increment = ctx.prisma.playlist.update({
35        where: whereClause,
36        data: {
37          games: {
38            updateMany: {
39              where: {
40                order: {
41                  gte: input.order,
42                },
43              },
44              data: {
45                order: {
46                  increment: 1,
47                },
48              },
49            },
50          },
51        },
52      });
53

```

Σχήμα 2.14: Παράδειγμα μεθόδου που αφορά την αλλαγή της σειράς των βιντεοπαιχνιδιών στη λίστα, *Μέρος 1*

```

1  const decrement = ctx.prisma.playlist.update({
2    where: whereClause,
3    data: {
4      games: {
5        updateMany: {
6          where: {
7            order: {
8              lte: input.order,
9            },
10           },
11           data: {
12             order: {
13               decrement: 1,
14             },
15           },
16           },
17         },
18       },
19     });
20
21 const update = ctx.prisma.playlist.update({
22   where: whereClause,
23   data: {
24     games: {
25       update: {
26         where: {
27           gameId_playlistId: {
28             gameId: input.gameId,
29             playlistId: input.id,
30           },
31         },
32         data: {
33           order: input.order,
34         },
35       },
36     },
37   },
38   include: {
39     games: {
40       orderBy: {
41         order: "asc",
42       },
43       include: {
44         game: true,
45       },
46     },
47   },
48 });
49
50 switch (true) {
51   case oldOrder instanceof Ok && oldOrder.val instanceof Object && typeof oldOrder.val.order === "number":
52     const result: Result<Playlist & { games: Array<GameToPlaylist & { game: Game }> }, TRPCError> =
53       oldOrder.val.order > input.order
54         ? await ctx.prisma.$transaction([increment, update]).then(([res] => Ok(res[1]), handlePrismaError))
55         : await ctx.prisma.$transaction([decrement, update]).then(([res] => Ok(res[1]), handlePrismaError));
56     return result;
57   default:
58     return new Err(new TRPCError({ code: "INTERNAL_SERVER_ERROR" }));
59   }
60 }
61 ),

```

Σχήμα 2.15: Παράδειγμα μεθόδου που αφορά την αλλαγή της σειράς των βιντεοπαιχνιδιών στη λίστα, *Μέρος 2*

Πέρα από τη σειρά των βιντεοπαιχνιδιών, σημαντική ήταν και η εξασφάλιση της ορθής ορατότητας της λίστας σε άλλους χρήστες, χωρίς να επιτρέπεται η πρόσθιαση σε λίστες που δεν είναι είτε δημόσιες, είτε ανήκουν σε χρήστες που ο χρήστης ακολουθεί, είτε είναι προσωπικές λίστες του ίδιου. Στην περίπτωση αυτή, αν και το μοντέλο ακολούθησε τη σωστή λογική, αδυνάτισε να βρει το σωστό ερώτημα προς τη βάση όσο αφορά την ορατότητα της λίστας προς τους ακολούθους του χρήστη.

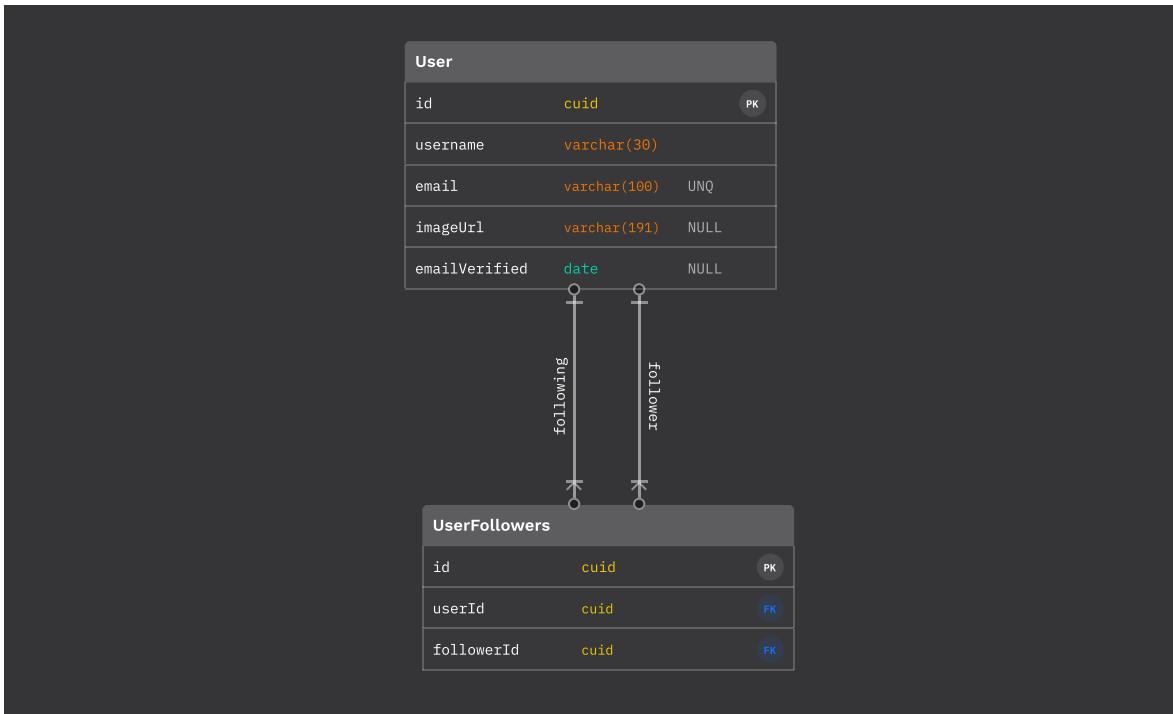
```

1  getById: publicProcedure
2    .input(z.object({ id: z.string().cuid2() }))
3    .query(async ({ ctx, input }): Promise<Result<Playlist, TRPCError>> => {
4      const whereClause: Prisma.PlaylistWhereInput = {
5        id: input.id,
6        deleted: null,
7        OR: [{ visibility: "PUBLIC" }],
8      };
9
10     if (ctx.session?.user?.id) {
11       (whereClause.OR as Prisma.PlaylistWhereInput[]).push(
12         // REWRITE_2: use user relation instead of userId
13         // {userId: ctx.session.user.id},
14         { user: { id: ctx.session.user.id } },
15         {
16           visibility: "FOLLOWERS_ONLY",
17           // END_COPILOT_CODE
18           // BEGIN_NON_COPILOT_CODE
19           // Copilot didn't find the bug
20           // user: { followers: { some: { id: ctx.session.user.id } } },
21           user: {
22             followers: { some: { follower: { id: ctx.session.user.id } } },
23           },
24           // END_NON_COPILOT_CODE
25         }
26       );
27     }
28
29     const result: Result<Playlist, TRPCError> = await ctx.prisma.playlist
30       // REWRITE_1: use findfirst instead of findUnique
31       // .findUnique({ where: whereClause })
32       .findFirst({ where: whereClause })
33       .then((res) => {
34         return res
35         ? Ok(res)
36         : new Err(
37           new TRPCError({
38             code: "NOT_FOUND",
39             message: "Playlist not found",
40           })
41         );
42       }, handlePrismaError);
43
44     return result;
45   },

```

Σχήμα 2.16: Παράδειγμα μεθόδου που αφορά την ανάκτηση λίστας με βάση το αναγνωριστικό της

Όπως παρατηρείται στο διάγραμμα ;;, το μοντέλο ορθά αναγνώρισε ότι σε περίπτωση που μια λίστα δημοσιεύονταν μόνο στους ακολούθους του δημιουργού της, θα έπρεπε ο χρήστης που την ανακτά θα έπρεπε να ανήκει στην λίστα των ακολούθων του δημιουργού, η υλοποίηση του κώδικα δεν ήταν σωστή, καθώς αγνόησε την M:N σχέση μεταξύ των χρηστών και των ακολούθων τους.



Σχήμα 2.17: Σχεσιακό διάγραμμα της σχέσης μεταξύ χρηστών και ακολούθων

Στις περιπτώσεις των αρχείων που αφορούσαν το Integration Testing, οι αλλαγές ήταν οι περίσσοτερες συγκριτικά με τις περιπτώσεις ανάπτυξης του API και του Unit Testing και οι αλλαγές από την πλευρά του μοντέλου ήταν αρκετά πιο ισορροπημένες, με τις αλλαγές σε μεγάλη έκταση κώδικα να υπερβαίνουν αυτών της μικρής έκτασης κώδικα.

Στην περίπτωση αυτή, ρόλο έπαιξε το γεγονός ότι κατά την γραφή των ελέγχων, το εργαλείο Jest, παρέχει τη δυνατότητα στον προγραμματιστή να συντάξει μια περιγραφή για τον έλεγχο, μέσω των μεθόδων *describe* και *it*. Οι μέθοδοι αυτές, παίρνουν ως πρώτο όρισμα μια συμβολοσειρά που περιγράφει τον έλεγχο, και ως δεύτερο όρισμα μια συνάρτηση *callback* [21] που περιέχει τον κώδικα του ελέγχου. Η περιγραφή του ελέγχου, ακολουθεί την ανάπτυξη **Λογισμικού Καθοδηγούμενη Από Τη Συμπεριφορά** (Behavior Driven Development, BDD).

Η BDD είναι μια προσέγγιση στην ανάπτυξη λογισμικού που επικεντρώνεται στη συμπεριφορά του συστήματος από την οπτική γωνία του χρήστη. Ενθαρρύνει τη συνεργασία μεταξύ προγραμματιστών, ελεγκτών και μη τεχνικών ενδιαφερομένων, χρησιμοποιώντας μια κοινή γλώσσα για να περιγράψει την επιθυμητή συμπεριφορά του λογισμικού. Συνήθως περιλαμβάνει τη συγγραφή σεναρίων ή παραδειγμάτων που περιγράφουν πώς το λογισμικό θα πρέπει να συμπεριφέρεται σε διάφορες καταστάσεις, συχνά χρησιμοποιώντας τη μορφή 'Δεδομένου-Όταν-Τότε' ('Given-When-Then'). Αυτά τα σενάρια λειτουργούν τόσο ως προδιαγραφές όσο και ως περιπτώσεις δοκιμών, βοηθώντας να διασφαλιστεί ότι το αναπτυγμένο λογισμικό ανταποκρίνεται στις προβλεπόμενες απαιτήσεις και συμπεριφέρεται σωστά από την οπτική γωνία του χρήστη. [64, 48, 23]



```
1 describe("When creating a feature", () => {
2   describe("and the user is not authenticated", () => {
3     it("should throw an error", async () => {
4       ...

```

Σχήμα 2.18: Παράδειγμα μεθόδου που αφορά την δημιουργία ενός ελέγχου, με χρήση των μεθόδων *describe* και *it*

```
PASS __tests__/api/game.test.ts
When creating a game
  and the user is not authenticated
    ✓ should throw an error (6 ms)
  and the user is authenticated
    and the user is not an admin
      ✓ should throw an error (1 ms)
    and the user is an admin
      and either the publisher or the franchise don't exist
        ✓ should return an error (4 ms)
    and the publisher and franchise exist
      ✓ should create a game successfully (3 ms)
```

Σχήμα 2.19: Παράδειγμα αποτελέσματος ελέγχου σε διεπαφή τερματικού, με χρήση των μεθόδων *describe* και *it*, μέσω του εργαλείου Jest

Επειδή το μοντέλο δουλεύει με next token prediction, και έλεγχοι γράφονται ο ένας μετά τον άλλο, το μοντέλο κατά τη συγγραφή των ελέγχων, λόγω των περιγραφών που περιέχονται στις μεθόδους, προσπαθούσε να γράψει τον έλεγχο με βάση την περιγραφή, αντί του κάθικα της μεθόδου, με αποτέλεσμα να οδηγείται συχνά σε λάθη και ανακρίβειες. Αξιοσημείωτη είναι η προσθήκη της δυνατότητας του GitHub Copilot Chat να γράψει ελέγχους για μια μέθοδο μέσω μιας εντολής στο πλαίσιο προτροπής με το μοντέλο [65] τον Μάρτιο του 2024, ωστόσο κατά την διάρκεια της δοκιμής του μοντέλου, η συγκεκριμένη δυνατότητα δεν ήταν διαθέσιμη.

Για την διαφορά μεταξύ αναγκαίων αλλαγών στις περιπτώσεις των αρχείων που αφορούσαν το Unit Testing και το Integration Testing, συνέβαλε και το γεγονός στην περίπτωση των ελέγχων μονάδας, οι έλεγχοι που πραγματοποιούνται είναι πιο απλοί και πολλές φορές χρησιμοποιούνται τεχνικές εικονικής αναπαράστασης εξαρτήσεων, γνωστές και ως τεχνικές προσομοίωσης εξαρτήσεων (dependency mocking). Η προσομοίωση εξαρτήσεων είναι μια τεχνική που χρησιμοποιείται στους ελέγχους λογισμικού, όπου δημιουργούνται ψεύτικα ή προσομοιωμένα αντικείμενα που μιμούνται τη συμπεριφο-

ρά πραγματικών εξαρτήσεων ενός συστήματος. Αυτό επιτρέπει στους προγραμματιστές να ελέγχουν μια μονάδα κάθικα σε απομόνωση, χωρίς να επηρεάζονται από εξωτερικές εξαρτήσεις όπως βάσεις δεδομένων, υπηρεσίες δικτύου ή πολύπλοκες βιβλιοθήκες. Η προσομοίωση εξαρτήσεων βοηθά στη δημιουργία πιο ελεγχόμενων και προβλέψιμων συνθηκών δοκιμής, επιτρέποντας τον έλεγχο διαφόρων σεναρίων και καταστάσεων σφάλματος που μπορεί να είναι δύσκολο να αναπαραχθούν με πραγματικές εξαρτήσεις [25]. Αντίθετα, στην περίπτωση των ελέγχων ενσωμάτωσης, οι εξαρτήσεις είναι πραγματικές, και οι ελέγχοι που πραγματοποιούνται είναι πιο πολύπλοκοι και απαιτούν την ύπαρξη πραγματικών εξαρτήσεων, και χρειάζονται περισσότερη σκέψη για την υλοποίησή τους. Παρατίθενται δύο παραδείγματα ελέγχων, ένα από την περίπτωση των ελέγχων μονάδας και ένα από την περίπτωση των ελέγχων ενσωμάτωσης, για τον ίδιο έλεγχο.

```
1 describe("When creating a game", () => {
2   describe("and the user is authenticated", () => {
3     describe("and the user is an admin", () => {
4       describe("and the publisher and franchise exist", () => {
5         it("should create a game successfully", async () => {
6           // Arrange
7           const caller = appRouter.createCaller({
8             prisma: mockCtx.prisma,
9             session: mockAdminSession,
10            });
11
12           const gameId = createId();
13           const franchiseId = createId();
14
15           const gameData = {
16             name: faker.company.name(),
17             description: faker.lorem.words(),
18             coverImage: faker.image.url(),
19             backgroundImage: faker.image.url(),
20             releaseDate: new Date(),
21             franchiseId: franchiseId,
22             publisherId: gameId,
23           };
24
25           mockCtx.prisma.game.create.mockResolvedValue({
26             ...gameData,
27             id: gameId(),
28           });
29
30           // Act
31           const result = await caller.game.create(gameData);
32
33           // Assert
34           expect(result.ok).toBe(true);
35           expect(mockCtx.prisma.game.create).toHaveBeenCalledTimes(1);
36           expect(mockCtx.prisma.game.create).toHaveBeenCalledWith({
37             data: {
38               name: gameData.name,
39               description: gameData.description,
40               coverImage: gameData.coverImage,
41               backgroundImage: gameData.backgroundImage,
42               releaseDate: gameData.releaseDate,
43               franchise: {
44                 connect: {
45                   id: gameData.franchiseId,
46                 },
47               },
48               publisher: {
49                 connect: {
50                   id: gameData.publisherId,
51                 },
52               },
53             },
54           });
55         });
56       });
57     });
58   });
59 });
```

Σχήμα 2.20: Παράδειγμα ελέγχου μονάδας για την επιτυχημένη δημιουργία παιχνιδιού



```
1  describe("When creating a game", () => {
2      describe("and the user is an admin", () => {
3          describe("and the franchise and publisher exist", () => {
4              it("should create a game", async () => {
5                  // Arrange
6                  const franchise = await prisma.franchise.create({
7                      data: {
8                          name: faker.company.name(),
9                          // REWRITE_2: add description and image
10                         description: faker.company.catchPhrase(),
11                         image: faker.image.url(),
12                         },
13                     });
14
15                  const publisher = await prisma.publisher.create({
16                      data: {
17                          name: faker.company.name(),
18                          // REWRITE_2: add description and image
19                          description: faker.company.catchPhrase(),
20                          image: faker.image.url(),
21                          },
22                     });
23
24                  const game: z.infer<typeof createGameSchema> = {
25                      name: faker.company.name(),
26                      description: faker.company.catchPhrase(),
27                      coverImage: faker.image.url(),
28                      backgroundImage: faker.image.url(),
29                      releaseDate: new Date(),
30                      franchiseId: franchise.id,
31                      publisherId: publisher.id,
32                  };
33
34                  // Act
35                  const result = await adminCaller.game.create(game);
36
37                  // Assert
38                  expect(result.ok).toBe(true);
39                  expect(result.val).toMatchObject(game);
40              });
41          });
42      });
43  });
```

Σχήμα 2.21: Παράδειγμα ελέγχου συμαδότητας για την επιτυχημένη δημιουργία παιχνιδιού

Εκ πρώτης όψεως, ο κώδικας στο Σχήμα :: φαίνεται να είναι πιο περίπλοκος αυτού στο Σχήμα ::, ωστόσο, στην περίπτωση του ελέγχου μονάδας, οι εξαρτήσεις είναι προσομοιωμένες. Δεν υπάρχει κάποιος πραγματικός χρήστης, ή κάποια πραγματική βάση δεδομένων. Αντίθετα, στην περίπτωση του ελέγχου συμβατότητας, οι εξαρτήσεις είναι πραγματικές, και ο έλεγχος πρέπει να ελέγξει την συμβατότητα του συστήματος με μια πραγματική βάση δεδομένων, ένα πραγματικό χρήστη και μετά από τον κάθε έλεγχο, πρέπει να είναι σίγουρο ότι εσωτερική κατάσταση του προγράμματος και της βάσης είναι κοινή για όλες τις περιπτώσεις, αποφεύγοντας περιπτώσεις ασταθών δοκιμών (flaky tests), όπου το αποτέλεσμα της δοκιμής δεν είναι πάντοτε αμετάβλητο. [55]

2.3 Αξιολόγηση Απαντήσεων

Κατά την αλληλεπίδραση με το μοντέλο, για κάθε απάντηση που δόθηκε έπειτα προτροπής, πραγματοποιήθηκε μια αξιολόγηση σε μια κλίμακα μεταξύ μείον δύο (-2) και δύο (2).

2.4 Συμπεράσματα

Όντας μια καινοτόμος επιλογή τεχνολογικής στοίβας, η χρήση της Typescript, σε συνδυασμό με το tRPC και το Prisma, αποτέλεσε μια πρόκληση για το μοντέλο του GitHub Copilot. Η επιλογή αυτή έγινε με σκοπό την ορθότερη αξιολόγηση της ικανότητας του μοντέλου να ακολουθήσει την ταχεία εξέλιξη των τεχνολογιών και συγκεκριμένα στο οικοσύστημα της Javascript, καθώς έχει συνεχείς προσθήκες και αλλαγές στον τρόπο που γράφεται ο κώδικας και επίσης αποτελεί τη βασική προγραμματιστική γλώσσα του διαδικτύου.

Εκ του αποτελέσματος των δοκιμών, η ιδανικότερη χρήση του μοντέλου του GitHub Copilot φαίνεται είναι να ακολουθήσει τις επιλογές του προγραμματιστή για την ανάπτυξη του κώδικα, την αρχιτεκτονική, τα πρότυπα σχεδίασης και τις βιβλιοθήκες που θα χρησιμοποιηθούν κατά την ανάπτυξη της εφαρμογής, όπως και αποδείχθηκε από τα αποτελέσματα στις περιπτώσεις που δόθηκε η ευκαιρία στο μοντέλο να λάβει αποφάσεις, καθώς οι περισσότερες αλλαγές που πραγματοποιήθηκαν από την πλευρά του προγραμματιστή ήταν σε μεγάλες εκτάσεις κώδικα και συγκεκριμένα στις περιπτώσεις των ελέγχων, όπου εκεί το μοντέλο έλαβε πρωτοβουλίες κατά τη σύνταξη των ελέγχων, οδηγώντας σε λάθη.

Παράτημα Α'

Ακρωνύμια και συντομογραφίες

LLM Large Language Model

NLP Natural Language Processing

IDE Integrated Development Environment

LSP Language Server Protocol

API Application Programming Interface

ORM Object Relational Mapping

CRUD Create Read Update Delete

BDD Behavior Driven Development

Bibliography

- [1] “Faker.js.” [Online]. Available: <https://fakerjs.dev/>
- [2] “Github copilot.” [Online]. Available: <https://github.com/features/copilot>
- [3] “Github student developer pack.” [Online]. Available: <https://education.github.com/pack>
- [4] “Lorem ipsum - all the facts - ipsum generator,” <https://www.lipsum.com/>, retrieved June 11, 2024.
- [5] *Guide to the Software Engineering Body of Knowledge*, 2004, a project of the IEEE Computer Society Professional Practices Committee.
- [6] “Trendforce says with cloud companies initiating ai arms race, gpu demand from chatgpt could reach 30,000 chips as it readies for commercialization,” *TrendForce*, Nov 2023, archived from the original on May 30, 2024. Retrieved November 2, 2023.
- [7] “Visual studio code.” [Online]. Available: <https://code.visualstudio.com/>
- [8] S. Adams, I. Arel, J. Bach, R. Coop, R. Furlan, B. Goertzel, J. S. Hall, A. Samsonovich, M. Scheutz, M. Schlesinger *et al.*, “Mapping the landscape of human-level artificial general intelligence,” *AI Magazine*, vol. 33, no. 1, pp. 25–42, 2012.
- [9] A. Ait Baha, M. El Hajji, Y. Es-saady, and H. Fadili, “The power of personalization: A systematic review of personality-adaptive chatbots,” *SN Computer Science*, vol. 4, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:261346287>
- [10] M. Asaduzzaman, C. K. Roy, K. A. Schneider, and D. Hou, “Csc: Simple, efficient, context sensitive code completion,” in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 71–80.
- [11] Atlassian, “Join relationships and joining tables,” <https://www.atlassian.com/data/sql/joins>, retrieved June 11, 2024.
- [12] T. Ball, *Writing An Interpreter In Go*. Thorsten Ball, 2018.
- [13] J. Bays, “Aws announces amazon codewhisperer (preview),” Jun 2022. [Online]. Available: <https://aws.amazon.com/about-aws/whats-new/2022/06/aws-announces-amazon-codewhisperer-preview/>
- [14] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, “On the opportunities and risks of foundation models,” 2021.

- [15] H. Bunder, “Decoupling language and editor - the impact of the language server protocol on textual domain-specific languages,” in *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development - MODELSWARD*, INSTICC. SciTePress, 2019, pp. 131–142.
- [16] T. S. BV, “Tiobe index for javascript,” <https://www.tiobe.com/tiobe-index/javascript/>, retrieved June 01 2024.
- [17] S. Carter, “Exponential baby! navigating the ai convergence of tech with nvidia,” *Forbes Digital Assets*, May 2024, retrieved May 30, 2024. [Online]. Available: <https://www.forbes.com/sites/digital-assets/2024/05/11/exponential-baby-navigating-the-ai--convergence-of-tech-with-nvidia/>
- [18] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, “Evaluating large language models trained on code,” 2021.
- [19] M. Ciniselli, N. Cooper, L. Pascarella, A. Mastropaoletti, E. Aghajani, D. Poshyvanyk, M. Di Penta, and G. Bavota, “An empirical study on the usage of transformer models for code completion,” *IEEE Transactions on Software Engineering*, 2021.
- [20] M. Ciniselli, N. Cooper, L. Pascarella, D. Poshyvanyk, M. Di Penta, and G. Bavota, “An empirical study on the usage of bert models for code completion,” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 108–119.
- [21] D. Crockford, *JavaScript: The Good Parts*, 1st ed., ser. Yahoo Press. Sebastopol, CA: O'Reilly Media, 2008, pp. 36–40.
- [22] T. Dohmke, “Github copilot x: The ai-powered developer experience,” *GitHub Blog*, March 2023, retrieved May 30, 2024. [Online]. Available: <https://github.blog/2023-03-22-github-copilot-x-the-ai-powered-developer-experience/>
- [23] S. Farooq, U. Omer, A. Ramzan, M. Rasheed, and Z. Atal, “Behavior driven development: A systematic literature review,” *IEEE Access*, vol. PP, pp. 1–1, 01 2023.
- [24] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, “Codebert: A pre-trained model for programming and natural languages,” in *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov 2020, pp. 1536–1547.
- [25] S. Freeman and N. Pryce, *Growing Object-Oriented Software, Guided by Tests*. Upper Saddle River, NJ: Addison-Wesley Professional, 2009.

- [26] D. Fried, A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, S. Yih, L. Zettlemoyer, and M. Lewis, “Incoder: A generative model for code infilling and synthesis,” in *The Eleventh International Conference on Learning Representations*, 2023.
- [27] A. Gabillon, M. Munier, J. J. Bascou, L. Gallon, and E. Bruno, “An access control model for tree data structures,” in *Information Security, 5th International Conference, ISC 2002 Sao Paulo, Brazil, September 30 - October 2, 2002, Proceedings*, Sao Paulo, Brazil, 2002, pp. 117–135. [Online]. Available: <https://hal.archives-ouvertes.fr/hal-01912344>
- [28] GitHub, “Introducing github copilot: your ai pair programmer,” *Github Blog*, June 2021, retrieved May 30, 2024. [Online]. Available: <https://github.blog/2021-06-29-introducing-github-copilot-ai-get-code-done-faster/>
- [29] B. Goertzel, “Artificial general intelligence: Concept, state of the art, and future prospects,” *Journal of Artificial General Intelligence*, vol. 5, no. 1, pp. 1–46, 2014, submitted 2013-2-12, Accepted 2014-3-15.
- [30] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, “Unircoder: Unified cross-modal pre-training for code representation,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 7212–7225.
- [31] J. Irena, *Software Testing Methods and Techniques*, 2008.
- [32] M. Izadi, R. Gismondi, and G. Gousios, “Codefill: Multi-token code completion by jointly learning from structure and naming sequences,” in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE ’22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 401–412.
- [33] M. Izadi, J. Katzy, T. van Dam, M. Otten, R. M. Popescu, and A. van Deursen, “Language models for code completion: A practical evaluation,” in *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE ’24)*. New York, NY, USA: ACM, April 14–20 2024, pp. 1–13.
- [34] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley, 1999, vol. 1.
- [35] A. Jamil, M. Arif, N. Abubakar, and A. Ahmad, “Software testing techniques: A literature review,” 11 2016, pp. 177–182.
- [36] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson, 2009.
- [37] S. Kathiriya, R. Karangara, and N. Challa, “Optimizing automated software testing with machine learning techniques,” *International Journal of Science and Research (IJSR)*, vol. 7, pp. 2319–7064, 03 2018.
- [38] J. Kemper. (2023, March) Openai kills its codex code model, recommends gpt3.5 instead. Retrieved May 27, 2024. [Online]. Available: <https://thedecoder.com/openai-kills-its-codex-code-model-recommends-gpt3-5-instead/>

- [39] S. Kim, J. Zhao, Y. Tian, and S. Chandra, “Code prediction by feeding trees to transformers,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 150–162.
- [40] H. H. Koester and S. Levine, “Effect of a word prediction feature on user performance,” *Augmentative and alternative communication*, vol. 12, no. 3, pp. 155–168, 1996.
- [41] Z. Liu, “Chatgpt will command more than 30,000 nvidia gpus: Report,” *Tom’s Hardware*, Mar 2023, archived from the original on May 30, 2024. Retrieved November 2, 2023.
- [42] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, “Codexglue: A machine learning benchmark dataset for code understanding and generation,” in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.
- [43] C. Luo, “A report on automatic code completion,” 03 2017.
- [44] “React,” Meta. [Online]. Available: <https://react.dev/>
- [45] C. Metz and T. Mickle, “Openai completes deal that values the company at \$80 billion,” *The New York Times*, Feb 2024, retrieved May 30, 2024.
- [46] “Typescript,” Microsoft. [Online]. Available: <https://www.typescriptlang.org/>
- [47] E. F. Miller, “Introduction to software testing technology,” in *Software Testing & Validation Techniques*. IEEE, 1981, pp. 4–16.
- [48] A. H. Mughal, “Advancing bdd software testing: Dynamic scenario re-usability and step auto-complete for cucumber framework,” 2024. [Online]. Available: <https://arxiv.org/abs/2402.15928>
- [49] Netscape and Sun, “Netscape and sun announce javascript, the open, cross-platform object scripting language for enterprise networks and the internet,” Press release, 12 1995, retrieved June 01 2024.
- [50] “Nextauth,” NextAuth. [Online]. Available: <https://next-auth.js.org/>
- [51] C. Omar, Y. Yoon, T. LaToza, and B. Myers, “Active code completion,” in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 859–869.
- [52] OpenAI, “Chatgpt: Optimizing language models for dialogue,” 2022. [Online]. Available: <https://web.archive.org/web/20221130180912/https://openai.com/blog/chatgpt/>
- [53] “Jest,” OpenJS Foundation. [Online]. Available: <https://jestjs.io/>
- [54] “Mysql,” Oracle. [Online]. Available: <https://www.mysql.com/>
- [55] O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, “A survey of flaky tests,” *Transactions on Software Engineering and Methodology*, vol. 31, no. 1, 2022.

- [56] R. Patton, *Software Testing*. Indianapolis, IN: Sams Publishing, 2005.
- [57] “Prisma,” Prisma. [Online]. Available: <https://www.prisma.io/>
- [58] Prisma, “Many-to-many relations,” <https://www.prisma.io/docs/concepts/components/prisma-schema/relations/many-to-many-relations>, 2022, retrieved 2024-06-11.
- [59] J. K. Rask, F. P. Madsen, N. Battle, H. D. Macedo, and P. G. Larsen, “The specification language server protocol: A proposal for standardised lsp extensions,” 2022.
- [60] V. Raychev, M. Vechev, and E. Yahav, “Code completion with statistical language models,” in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2014, pp. 419–428.
- [61] R. Rodriguez-Echeverria, J. L. Canovas Izquierdo, M. Wimmer, and J. Cabot, “Towards a language server protocol infrastructure for graphical modeling,” in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS ’18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 370–380.
- [62] E. Roth, “Microsoft spent hundreds of millions of dollars on a chatgpt supercomputer,” *The Verge*, Mar 2023, archived from the original on May 30, 2023. Retrieved March 30, 2023.
- [63] M. Shaw, “Prospects for an engineering discipline of software,” *IEEE Software*, pp. 15–24, Nov 1990.
- [64] C. Solis Pineda and X. Wang, “A study of the characteristics of behaviour driven development,” 10 2011, pp. 383 – 387.
- [65] Staff, “Using copilot chat in vs code,” *Visual Studio Code Documentation*, March 2024, retrieved June 20, 2024.
- [66] ——, “Github copilot – november 30th update,” *GitHub Blog*, November 2023, retrieved May 30, 2024. [Online]. Available: <https://github.blog/changelog/2023-11-30-github-copilot-november-30th-update/>
- [67] ——, “The top programming languages,” 2022, retrieved May 30, 2024. [Online]. Available: <https://octoverse.github.com/2022/top-programming-languages>
- [68] K. Staff, “Microsoft-backed openai valued at \$80bn after company completes deal,” *The Guardian*, Feb 2024, retrieved May 30, 2024.
- [69] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, “Intellicode compose: Code generation using transformer,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1433–1443.
- [70] A. Svyatkovskoy, S. Lee, A. Hadjitofi, M. Riechert, J. Franco, and M. Al-lamanis, “Fast and memory-efficient neural code completion,” *arXiv preprint arXiv:2004.13651*, 2020.
- [71] “create-t3-app,” <https://github.com/t3-oss/create-t3-app>, T3 Open Source, 2022.

- [72] T. V. C. Team, “Visual studio code 1.0!” *Visual Studio Code Blog*, April 2016, retrieved June 1, 2024.
- [73] “trpc,” tRPC. [Online]. Available: <https://trpc.io/>
- [74] Unknown, “Tabnine: Coding in vs code with the help of an ai assistant,” *learn.microsoft.com*, March 2021, retrieved April 19, 2024.
- [75] “Next.js,” Vercel. [Online]. Available: <https://nextjs.org/>
- [76] J. Vincent, “This ai-powered autocompletion software is gmail’s smart compose for coders,” *The Verge*, July 2019, retrieved May 19, 2024.
- [77] Y. Wang and H. Li, “Code completion by modeling flattened abstract syntax trees as graphs,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 14 015–14 023.
- [78] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, “Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.
- [79] B. Xu, A. Yang, J. Lin, Q. Wang, C. Zhou, Y. Zhang, and Z. Mao, “Expertprompting: Instructing large language models to be distinguished experts,” 2023.
- [80] S. Zhao, “Github copilot chat now generally available for organizations and individuals,” *GitHub Blog*, December 2023, retrieved May 30, 2024. [Online]. Available: <https://github.blog/2023-12-29-github-copilot-chat-now-generally-available-for-organizations-and-individuals/>
- [81] S. Z. Zhao, “Smarter, more efficient coding: Github copilot goes beyond codex with improved ai model,” *GitHub Blog*, July 2023, retrieved May 27, 2024. [Online]. Available: <https://github.blog/2023-07-28-smarter-more-efficient-coding-github-copilot-goes-beyond-codex-with-improved>
- [82] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, “A survey of large language models,” 2023.
- [83] Q. Zhou, C. Li, J. Yu, Y. Liu, G. Wang, K. Zhang, C. Ji, Q. Yan, L. He *et al.*, “A comprehensive survey on pretrained foundation models: A history from bert to chatgpt,” *arXiv preprint arXiv:2302.09419*, 2023.
- [84] X. Zhou, H. Zhu, L. Mathur, R. Zhang, H. Yu, Z. Qi, L.-P. Morency, Y. Bisk, D. Fried, G. Neubig, and M. Sap, “Sotopia: Interactive evaluation for social intelligence in language agents,” 2024.